Essays in Information Management:

Contributions to the Modeling and Analysis of Quality in Information Systems Engineering

Ivan J. Jureta

Thèse présentée en vue de l'obtention du grade
de Docteur en Sciences de Gestion.

Department of Business Administration
University of Namur

February 2008

Summary

Efficient organization requires rigorous and systematic information management, which encompasses information processing and decision making. Within the efforts in management science and informatics invested towards advancing the knowledge on, and providing assistance to decision making, this thesis focuses on the conceptualizations and techniques intended to facilitate the identification, evaluation, and selection of decisions during the earliest stages of information systems engineering, whereby the systems of interest are deployed to partly or fully automate various organizational processes, including information processing ones. The overall motivating problem that drove to, and that unites the various contributions presented in this thesis is how to better inform decision making and guide it towards decisions that will increase the quality (as evaluated both by the engineer and the stakeholders) of the information system being engineered.

Topics in two key related areas are therefore addressed. First, boundedly rational individuals cannot take engineering decisions by accounting for all information that may be, or actually is available to them. As their information processing abilities are limited and their perception biased, it is necessary to filter the available information to a manageable level, and to bring it to a format that facilitates the rigorous reasoning invested in decision making. Second, it is necessary to provide guidance on how to use the given information in decision making.

The first part of this thesis therefore focuses on conceptualizations that facilitate the identification of relevant information and its organization for subsequent analysis, all in the aim of achieving high quality of the system being engineered. In particular, Part I discusses, shows deficiencies, and accordingly revises the conceptual foundations of requirements engineering, a field of information systems engineering that focuses on the identification and analysis of requirements communicated by the stakeholders to the engineer of the system. The novelty of the suggested revision lies primarily in (i) the separation between functional and nonfunctional (i.e., quality) requirements grounded in a foundational ontology, (ii) the introduction of stakeholders' communicated attitudes as important sources of information for the evaluation of alternative requirements engineering decisions, (iii) the reformulation of the so-called "requirements problem" – which precisely defines when the requirements engineering effort is successfully completed – to account for attitudes and nonfunctional requirements, and (iv) the recognition of the importance of defeasible reasoning in the search for a solution to the requirements problem. Acknowledging the importance of defeasible reasoning leads – in Part II – to the study of how defeasible reasoning can be incorporated into established decision making processes involved in the identification and analysis of requirements. Novelty in Part II lies mainly in (i) the use of argumentation and justification processes in the modeling and analysis of requirements, (ii) the combined use of design rationale approaches with argumentation and justification, (iii) the recognition that the clarity of arguments is variable (due to ambiguity, vagueness, synonymy, and overgenerality of information going into premises and conclusions in arguments), (iv) the definition of a number of techniques for the detection of unclear information and its clarification, and (v) the use of "clarity" as a criterion for the discrimination among arguments. Part III shows how the conceptualizations and techniques introduced in Parts I and II are applied within and are relevant to the engineering of information systems, including those that rely on heterogenous and distributed components, as in service-oriented and agent-oriented computing.

Acknowledgements

I thank Stéphane Faulkner, the supervisor of this thesis for our many discussions of the topics studied herein, his continuing support, and relevance and detail of his feedback. Most of all, I thank Stéphane for giving me the freedom to choose the issues to explore.

I thank Pierre-Yves Schobbens, the co-supervisor of this thesis, for constructive feedback on the contributions presented herein. I am particularly grateful to Pierre-Yves for his rigorous critical attitude towards the ideas discussed in the thesis.

I thank other members of the jury, Manuel Kolp, Esteban Zimányi, and John Mylopoulos for their valuable comments and words of encouragement.

I am indebted to John Mylopoulos for our many discussions and the ideas that ensued, of which only a small part is presented in this thesis. Our future publications will showcase those missing here, along with many more. As John recently said: "Now, onward to bigger and better things."

I am grateful to Katia Sycara for inviting me to Carnegie Mellon University. Martin Kollingbaum and Katia Sycara introduced me to the problems that led us to bridge requirements engineering and the writing of contracts in norm-governed multiagent systems.

This work would not have been possible without the funding from Collège Interuniversitaire pour les sciences du Management (CIM). I thank Francoise Charlez and Dirk Symoens at CIM for encouragement and support.

I am infinitely grateful to my parents for their unconditional love and support.

Ivan Jureta

Contents

1

Introduction

Efficient organization requires rigorous and systematic information management, which encompasses information processing and decision making. Within the efforts in management science and informatics invested towards advancing the knowledge on, and providing assistance to decision making, this thesis focuses on the conceptualizations and techniques intended to facilitate the identification, evaluation, and selection of decisions during the earliest stages of information systems engineering, whereby the systems of interest are deployed to partly or fully automate various organizational processes, including information processing ones.

The overall motivating problem that drove to, and that unites the various contributions presented in this thesis is how to better inform decision making and guide it towards decisions that will increase the *quality* (as evaluated both by the engineer and the stakeholders) of the information system being engineered. Below, a broad outline of the background and related work is first provided in order to situate the work presented in this thesis within information management and information systems engineering (§1.1). The discussion abstracts from details because each subsequent chapter – having been peer-reviewed and published separately – provides its local and detailed discussion of the background and related work. The specific issues that the thesis discusses and suggests answers to are then presented (§1.2). Contributions elaborated in response to the raised issues are mentioned and outlined (§1.3), before this introductory chapter closes with an outline of the remainder of the thesis (§1.4).

1.1 Quality within Information Management and Information Systems Engineering

The concept of *quality* has been variously defined as value, conformance to specifications, conformance to requirements, fitness for use, loss avoidance, or achieving and/or exceeding customer expectations (see, e.g., [270] for an overview). Reasoned, structured, and systematic action taken to achieve desired quality—i.e., *quality management*—has been an active area of enquiry in various fields, most notably management science.

Research and empirical evidence accumulated over the last half-century in management science highlights essentially two components to quality management: (i) methods and tools appropriate for the representation and reasoning about quality of a service or product, and (ii) methods for establishing the organizational setting conducive to achieving some desired level of quality. The former (i) usually involve metrics and statistical techniques for metric analysis. The latter (ii) are concerned with means for ensuring, e.g., leadership in and cross-functional support for quality initiatives, participation, responsibility, education, and training of both management and employees. Seminal frameworks that cover both components, include the Total Quality Management approach [115], which originated in works from Juran [146], Deming [73], and Ishikawa [138], and the Six Sigma initiative [118, 119]. Statistical tools are used — in both classes of frameworks — to analyze quality levels and are combined with management principles and methods for bringing about the organizational conditions for achieving and maintaining desired quality levels. For instance, Ishikawa [138] suggests a set of tools (including, e.g., Pareto chart, Scatter diagram, Check sheet) and a set of principles (e.g., cross-functional management) to apply when implementing a quality initiative. There is now no doubt about the benefits of quality initiatives in firms; in addition to specific and widely publicised success stories of, e.g., General Electric and Motorola, large-scale studies indicate that firms that receive quality awards outperform those that do not in terms of operating income and revenues [123] and stock performance [124].

Notable contributions in management science have been carried over to information systems (IS) engineering over the last three decades (e.g., [97]). For instance, the Capability Maturity Model [137] from the Carnegie Mellon University Software Engineering Institute is grounded in contributions from Deming [73] and Juran [146]. Novel proposals appared alongside, motivated by the specifics of IS and of its lifecycle [176]. Overall, the research clusters into three groups: (a) *quality modeling*, aimed at determining what quality is for given (class of) IS; (b) *quality measurement*, employed to quantitatively determine levels of quality in IS and (c) *quality assurance*, concerned with policies, processes, and tools needed to achieve and maintain some desired quality levels.

In quality modeling (a), the primary concern is on how to appropriately represent relevant information on, and reason about the quality of IS. Early on, Boehm and colleagues [30] defined quality of IS in terms of a collection of distinct, interrelated, and interdependent quality attributes, including general utility, decomposed onto as-is utility, maintainability, and portability, which are subsequently decomposed onto, e.g., reliability, efficiency, and so on. Effort on obtaining precise and widely applicable definitions of software quality ensued. Quality models grounded in early research have been standardized (see, e.g., ISO 9126 [93]). Following observations that the choice of quality attributes and of their hierarchy may appear arbitrary, and that there are limited means for measuring characteristics at the hierarchies' top [176], such "top-down" approaches to quality modeling currently remain of interest mainly as catalogs of quality attributes (e.g., [113]). The catalogs are now applied usually in combination with a "bottom-up" approach [78], whereby measurable IS properties are linked to quality attributes these properties affect. In parallel with developments in quality models focused on quantitative evaluation of quality, qualitative models have also been proposed: the Nonfunctional Requirements approach [225, 55] allows both top-down and bottom-up quality modeling, while evaluating quality through a qualitative reasoning procedure.

Quality measurement (b) (e.g., [88]) involves the definition of measures (i.e., metrics) and methods on how measurement data is collected, stored, and used for decision making. The aim is to provide quantitative information for estimation, prediction, assessment, and benchmarking of quality attributes identified in a quality model, and of effort and cost (to be) invested in the various lifecycle phases of IS. Since the early metrics on program code (e.g., [104]), it is now recognized that effective measurement requires the definition of the measurement context, which facilitates the selection of particular metrics and interpretation of measurement data. Notable measurement approaches which implement these guidelines include the Goal Question Metric (GQM) approach [19, 18, 43], Quality Function Deployment [5], and the Software Quality Metrics approach [31]. For instance, the GQM defines the measurement context by identifying goals (e.g., improve the timeliness of...) w.r.t. objects of measurement (e.g., ...the change request processing), then questions that need to be answered by the collected data (e.g., "What is the current change request processing speed?"). Metrics (e.g., Average cycle time) are then identified to answer the questions. Recent advances in measurement involve the use of causal models [89] (namely, Bayesian belief nets [251]) that can integrate diverse variables, empirical evidence and expert judgement, cause and effect relationships, uncertainty, and incomplete information.

Quality assurance (c) focuses on the identification and application of policies, processes, and tools needed to achieve and maintain some desired quality levels. Significant attention is paid on means for eliciting, understanding, and making precise the requirements of various system stakeholders (e.g., [267, 101, 16]). To specify requirements, one describes the stimuli that the future system may encounter in its operating environment and defines the system's responses according to the stakeholders' requirements. The requirements engineering field addresses this issue through various tools and methods intended to facilitate dealing with requirements. Various paradigms have been suggested, each grounded in a set of concepts, constructs, and techniques shown to be relevant in facilitating the engineering of requirements. For instance, in goal-oriented requirements engineering (e.g. [225, 71, 332, 197, 100]), the engineer elicits abstract and imprecise goals that the stakeholders expect the system to achieve once operational. The engineer then proceeds to make these goals precise and determine the properties and behaviors of the system needed to achieve the goals. Total Quality Management concepts, such as statistical quality control have also been applied to quality assurance (e.g., [301]). In Software Quality Function Deployment (see, [302, 114]), stakeholder requirements are elicited and recorded, then converted into technical and measurable statements about the IS, requirements are prioritized, and a technical specification is produced. Effort has been invested in reseach on sharing knowledge about the IS development processes (e.g., [137]), on increasing stakeholder involvement during development (e.g., [127, 29]), and on managing change that results from the introduction of a new automated system (e.g., [240, 212]).

1.2 Unifying Motive and Two Principal Issues

1.2.1 Two Perspectives on the Quality of an Information System

The principal aim of representing and reasoning about quality is to facilitate the construction and running of IS capable of meeting and exceeding stakeholders' requirements. The quantitative, metric-based approach to quality enables a characterization of the system in terms of its measurable properties and behaviors. Within such a perspective, the engineer of the system can argue that the IS achieves some levels of quality according to the engineer's quality conceptualization or one adopted from international standards. *This, however, gives no guarantee that the system's stakeholders will share the same perception and evaluation of the quality of the IS.* Indeed, it is acknowledged in management (e.g., [270, 236, 247, 248]) and software engineering research (e.g., [176]) that quality is a subjective experience: evaluations of quality will vary among users and will not necessarily correspond to evaluations based on metrics over measurable properties and behaviors of the system. [1]

In management science, marketing research that focuses on service quality highlights the importance of expectations for customers' evaluation of quality. Seminal contributions (e.g., [247]) suggest that a customer's subjective evaluation of quality is related to the gap between what she expects and what she perceives at service delivery (e.g., [111, 247]). Following this line of thinking, a recent definition suggests that service quality is "the degree and direction of discrepancy between customers' service perceptions and expectations" [248]. To better understand how consumers evaluate service quality, marketing scholars have adopted an approach similar to that taken in software quality modeling. Namely, sets of factors determining perceived service quality have been suggested, not unlike quality attributes in software quality conceptualizations.[2] The common approach to measure perceived service quality is to make quality determinants measurable through questionnaires and seek trends in data based on postulated models.

The above discussion leads to the observation that *to engineer high quality IS, it is necessary to take into account both the information referring to the engineer's perspective on quality and the information referring to the stakeholders' perspective.*

1.2.2 Integrating the Two Perspectives in Decision Making during Information Systems Engineering

While it may appear intuitive and somewhat apparent that high quality IS cannot be engineered without combining the system engineer's understanding and evaluation of quality and those of the stakeholders, this is not acknowledged in conceptualizations and techniques used over the course of the requirements engineering (RE) effort involved in IS engineering. RE – a field of information systems engineering that focuses on the identification and analysis of requirements communicated by the stakeholders to the engineer of the system – is critical for improving the quality of the system being developed: it is during RE that stakeholders communicate their quality requirements to the engineer, who in turn is expected to account for these in subsequent decision making during IS engineeering. To successfully account for this information during decision making requires that the following two issues be addressed.

- *Problem of conceptualization.* Stakeholders will communicate their quality expectations to the engineer so that the latter can take these into account during decision making. By doing so, stakeholders will be conveying beliefs, desires, intentions, and attitudes about what the IS will need to do and how well it will need to do it in order for stakeholders to evaluate the IS as being of acceptable or high quality. Given this communication, the engineer needs a conceptual framework to properly distinguish among, and organize the communicated beliefs, desires, intentions, and attitudes, as each is relevant in its own particular way for subsequent analysis and decision making.

[1] That quality is a subjective experience is clear in industry. For instance, one of the criteria for U.S. Department of Commerce's award for outstanding quality states:

> "Quality is judged by the customer. All product and service attributes that contribute value to the customer and lead to customer satisfaction and preference must be the foundation of a company's value system. Value, satisfaction, and preference may be influenced by many factors throughout the customer's overall purchase, ownership, and service experiences." [234]

[2] The prominent set of determinants [247] includes: knowledge and curtesy of employees delivering the service and their ability to inspire trust and confidence (termed: assurance), caring and individualized attention providede to the customer (empathy), ability to perform the promised service dependably and accurately (reliability), willingness to help customers and provide prompt service (responsiveness), and appearance of physical facilities, equipment, personnel, and communications material (tangibles).

- *Problem of information use.* Decision making is facilitated if it is based on information that lends itself to quantitative analysis and comparison using, e.g., some multicriteria decision aid. Information communicated to the engineer are rarely precise enough to allow immediate or easy quantification. Once the engineer needs to account for the variety of beliefs, desires, intentions, and attitudes of the stakeholders, the engineer is confronted with a veriety of information that is difficult to account for in decision making. Relevant information may take qualitative or quantitative form, and be expressed in formal or informal notation. Systematic processes that are robust with regards to the form of, and precision of the information, and that can accommodate the full scope of the said conceptual framework are necessary to assist decision making.

The two issues above are recurrent in RE, having been the subject of various discussions and many responses thereto have been suggested. We review and discuss these in relation to the precise topics treated in each chapter in the remainder. However, as various chapters of this thesis show, the available responses are deficient with regards to the treatment of quality-related information. In particular, the thesis asks the following questions:

1. Given that stakeholders communicate desires, beliefs, intentions, and attitudes about the future IS, are there conceptualizations in RE that allow the representation and analysis of all such information?
2. How do stakeholders' desires, beliefs, intentions, and attitudes relate to the concept of quality?
3. What are the criteria for judging that the RE effort is successfully completed, whereby successful completion subsumes that the chosen design of the IS will arguably lead to high perceived quality by the stakeholders?
4. Given that the information communicated by the stakeholders usually takes different form and itself is of different quality, what techniques can be applied for decision making during RE and that are robust with regards to such characteristics of information?
5. What techniques can be applied to analyze the mostly informal, and potentially vague, ambiguous, and in general unclear information communicated by the stakeholders during RE, and that conveys their beliefs, desires, intentions, and attitudes?

The thesis provides a response to these questions. Validation of the responses proves particularly difficult for the contributions are conceptual and remain often at a too high level of abstraction to lend themselves to direct use and experimentation. This is in general a difficulty of the field, as industrial application often comes years after the contributions are presented to the relevant communities. Appropriateness of the contributions is, however, argued on the following grounds. First, rigorous arguments are provided when identifying the deficiencies of related work and strong arguments and rigorous argumentation – as illustrated throughout the thesis – have gone into the elaboration of the suggested responses to the raised issues. Second, the various contributions are accepted by the respective fields: the chapters of the thesis have been peer-reviewed and published. Finally, the usual means of validation in the concerned fields – that is, case studies – are used throughout the thesis to provide concrete examples of how the contributions depart from related work and how they advance the available research results in the relevant fields of enquiry.

1.3 Contributions

This thesis makes specific contributions towards the resolution of the given problems of conceptualization and information use involved in decision making in IS engineering. Contributions respond to the issues 1–5 raised above.

1.3.1 Revisiting the Role of Quality within the Conceptual Foundations of Information Systems Engineering

The first part of this thesis focuses on conceptualizations that facilitate the identification of relevant information and its organization for subsequent analysis, all in the aim of achieving high quality of the system being engineered. In particular, Part I discusses, shows deficiencies, and accordingly revises the conceptual foundations of RE, thus addressing the issues 1–3 raised earlier.

In IS engineering, the principal concept used for the representation of, and as a foundation for reasoning about stakeholders' expectation is the concept of *requirement*. Review of the literature shows that the requirement concept has been variously defined as a purpose, a need, a goal, a function(ality), a constraint,

a behavior, a service, a condition, or a capability. Limited effort has been put into making explicit the assumptions and choices behind the various available definitions. In contrast to the available definitions, we propose one given in a restricted vocabulary of a foundational ontology. Assumptions and philosophical choices are discussed along with implications. The requirement and associated concepts are used in the first step of IS engineering, namely RE. Much of the information that the IS engineer needs for decision making during RE is communicated by the stakeholders. We show that the current basic terminology for RE (one that underlies established software and RE frameworks) does not allow the representation and reasoning about all concerns – namely, all of beliefs, desires, intentions, and attitudes – that stakeholders communicate. In response, we provide an ontology that covers all of these concerns and rests on sound conceptual foundations defined by an explicit foundational ontology, which captures ontological categories underlying natural language and human common sense. The new ontology leads to a new formulation of the so-called "requirements problem", that is, the fundamental problem of the RE and IS engineering fields. The new formulation avoids the pitfalls of the previously accepted formulation established in Zave and Jackson's seminal work [339].

The novelty of the suggested revision lies primarily in:

- the separation between functional and nonfunctional (i.e., quality) requirements grounded in a foundational ontology, which gives more precise definitions;
- the introduction of stakeholders' communicated attitudes as important sources of information for the evaluation of alternative RE decisions;
- the reformulation of the requirements problem – which precisely defines when the requirements engineering effort is successfully completed – to account for attitudes and nonfunctional requirements;
- the recognition of the importance of defeasible reasoning in the search for a solution to the requirements problem.

Overall, the ontology extends the current conceptual foundations in RE by integrating notions relevant for the representation of various information communicated by the stakeholders, and in particular – through care taken in dealing with attitudes – for information that concerns quality expectations.

1.3.2 Argumentation, Justification, and Clarification in Decision Making for Requirements Engineering

Acknowledging the importance of defeasible reasoning leads – in Part II – to the study of how defeasible reasoning can be incorporated into established decision making processes involved in the identification and analysis of requirements. While it is of apparent interest to have a rich ontology to guide the identification and organization of information relevant for decision making during RE, the ontology itself does not help in dealing with information of various levels of precision, formality, and that takes qualitative or quantitative form. Meeting quality expectations involves the use of the available information when defining the purpose of the IS. Representation and reasoning about IS purpose thus unavoidably involve the transformation of unclear stakeholder requirements into an instance of a model of IS goals. Defeasible reasoning, and in particular the use of argumentation makes it possible to accommodate such information and its transformation in order to resolve the requirements problem.

It is suggested in Part II of the thesis to use defeasible reasoning in the construction and revision of models that represent the system and its relevant surroudings, and that are used to support decision making during RE. In particular, it is suggested to (i) externalize and document arguments that led the stakeholders to express some requirements as well as the arguments that led the engineer to transform these requirements into model content; and, (ii) to analyze the clarity of these arguments and of the information given within the model, revising it if proves necessary. To facilitate tasks (i) and (ii), the thesis proposes the so-called "Goal Argumentation Method (GAM)", which integrates a decision process, an argumentation model, and techniques combined to enable the analysis of argument structure, justification of modeling choices, and clarification of information appearing in arguments and elsewhere in the the goal modeling decision process. Drawing on design rationale literature, the decision process suggests an intuitively acceptable organization of the goal modeling task. The argumentation model, inspired by work in artificial intelligence argument models, is introduced in the evaluative step of the decision process, allowing various degrees of structure and rigor in the provision of arguments for modeling choices. The analysis techniques serve for the justification of modeling choices, the study of argument interaction (e.g., defeat and counterargumentation), the detection of deficient argumentation, and the checking for unclear information and subsequent clarification. An important characteristic of GAM is that it does not integrate a particular IS model; instead, it is independent of specific syntax and semantics of the IS model, allowing

its combined use with any available RE framework which models the purpose of the IS using the notion of "goal". In conjunction with any available goal-oriented RE framework, GAM fulfils three roles:

1. GAM guides argumentation and justification of modeling choices during the construction or revision/critique of IS goal diagrams.
2. It enables the detection of deficient argumentation within available goal diagrams (which need not have been built using GAM).
3. It provides practical techniques for the requirements engineer to ensure (to a reasonable extent) that information appearing both in arguments and in diagram elements is clear (i.e., is not ambiguous, overgeneral, vague, among others).

Novelty in Part II lies mainly in:

- the use of argumentation and justification processes in the modeling and analysis of requirements;
- the combined use of design rationale approaches with argumentation and justification;
- the recognition that the clarity of arguments is variable (due to ambiguity, vagueness, synonymy, and overgenerality of information going into premises and conclusions in arguments);
- the definition of a number of techniques for the detection of unclear information and its clarification, and (v) the use of "clarity" as a criterion for the discrimination among arguments.

The principal contributions of GAM to the RE field are the introduction of generic argumentation and clarification conceptualizations and techniques in goal modeling for RE. The thesis highlights that the activities of argumentation and clarification are method-independent concerns and therefore significant for decision-making in RE at large. These contributions respond to the issues 4 and 5 raised above (§1.2.2).

1.4 Outline of the Thesis

The chapters in the remainder of this thesis have been published as peer-reviewed publications or have directly served in the elaboration of other publications. Most of these publications are listed in the following chapter. In this respect, this thesis provides neither a central treatment of related work nor a discussion of limitations. Each of these considerations is locally dealt with in the relevant chapters, and for the given topics.

Part I of the thesis introduces the conceptual framework that responds to the conceptualization problem outlined above. Chapter 4 introduces the conceptual framework, while Chapters 5 and 6 focus on a particular and highly debated concept used for the representation of quality requirements during RE decision making.

Part II introduces and shows the use of the techniques for the clarification of information that needs to be used in decision making. The argumentation and justification processes are explained and shown to enable systematic decision making in presence of information that may be more or less clear, more or less formal, and/or qualitative or quantitative.

Spread over four chapters, Part III of the thesis outlines the use of the theoretical contributions from Parts I and II to address specific issues in IS engineering. Each of these chapters adds specific contributions to the respective communities in which they are presented. In Chapter 10, we show that principles underlying GAM (introduced in Part II) apply to the design of classical enterprise IS, and in particular enable a new approach to traceability (i.e., the ability to describe and follow the life of a requirement). In Chapter 11, we adapt GAM and its underlying principles to the engineering of requirements for Adaptable and Open Service-oriented Systems (AOSS). Such systems pose novel problems in decision-making during RE for it is not feasible to specify requirements for AOSS by applying the commonly accepted generic approach in RE, which consists of moving from the elicitation, representation, and analysis of abstract stakeholder expectations towards a detailed specification of the entire system's behavior before deploying the system. Chapter 12 focuses on the problem of selecting among competing services those that are capable of satisfying functional and nonfunctional (i.e., quality) requirements. We combine a simple architecture with a novel algorithm to enable openness, distribution, and multi-criteria-driven service composition at runtime. The concern in Chapter 13 is the dynamic (i.e., runtime) engineering of norm-governed multi-agent systems (MAS), by automatically deriving executable norm specifications from rich models of requirements, built upon the conceptual foundations introduced in Part I of the thesis.

Part IV closes the thesis. Because the contributions in Parts I and II rely on a number of choices, these choices – for instance, the choice of a formal model of argumentation – are revisited in light of available alternatives. Indications are then given on how some of the contributions can be submitted to

further validation. Finally, a summary is given and principal directions for future work are identified and discussed.

2

Publications

The following peer-reviewed publications are directly based on and are derived from the material presented in this thesis. They are listed in reverse chronological order below:

1. **Ivan J. Jureta**, Stéphane Faulkner, Pierre-Yves Schobbens. Clear Justification of Modelling Decision for Goal-Oriented Requirements Engineering. Accepted for publication in: *Requirements Engineering Journal*, Springer-Verlag.
2. **Ivan J. Jureta**, Caroline Herssens, Stéphane Faulkner. A Comprehensive Quality Model for Service-Oriented Systems. Accepted for publication in: *Software Quality Journal*, Springer-Verlag.
3. **Ivan J. Jureta**, Stéphane Faulkner, Manuel Kolp. An Agent-Oriented Enterprise Model for Early Requirements Engineering. In Peter Rittgen (Ed.), *Handbook of Ontologies for Business Interaction*, IDEA Group Publishing, 2007.
4. John Mylopoulos, **Ivan J. Jureta**, Stéphane Faulkner. An Ontology of Requirements. *RIGiM'07 Keynote by John Mylopoulos. Conceptual Modeling - ER 2007, 26th International Conference on Conceptual Modeling, Auckland, New Zealand, November 2007, Lecture Notes in Computer Science, Springer.*
5. **Ivan J. Jureta**, Stéphane Faulkner, Pierre-Yves Schobbens. Achieving, Satisficing, Excelling. *Proceedings of the 1st International Workshop on Requirements, Intentions and Goals in Conceptual Modeling (RIGiM07)*, Auckland, New Zealand, November 2007, Lecture Notes in Computer Science, Springer.
6. **Ivan J. Jureta**, Stéphane Faulkner. Tracing the Rationale behind UML Model Change through Argumentation. *Proceedings of the 26th International Conference on Conceptual Modeling (ER'07)*, Auckland, New Zealand, November 2007, Lecture Notes in Computer Science, Springer.
7. **Ivan J. Jureta**, Stéphane Faulkner. Clarifying Goal Models. *Proceedings of the 26th International Conference on Conceptual Modeling (ER'07)*, Auckland, New Zealand, November 2007, Conferences in Research and Practice in Information Technology, Australian Computer Society.
8. **Ivan J. Jureta**, Stéphane Faulkner, Philippe Thiran. Dynamic Requirements Specification for Adaptable and Open Service Systems. *Proceedings of the 15th IEEE International Conference on Requirements Engineering (RE'07)*, New Delhi, India, October 2007, IEEE Computer Society Press.
9. **Ivan J. Jureta**, Stéphane Faulkner, Philippe Thiran. Dynamic Requirements Specification for Adaptable and Open Service-Oriented Systems. *Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC'07)*, Vienna, Austria, September 2007, Lecture Notes in Computer Science, Springer.
10. **Ivan J. Jureta**, Stéphane Faulkner, Youssef Achbany, Marco Saerens. Dynamic Web Service Composition within a Service-Oriented Architecture. *Proceedings of the 7th International Conference on Web Services (ICWS'07)*, Salt Lake City, Utah, July 2007, IEEE Computer Society Press.
11. **Ivan J. Jureta**, Stéphane Faulkner, Youssef Achbany, Marco Saerens. Dynamic Task Allocation Wihin an Open Service-Oriented MAS Architecture. *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multi-Agents Systems (AAMAS'07)*, Honolulu, Hawai'i, May 2007, ACM Press.
12. **Ivan J. Jureta**, Stéphane Faulkner, Pierre-Yves Schobbens. A More Expressive Softgoal Conceptualization for Quality Requirements Analysis. *Proceedings of the of the 25th International Conference on Conceptual Modelling (ER'06)*, Tucson, Arizona, November 2006, Lecture Notes in Computer Science, Vol. 4215, Springer, 281–295.

13. **Ivan J. Jureta**, Stéphane Faulkner, Pierre-Yves Schobbens. Justifying Goal Models. *Proceedings of the 14th IEEE International Conference on Requirements Engineering (RE'06)*, Minneapolis, Minnesota, September 2006, IEEE Computer Society Press, 119–129.
14. **Ivan J. Jureta**, Stéphane Faulkner, Manuel Kolp. Formalizing Agent-Oriented Enterprise Models. In *Agent-Oriented Information Systems III* (Revised Selected Papers), Lecture Notes in Computer Science, Vol. 3529, Springer, 184–199.
15. **Ivan J. Jureta**, Stéphane Faulkner, Pierre-Yves Schobbens. Allocating Goals to Agent Roles during MAS Requirements Engineering. *Proceedings of the 7th International Workshops on Agent-Oriented Software Engineering (AOSE'06), 5th International Joint Conference on Autonomous Agents and Multi-Agents Systems (AAMAS'06)*, Hakodate, Japan, May 2006, Lecture Notes in Computer Science, Springer.
16. **Ivan J. Jureta**, Stéphane Faulkner, Manuel Kolp. Multi-Agent Patterns for Deploying Online Auctions. *International Journal of Intelligent Information Technologies*, 2(3):21–39, 2006.
17. **Ivan J. Jureta**, Stéphane Faulkner. An Agent-Oriented Meta-model for Enterprise Modelling. *Proceedings of the 7th International Bi-Conference Workshops on Agent-Oriented Information Systems (AOIS'05), 24th International Conference on Conceptual Modelling (ER'05)*, Klagenfurt, Austria, November 2005, Lecture Notes in Computer Science, Vol. 3770, Springer, 151–161.
18. **Ivan J. Jureta**, Manuel Kolp, Stphane Faulkner, T. Tung Do. Patterns for Agent Oriented e-Bidding Practices. *Proceedings of the 9th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES'05)*, Melbourne, Australia, September 2005, Lecture Notes in Computer Science, Vol. 3682, Springer, 814–820.

Part I

Conceptual Foundations

3

Outline of Part I

The first part of this thesis focuses on conceptualizations that facilitate the identification of relevant information and its organization for subsequent analysis, all in the aim of achieving high quality of the system being engineered. In particular, Part I discusses, shows deficiencies, and accordingly revises the conceptual foundations of requirements engineering, a field of information systems engineering that focuses on the identification and analysis of requirements communicated by the stakeholders to the engineer of the system.

Chapter 4 revisits the core ontology for RE established by Zave and Jackson [339]. It is shown that their proposal can be improved in several directions in order to arrive at a core ontology that spans much of current RE research, while standing on more solid conceptual foundations. More light is shed on the intended interpretation of concepts such as "goal", "softgoal", and so on, commonly used in RE frameworks. Definitions grounded in a foundational ontology are given for functional and nonfunctional requirements, thus allowing their precise distinction. The new core ontology starts from the reasonable premise that the information relevant for RE is communicated by the stakeholders through speech acts, so that the core ontology should incorporate concepts needed to model the content of the beliefs, desires, intentions, and attitudes that the stakeholders communicate. The new ontology leads to the revision of the requirements problem, which precisely defines what it means for RE to be successfully completed. The new formulation of the requirements problem highlights the role of quality in the engineering of requirements during IS engineering, and calls for new concepts and techniques for the resolution of the requirements problem.

Chapter 5 discusses the different kinds of goals that are used in the modeling of IS requirements, giving formal definitions where appropriate. Given the new formulation of the requirements problem, Chapter 4 argues that satisficing no longer appropriately describes the effort invested in RE, but that a different notion, that of excelling is more appropriate. Broadly speaking, excelling amounts to continual revision of satisficing thresholds: the thresholds established for satisficing are improved over time in order to increase the quality of the IS. It is argued that this reflects current industrial practice: IS are improved through revisions and upgrades, so that quality does improve (at least, this is what is desired) over time, and this by continually aligning the IS to revised quality requirements.

Chapter 6 discusses th slippery softgoal concept, necessary for dealing with abstract nonfunctional requirements, such as security, safety, convenience, and so on. It is argued that the analysis of softgoals should account for the multiple facets of the concept, and guidelines are given for the construction of methodologies that aim to properly address softgoals and account for them during decision making.

4

Revisiting the Core Ontology and Problem in Requirements Engineering

Abstract. In their seminal paper in the ACM Transactions on Software Engineering and Methodology, Zave and Jackson established a core ontology for Requirements Engineering (RE) and used it to formulate the "requirements problem", thereby defining what it means to successfully complete RE. Starting from the premise that the stakeholders of the system-to-be communicate to the software engineer the information needed to perform RE, we show that Zave and Jackson's ontology is incomplete. It does not cover all types of basic concerns – namely, the beliefs, desires, intentions, and attitudes – that the stakeholders communicate. In response, we provide a core ontology that covers all types of possible basic concerns and rests on sound conceptual foundations defined by an explicit foundational ontology. The new core ontology for RE leads to the new formulation of the requirements problem that avoids the pitfalls of Zave and Jackson's formulation. We thereby establish new standards for what minimum information should be represented in RE languages and new criteria for determining whether RE has been successfully completed.

4.1 Introduction

A decade ago Zave and Jackson [339] observed that the field of Requirements Engineering (RE) left behind simplistic approaches to understanding what a system will do in favor of novel and varied terminology, methods, languages, tools, and issues considered to be critical. They went on to define a core ontology that establishes minimum standards for what information should be represented in any RE language. This allowed them to formulate the "requirements problem", which determines extactly what it means for RE to be successfully completed. Various representations and interpretations of the core ontology have since been suggested and used. Despite continued progress, there is limited consensus on the precise meaning of the core ontology in RE. Rather, a requirement is variously understood as (describing) a purpose, a need, a goal, a functionality, a constraint, a quality, a behavior, a service, a condition, or a capability. Terms such as "nonfunctional requirement", "softgoal", "preference", "priority" only add to the confusion. It is difficult to compare or combine contributions, and identify conceptual overlaps.

The only relevant core ontology is one which helps the software engineer in solving the requirements problem. Zave and Jackson's [339] provided an elegant characterization of the requirements problem by relying on three concepts that constitute their core ontology. A "requirement" is an optative (i.e., desired) property of the environment, which includes the system-to-be and its relevant surroundings. A "domain assumption" is an indicative property, describing the environment as it is and in spite of the system-to-be. An optative property, intended to be directly implementable and support the satisfaction of requirements, is a fragment of a "specification". From there on, Zave and Jackson suggest that *the requirements problem amounts to finding the specification* $\boldsymbol{S}$ *that for given domain assumptions* $\boldsymbol{D}$ *satisfies the given requirements* $\boldsymbol{R}$. If all three are written in a mathematical logic, the problem is solved once the engineer finds $\mathbf{S}$ such that $\mathbf{D}, \mathbf{S} \vdash \mathbf{R}$. This characterization is, however, deficient for the following reasons:

1. Satisfaction in $\mathbf{D}, \mathbf{S} \vdash \mathbf{R}$ is binary: $\mathbf{D}$ and $\mathbf{S}$ cannot satisfy $\mathbf{R}$ to some extent only. This is an oversimplification, for it is impossible to consider the software resulting from $\mathbf{S}$ as being of higher or lower quality - its quality is instead either acceptable ($\mathbf{D}, \mathbf{S} \vdash \mathbf{R}$) or unacceptable ($\mathbf{D}, \mathbf{S} \nvdash \mathbf{R}$). In stark contrast to reality, it is impossible for stakeholders (i.e., users, owners, etc.) to be more or less satisfied with the system.
2. The given formulation leads us to conclude that any two distinct specifications $\mathbf{S}_1$ and $\mathbf{S}_2$, such that $\mathbf{D}, \mathbf{S}_1 \vdash \mathbf{R}$ and $\mathbf{D}, \mathbf{S}_2 \vdash \mathbf{R}$, are equally desirable. This is again an oversimplification: e.g., say that

the stakeholders prefer lower response times to a query and certainly expect the response time to be below $2sec$, then if $\mathbf{S}_1$ and $\mathbf{S}_2$ result in, respectively, systems that respond to the query in less than $1sec$ and less than $0.5sec$ all else being equal, we will evidently choose $\mathbf{S}_2$ over $\mathbf{S}_1$.

3. The given characterization is overly generic to be of use, e.g., in guiding the construction of RE languages. While high abstraction is certainly desired, we see from points 1 and 2 above that Zave and Jackson's formulation stays at too high a level. We cannot say – given the above definition of the requirements problem – if the set of concepts in one language is more appropriate to resolve the requirements problem than that of another language.
4. The formulation of the requirements problem depends on the concepts chosen for the core ontology, and *vice versa*: the understanding of the requirements problem influences the choice of concepts for the core ontology. To resolve such circula cause and consequence, Zave and Jackson draw on common sense and experience. While this cannot be avoided altogether, moving towards *stable* conceptual foundations and problem formulation – for which interpretations converge – requires criteria for acceptance that are less dependent on the reader's opinion and the authors' authority.

Our aim is to suggest a core ontology that spans much of ongoing research in RE and is grounded in the communication between the stakeholders and the engineer (§4.2–4.3). This allows us to provide a new formulation of the requirements problem that responds to the above limitations (§4.4). We then draw conclusions, relate to comparable discussions, and highlight important directions for future work (§4.5).

4.2 Baseline

In selecting and defining the concepts in our core ontology, we start from the simple premise that the RE process is one in which *stakeholders communicate information to the software engineer*, whereby stakeholders include people, but also, e.g., legacy systems for which the documentation is available. The engineer's task is to classify information as requirements or otherwise (e.g., domain assumptions), then use it in writing a specification. While this appears to be an obvious understanding of the overall RE process, it has important consequences on the core ontology, and is the key source of novelty with regards to available research. *Utterances that stakeholders make in communicating with the engineer are actions intended to advance stakeholders' personal desires, intentions, beliefs, and attitudes*, in the aim of ensuring that the engineer can produce a specification that then leads to the system responsive to the communicated concerns. It is important to observe here that, *if we know the various kinds of communicative actions at disposal of the stakeholders, we can delimit the scope of the core ontology to concepts that allow us to represent all of the communicated concerns.* While this may appear a natural response to an issue outlined earlier (point 4 in §4.1), it contrasts with available research: therein, core concepts are derived from notions that have been found useful at lower levels of abstraction and/or later in system development stages. E.g., the common "goal" concept (which we discuss later on) comes from research in AI. By looking at the very origin of all requirements and related information, that is, the communication process, our concepts are instead grounded therein: our concepts are defined to cover the communicative actions available to the stakeholders.

We turn to speech-act theory to better understand the communication process. Therein, communication is considered as action [283]: a speaker makes an utterance in an attempt to change the state of the world. What distinguishes speech acts from non-speech actions is the domain of the speech act – i.e., the part of the world that the speaker wishes to modify through the act – is mostly the mental state of the hearer. In our setting, we have stakeholders as speakers, the engineer as the hearer. Given an utterance, *the engineer should distinguish its content from its illocutionary force in order to know to represent the content as a requirement or otherwise.* For example, when I say "Book the plane ticket" (or if I wrote in some documentation given to the engineer), I may be expressing a desire (illocutionary force) that a plane ticket be booked (content, i.e., what I desire). Being desired, the content indicates a requirement that I hope the system will ultimately satisfy.

Depending on the illocutionary force, the engineer will classify the content of the communication as requirements or otherwise, and thereby differently represent and act upon the communicated content: the core ontology should include concepts that cover all kinds of illocutionary force. Depending on illocutionary force, Searle distinguishes the following kinds of speech acts [284]: (i) *assertives*, which assert the proposition that the speaker believes is true; (ii) *directives*, which convey the proposition that the speaker wants to see become true; (iii) *commissives* stating what the speaker intends to (do to) make a proposition true; (iv) *expressives* that convey a speaker's emotion/attitude about herself or the hearer; (v) *declarations* which by the very act of being stated make a proposition true; and (vi) *representative*

declaratives, which recognize the truth of a proposition that has been made true by a declaration. Assertives and representative declaratives communicate beliefs, directives desires, commissives intentions, expressives attitudes, declarations become beliefs when communicated.

Once we know what to cover with the core ontology, we need to produce definitions of the concepts. One commonly defines a concept by mapping it to others, hopefully more familiar and more precise in the reader's vocabulary. It is then hoped that the more these other concepts fit the reader's intuition, the more relevant in her view the definitions. To ensure an acceptable degree of rigor, we define our concepts by mapping them to concepts defined in a restricted vocabulary, namely a *foundational* ontology for which the underlying philosophical choices are clear and ontological categories well-delimited and openly discussed. This makes our assumptions clearer and easier to discuss in future efforts.

A foundational ontology is a theory about the abstract domain-independent categories in the real world. Its main purposes are to act as a starting point for building new ontologies, as a reference point for easy and rigorous comparisons among ontological approaches, and as a foundational framework for analyzing, harmonizing, and integrating existing ontologies. Among the various available foundational ontologies [293, 281, 269, 252] (for comparisons, see [214, 56]), we choose DOLCE [214]. DOLCE rests on intuitively attractive ontological choices for the present discussion. In particular, DOLCE is descriptive in that it aims to capture the ontological categories underlying natural language and human common sense. Categories in the ontology are therefore conceived as cognitive artifacts ultimately depending on human perception, cultural imprints, and social conventions. DOLCE is an ontology of particulars that distinguishes four basic categories – any particular is either an *endurant*, a *perdurant*, a *quality*, or an *abstract*. All proper parts of an endurant are wholly present at any time, whereas a perdurant accumulates parts over time (only some of its parts are present at any time). A quality is a basic perceivable and measurable entity that inheres to other entities (e.g., the color of this desk). A quality maps to a "value" (the perceived color of the desk), called a quale, which describes the position of the given particular entity within a particular conceptual space (here, quality space). A quality space has some structure that reflects our cognitive and perceptual bias (e.g., length can be given in a metric linear space). Finally, an abstract entity has no spatial or temporal qualities, and is not a quality itself (e.g., fact, set, time interval, etc.). In terms of basic ontological categories in DOLCE, communication is a perdurant, and speech acts are sub-categories of communication. Mental states expressed by speech acts are endurants. More precisely, they are sub-categories of DOLCE's notion of mental object. In the remainder, we use DOLCE with explicit speech acts and mental states as the foundational ontology that gives us a restricted vocabulary for defining the core ontology for RE.

4.3 Core Ontology for RE

This section introduces the core ontology and explains the rationale behind it. Facing a speech act, the engineer should first distinguish its type (i.e., assertive, commissive, etc.), then separate its modus (modality) from its dictum (content). We have four modalities – belief (B), desire (D), intention (I), and attitude (A) – that correspond to the mental states underlying the speech acts. The engineer associates a modality to the content of a given speech act, and then proceeds to determine if the obtained result is an instance of *domain assumption*, *goal*, *softgoal*, *quality constraint*, *plan*, or *preference* concepts, which together constitute our core ontology for RE. Below, we first explain the association of modalities to communicated content on the basis of speech act type (§4.3.1). We subsequently discuss each concept of the core ontology. We indicate throughout the section precisely how our ontology departs from Zave and Jackson's.

4.3.1 Classifying Communicated Content

Consider the problem of designing a system for booking flights online, for which a stakeholder suggests:

(Ex.1) *No business seat has a lower price than an economy seat.*

If the stakeholder believes the above is already true, the statement is an assertive speech act. If the stakeholder intends to institute the above as a rule, the statement is declarative speech act. If the stakeholder merely reiterates that the above applies, it is representative declarative.

(Ex.2) *I hope/wish/desire/expect booking to be always confirmed with the new system.*

(Ex.3) *I will ensure that booking is always confirmed.*

(Ex.4) *It is preferred that the booking confirmation be sent quickly after booking.*

Statement Ex.2 is directive as it indicates what is desired. Ex.3 is commissive for it points out the stakeholders' intention on bringing about something desired. Finally, Ex.4 is expressive, as it makes explicit stakeholder's attitude on how the booking confirmation is to be sent to the user. While the illocutionary point is not apparent from written text (as in the examples above), it is not absent [285]: when given documentation, the engineer must determine if it expresses desires, facts, or otherwise.

Following Zave and Jackson [339], we know that the critical distinction between a requirement and a domain assumption lies in that the former expresses something desired, while the latter something that is already the case, or is/will be the case regardless of the system. It is not difficult to see that Ex.1 is thereby a domain assumption, while Ex.2 is a requirement. We also know from Zave and Jackson that a specification together with domain assumptions satisfies requirements. Moreover, we know from Cohen and Levesque [62] that an agent adopts intention X if (i) it believes X is possible, (ii) it does not believe it will not bring about X, (iii) it believes it will bring about X, (iv) it does not intend all the side effects of bringing about X, and (v) it invests effort in trying to bring about X. *Specification thereby involves intentions*: agents (including, e.g., stakeholders and the system-to-be) choose how to act and commit to do so, and this in order to bring about states of the world in which requirements are satisfied and domain assumptions are not violated. Hence, while desires lead to requirements and beliefs to domain assumptions, intentions lead to specifications. Ex.3 is part of a specification.

We see therefore that Zave and Jackson's domain assumption, requirement, and specification concepts cover propositional attitudes (i.e., belief, desire, and intention) as they are usually conceptualized in philosophy [239]. More can be, and often is communicated than propositional attitudes. Looking at the notion of attitude in psychology [28] (which is what expressive speech acts convey), attitude is identified most closely with affect, i.e., a general evaluative reaction (e.g., "I like Y" or "I like Y more than Z"). Ex.4 is an expressive as it communicates an attitude. Zave and Jackson's core ontology is incomplete in that attitudes are missing, which, as we show further on, has significant consequences on the formulation of the requirements problem.

Our approach consists first of classifying communicated content by associating modalities to it, whereby the choice of modality depends on the kind of speech act used in communication. We have the following rules for associating modalities to content:

- The content ϕ of an assertive or declarative or representative declarative speech act that stakeholders communicated to the software engineer is a belief, $\mathsf{B}\phi$.
- The content ϕ of a directive speech act that stakeholders communicated to the software engineer is a desire, $\mathsf{D}\phi$.
- The content ϕ of a commissive speech act that stakeholders communicated to the software engineer is an intention, $\mathsf{I}\phi$.
- The content ϕ of an expressive speech act that stakeholders communicated to the software engineer is an attitude, $\mathsf{A}\phi$.

If we encounter conjunctive, disjunctive, or conditional (e.g., if [atomic speech act 1] then [atomic speech act 2]) linking of speech acts, atomic speech acts are separately considered when associating modalities to their content. Observe the parallels between the above and Zave and Jackson's ontology: their requirement here corresponds to $\mathsf{D}\phi$, while $\mathsf{I}\phi$ and $\mathsf{B}\phi$ correspond to, respectively, a specification (that is, a fragment thereof) and a domain assumption. There is no departure in this respect from the accepted intuitions in RE: requirements cover what is desired, while domain assumptions concern what is true. Intentions give specification fragments for they determine what will be done to satisfy requirements.

4.3.2 Domain Assumption

Beliefs communicated by way of assertive, declarative, or representative declarative speech acts constrain the possible states of the world only to those in which beliefs are not violated. They correspond to domain assumptions, which should not be violated by the sytem or its relevant surroundings, as the following definition indicates.

Definition 4.1. *Believed content ϕ – i.e., ϕ in $\mathsf{B}\phi$ – communicated by way of assertive, declarative, or representative declarative speech acts is a* ***domain assumption****, denoted generically* ***d****.*

Believed content is thus considered a member of the set **D** of domain assumptions that the software engineer elicits and should account for when building a specification. **D** therefore contains the content of the kinds of speech acts indicated in Definition 4.1.

(Ex.5) *Standard credit card symbols must be displayed whenever the customer is asked to make a payment.*

If the statement above is the content of an assertive speech act, the engineer sees it as a domain assumption. In other words, the content above delimits possible states of the world only to those in which payment card symbols are displayed to the customer, whenever she is asked to make a payment. Consequently, a specification is not acceptable if it leads to states in which the above is violated.

4.3.3 Goal, Quality Constraint, Softgoal

The concepts of goal and quality constraint are introduced to cover the classical taxonomic dimension for the requirement concept – namely, the distinction between the notions of functional and nonfunctional requirement. It is widely accepted in RE that functional requirements refer to what the system does, as opposed to how well the system does what it does. The latter is covered by nonfunctional requirements. Ex.6 thud gives a functional requirement.

(Ex.6) *Once the payment for the flight is confirmed, book the flight.*

(Ex.7) *It should be possible to fully book a flight through less than 5 different screens.*

Ex.7 indicates how well booking performs for it identifies a measure on the behavior of the system (i.e., the number of different screens the user needs to go through to book a flight). In other words, the second statement characterizes flight booking – it points to the existence of *qualities* for which only some (sets of) possible *values* are desired. It follows that the functional vs. nonfunctional distinction is grounded in the notion of quality in the sense of DOLCE. Namely, a requirement that describes qualities and constrains quality values is a nonfunctional requirement, and that the requirement is otherwise a functional requirement. This still remains within accepted intuitions in RE: any functional requirement will describe what the system does, whereas a nonfunctional requirement will refer to measures on the system's doings, thus allowing us to characterize how well the system behaves. Although uncontroversial, this separation obtains a solid foundation herein, leading to the following definitions.

Definition 4.2. *Desired content ϕ – i.e., ϕ in $D\phi$ – communicated by way of a directive speech act is a* ***quality constraint****, denoted generically* $\mathbf{q}$*, if and only if ϕ describes qualities and constrains quality values. Described qualities must have quality space with a well-defined and shared structure.*

Definition 4.3. *Desired content ϕ – i.e., ϕ in $D\phi$ – communicated by way of a directive speech act is a* ***goal****, denoted generically* $\mathbf{g}$*, if and only if ϕ neither describes qualities nor constrains quality values.*

It should be possible to verify whether a system satisfies requirements before delivering it to the stakeholders. Any goal and any quality constraint is verifiable, in the sense that the software engineer can check at any time whether the system satisfies the chosen goal or quality constraint. Goals clearly are verifiable for the software engineer can determine whether what is functionally required is indeed delivered by the system. Whether the flight is booked whenever the payment is confirmed is not a matter of cognitive bias or ill-defined verification criteria – the flight is either booked or not. It is the second sentence in Definition 4.2 that ensures verifiability for quality constraints: a quality constraint requires a well-defined quality space (e.g., possible values are known, values and their relationships are precisely defined, and so on) *and* the quality space must be shared among the stakeholders. Number of screens involved in flight booking is a quality for which there is no controversy on the structure of the quality space or on the association of the system behaviors to the values in the quality space. But nonfunctional requirements are taken to include abstract considerations such as security, safety, maintainability, convenience, and so on. The question then is: How do these abstract notions relate to our quality constraint concept? In other words, does desiring, e.g., "high convenience in flight booking" give us a quality constraint or something else?

Given that a quality (in our foundational ontology of choice) is a perceivable and measurable entity that inheres in other entities, there must be a quality space and values for, e.g., convenience in order to call it a quality, and thereby state that asking for, e.g., "high convenience" is a quality constraint. Now, we know that people have the ability to evaluate convenience since we observe that they can say to what extent a system is convenient. We also know that such evaluations are not necessarily (or rarely are) shared: while one person gives a strong favorable evaluation, another one may disagree. It is reasonable to argue that any evaluation requires that a somehow structured quality space exists. We could consequently

conclude that high convenience is a quality constraint. To do so is, however, a mistake for the following reasons:

- Divergent evaluations of the same phenomenon mean that the structure of the underlying quality space is dependent on each individual's cognitive bias (i.e., the structure is subjective).
- While there seems to be some kind of quality space at each individual, people are rarely (if ever) capable of defining and/or conveying the structure of that quality space (e.g., for convenience) in a precise manner, so that verifiability remains elusive.
- Subjective structure of the quality space counters the need for a quality space with a structure shared among the stakeholders, which would subsequently facilitate verification.

To aim the verifiability of quality constraints, we require well-defined quality spaces (e.g., as in metrics) and we need quality spaces shared among the stakeholders, or such that the stakeholders can accept to share these in relation to the system of interest. For all practical purposes, qualities take the form of measures for which we make explicit the desired values, that is, we define quality constraints. Measures will have a particular level of measurement — i.e., we can have nominal, ordinal, interval, ration or other level of measurement [185], so that there is no limit, e.g., on having necessarily measures that have rich sets of permissible transformations. We conclude thus that while quality constraints cover some of nonfunctional requirements, abstract nonfunctional requirements on, e.g., convenience, security, maintainability, and so on, are *not* quality constraints – we call them *softgoals* instead.

Definition 4.4. *Desired content ϕ – i.e., ϕ in $D\phi$ – communicated by way of a directive speech act is a* ***softgoal****, denoted generically* $\hat{\boldsymbol{q}}$*, if and only if ϕ describes qualities or constrains quality values, whereby the described qualities must have a quality space with a subjective and/or ill-defined structure.*

The following statement, communicated by way of a directive speech act gives us a softgoal.

(Ex.8) *Flight booking should be convenient.*

A salient characteristic of softgoals is that they cannot be satisfied to the ideal extent, not only because of subjectivity, but also because the ideal level of satisfaction is beyond the resources available to (and including) the system. It is therefore said that a softgoal is not satisfied, but *satisficed* – that is, the software engineer seeks the specification that satisfies to the highest extent compared to the *considered* alternatives. This contrasts to seeking (under resource constraints) the best among all possible alternatives, which amounts to optimization.

To acknowledge, however, that some nonfunctional requirements – softgoals – relate to qualities with subjective and/or ill-defined quality spaces does not advance the matter on how to actually ensure that satisficing occurs. Mylopoulos and colleagues [225] introduced so-called contribution links between goals and softgoals to indicate that bringing about states in which the goal is satisfied contributes positively or negatively to the satisficing of the softgoal. The aim with contribution links is to enable the engineer to claim that satisfying goals in some particular way leads to some degree of satisfaction of the softgoals. While contribution links are attractive for they allow side-by-side comparison of alternative system structures (e.g., [50]), their semantics remain vague, which has led to various interpretations and use. To properly verify whether and to what extent satisficing occurs, we need to understand more precisely how goal, quality constraint, and softgoal are related. This is necessary in order to show later on (§4.4) the role of softgoals in the requirements problem.

When the engineer aims to ensure convenience of the flight booking process, she will try to understand how various values of measurable system behaviors affect stakeholders' perception of convenience. For instance, she may consider that 6 screens for flight booking correspond to a threshold level of convenience, and that convenience improves as the number of screens reduces, whereby anything less than 3 screens is unacceptable for the user is expected to fill out too big a form over too few screens. Doing this amounts to *approximate* a subjective and/or ill-defined quality space by one or more other, well-defined and shared quality spaces. A contribution link between quality constraints over the latter and the softgoals over the former thus indicates that the said quality constraints approximate the degrees of satisfaction of the given softgoals. Hence, there is correlation between the values of the quality spaces underlying the quality constraints, and the values in quality spaces underlying the softgoals, which leads us to the following definition for contribution links within our ontology.

Definition 4.5. *A* ***contribution link*** *exists and is directed from a quality constraint* $\boldsymbol{q}$ *to a softgoal* $\hat{\boldsymbol{q}}$*, denoted* $\boldsymbol{c}(\hat{\boldsymbol{q}}, \boldsymbol{q})$*, if and only if there is correlation between values in the quality space of* $\boldsymbol{q}$ *and quality space of* $\hat{\boldsymbol{q}}$*.*

There are two important nuances to understand at this point. First, classical contribution links in RE ([225]) are labeled: e.g., if we choose a system design in which there will be less than 5 screens for flight booking, we will label the contribution link as positive (we would draw the symbol '+' on it). The label is not itself related to the sign of the correlation (e.g., the number of screens for flight booking is negatively correlated with the 'level' of convenience) and there is no absolute rule for deriving one from the other. Second, not all approximations (and thereby contributions and their labels) are equivalently appropriate or acceptable to the stakeholders. Evidently, the engineer cannot prove (in the formal sense) that it is indeed relevant to approximate convenience with the number of screens in flight booking and that the correlation is as described above; the closest feasible solution instead is to *justify* that a quality referred in a quality constraint approximates a quality referred in a softgoal. In other words, we are interested in justifying that a quality constraint approximates a softgoal, whereby the justification applies under two caveats: approximation stands *for all practical purposes within the given development project*, and approximation is justified only on grounds of the information *available* to the software engineer (i.e., new information – e.g., more details on how stakeholders evaluate convenience – may lead the engineer to revise the approximation that was previously justified). The first caveat means that approximating convenience by (among others) the number of screens for booking is justified only locally, within the given project (or even part thereof; it may be appropriate elsewhere, but that remains to be justified); the second caveat means that the approximation can become irrelevant, e.g., if the engineer finds out that stakeholders care in no way for the number of screens, and that convenience is merely related to the length of the form to fill when booking a flight.

Definition 4.6. *There is a* ***justified approximation****, denoted* $\boldsymbol{jApprox}(\hat{\boldsymbol{q}}, \boldsymbol{q})$ *if and only if there is a justification for the claim "*$\boldsymbol{q}$ *approximates* $\hat{\boldsymbol{q}}$*".*

Broadly speaking, we have $\mathbf{jApprox}(\hat{\mathbf{q}}, \mathbf{q})$ if there is no reason to believe that $\mathbf{q}$ does not approximate $\hat{\mathbf{q}}$. What justification provides is a structured process by which we can determine whether there is enough support for the claim that $\mathbf{q}$ approximates $\hat{\mathbf{q}}$. For details, the reader is referred to, e.g., Simari and Loui's seminal work [288] (see also [53], and for recent discussions of justification in RE, see [147]). In conclusion to the issue mentioned earlier, to claim that softgoals are satisficed, the engineer must identify quality constraints that justifiably approximate these softgoals.

Compared to contribution links herein, classical contribution links in RE fail to recognize the importance of justification and correlation between values in the respective quality spaces. Classical contribution links are indirect when they link goals to softgoals – it is more appropriate to speak of correlation between measures over the system behaviors that satisfy goals (given by qualities referred in quality constraints) and softgoals, than to abstract from quality constraints, as is the case in classical contribution links. Our ontology highlights the importance of correlation and justification for the existence of contributions, while making explicit the rationale behind the indirect, classical contribution links and thereby not rejecting these (only adding important details thereto: correlation ensures that a classical contribution link exists, and justified approximation gives the label to that contribution link).

4.3.4 Plan

Stakeholders and the system-to-be commit to act in the aim of satisfying requirements and without violating the domain assumptions. Given the content of communicated intentions, the specification will delimit the ways in which the involved parties will bring about the desired and allowed states of the world. We use the term *plan* to denote the content of communicated intentions, and this regardless of granularity (i.e., we can have primitive and composite plans – but this is of no interest in the present discussion).

Definition 4.7. *Intended content* ϕ *– i.e.,* ϕ *in* $\mathit{l}\phi$ *– communicated by way of a commissive speech act is a* ***plan****, denoted generically* $\boldsymbol{p}$*.*

Plans fit with the concepts introduced above as follows: the system and the stakeholders commit to execute plans in order to satisfy goals, whereby measures defined over plan executions act as qualities for which quality constraints can be defined. A specification – in the usual sense of documentation describing what the system does – amounts to a definition of plans.

We are interested in plans that satisfy generic goals, such as "book a flight", "schedule a meeting" or "fulfill a book request", where the goal is instantiated at runtime for each particular flight to be booked, meeting to be scheduled, or book that is requested. Given a vector x of parameters, we write $\mathbf{r}(\mathsf{x})$ the goal that takes the given parameters when instantiated: e.g., x in "Book a flight" goal may involve departure

and arrival dates and locations; when scheduling a meeting, the parameters would include the initiator, the time of the request, the list of participants, as well as the purpose of the meeting; for book requests, parameters would refer the author and title of the book, also the requestor and the time of the request.

Plans ought to be defined in such a way that their proper execution brings about states of the world in which goals verify and domain assumptions are not violated, all the while assuring that all quality constraints (including those that justifiably approximate softgoals) are satisfied. We can draw parallels here with Zave and Jackson's formulation of the requirements problem. For simplicity, let us leave out attitudes (we include them later - see §4.4) and recall that we would have $\mathbf{D}, \mathbf{S} \vdash \mathbf{R}$ according to Zave and Jackson [339]. Given the ontology we built up to now, it is tempting to suggest that the RE effort is successful once the engineer finds the specification $\mathbf{P}$ (that includes all relevant plans $\mathbf{p}$) such that $\mathbf{D}, \mathbf{P} \vdash \mathbf{G}, \mathbf{Q}$, whereby $\mathbf{Q}$ includes, for all $\hat{\mathbf{q}}$, one or more $\mathbf{q}$ standing in the justified approximation relationship to $\hat{\mathbf{q}}$ (i.e., for each $\hat{\mathbf{q}}$, we have $\mathbf{q}$ such that $\mathbf{jApprox}(\hat{\mathbf{q}}, \mathbf{q})$). $\mathbf{D}$ includes the domain assumptions $\mathbf{d}$, $\mathbf{P}$ carries the plans $\mathbf{p}$, and $\mathbf{G}$ incorporates the goals $\mathbf{g}$. The relation $\vdash$ is a monotonic one, so that (i) $\mathbf{D}$, $\mathbf{P}$, $\mathbf{G}$, $\mathbf{Q}$ must define a sound and complete theory, and (ii) we accept that if $\mathbf{D}, \mathbf{P} \vdash \mathbf{G}, \mathbf{Q}$, then, e.g., for $\mathbf{D} \subseteq \mathbf{D}'$, we still have $\mathbf{D}', \mathbf{P} \vdash \mathbf{G}, \mathbf{Q}$; in other words, monotony ensures that conclusions can never be "undone" by new information. Evidently, we can hardly ever guarantee to have a sound and/or complete theory for any but the smallest of toy systems, and we tend to continually encounter in practice new information that defeats previously valid conclusions (when, e.g., requirements are revised or domain assumptions change). Both $\mathbf{D}, \mathbf{S} \vdash \mathbf{R}$ and $\mathbf{D}, \mathbf{P} \vdash \mathbf{G}, \mathbf{Q}$ are thus removed from reality. Taking a non-monotonic consequence instead of the monotonic one resolves this problem, for it acknowledges that sound and complete theories are elusive for any realistic system, and that monotony does not apply. Taking, for example, the defeasible consequence relation $\mid\!\sim$ (as in [288]) for non-monotonic satisfaction, we can suggest that the following conditions ought to hold for generic goals to be satisfied and domain assumptions to hold (for all allowed parameter values in all parameterization vectors of the form x):

1. There are quality constraints $\mathbf{q}$ in $\mathbf{Q}$ that justifiably approximate all softgoals $\hat{\mathbf{q}}$ in $\hat{\mathbf{Q}}$.
2. $\mathbf{D}, \mathbf{P} \mid\!\sim \mathbf{G}, \mathbf{Q}$.

If the above do hold, the software engineer can claim it justified to believe that all goals and quality constraints will be satisfied if the plans that the involved parties execute at runtime do not violate $\mathbf{P}$. In other words, if the various parties succeed in their intentions, the desired and allowed states of the world will be reached and this while reaching values of qualities referred to in quality constraints. Having non-monotony instead of monotony indicates that there is no proof for this claim, merely that there is enough evidence – within the information available to the stakeholders and the software engineer and within the scope of the given development project – for accepting the given claim. It is not necessary in practice for $\mathbf{P}$ to describe the actual plans that need to be taken: we encounter situations in which we know how precisely one of the parties ought to act, while the best we can do for other parties is to restrict potential ways in which they can act — e.g., we often treat third party web services as black boxes of functionality, so that our *plan* would be written as logical constraints on the inputs and outputs of the service.

4.3.5 Attitude

The content of expressive speech acts gives attitudes. Written down, an attitude amounts to a description of an evaluation in terms of degree of favor or disfavor [80]. Such degrees vary in sign (positive or negative) and in intensity, whereby the intensity of the valuation is relative: considering an object of attitude on its own involves implicit comparison to a set of objects perceived by the evaluator to be of the same kind [158]. As Kahneman and colleagues observe [158], "objects of attitudes [as the term is used psychology] include anything that people can like or dislike, wish to protect or to harm, to acquire or to reject". Stakeholders can thus evaluate favorably or disfavorably, and with different intensity individual or alternative domain constraints (Ex.9), plans (Ex.10), goals (Ex.11), quality constraints (Ex.12), or softgoals (Ex.13).

(Ex.9) *Rule in Ex.1 is unacceptable for it limits the pricing options in case of promotions.*

(Ex.10) *Having the system confirm booking is preferred to having a person do it.*

(Ex.11) *It would be good if the user is informed of special flight offers.*

(Ex.12) *It is preferred to split the flight booking form over several screens than to show it fully one a single screen.*

(Ex.13) *Convenience of flight booking is more important than speed.*

As the software engineer hopes to construct high quality systems, she must be interested in attitudes: attitudes reflect stakeholders' level of satisfaction with plans, goals, domain constraints, and so on, thereby guiding the engineer during the decision-making on whether and how to define plans to satisfy goals, softgoals, quality constraints, and avoid violating domain constraints. What the engineer must have in order to rate alternative specifications is therefore an explicit account of stakeholders' attitudes.

Definition 4.8. *Attitudinal content ϕ communicated by way of an expressive speech act – i.e., ϕ in $A\phi$ – is an **attitude**, denoted generically **a**, if and only if it evaluates in terms of favor or disfavor one or more elements constituting **D**, **P**, **G**, **Q**, or $\hat{\mathbf{Q}}$.*

It is important to understand how the introduction of attitude changes our ontology defined up to this point. Any attitude **a** – by being an evaluation over components of **D**, **P**, **G**, **Q**, or $\hat{\mathbf{Q}}$ – is not a concept *per se*, but either partitions **D**, **P**, **G**, **Q**, or $\hat{\mathbf{Q}}$, or establishes orders between components of the same type (we do not consider in this paper mixed orders, in which we could determine precedence between some **d** and **g**, but only orders between different **d**, or between different **g**). The difference between these two roles of attitudes stems from the possibility to evaluate individually some component of either **D**, **P**, **G**, **Q**, or $\hat{\mathbf{Q}}$ (as in Ex.9 and Ex.11), and the possibility for that evaluation to amount to a comparison between different components of the same type (as in Ex.10, Ex.12, and Ex.13). We call the first role *optionality* and the second *preference*.

Optionality. Favorable or disfavorable evaluation of a single (i.e., independently of others) **d**, **p**, **g**, **q**, or $\hat{\mathbf{q}}$ is of interest to the software engineer to the extent that it indicates whether it is necessary to define plans that accord with the given **d**, **p**, **g**, **q**, or $\hat{\mathbf{q}}$. In other words, the engineer is interested in knowing if some **d**, **p**, **g**, **q**, or $\hat{\mathbf{q}}$ is *compulsory* or *optional*. It is compulsory if the stakeholders cannot accept a specification that does not account for that particular **d**, **p**, **g**, **q**, or $\hat{\mathbf{q}}$; otherwise, it is optional. Consequently, optionality completely partitions each of **D**, **P**, **G**, **Q**, and $\hat{\mathbf{Q}}$ onto optional and compulsory parts: **D** on $\mathbf{D_O}$ and $\mathbf{D_C}$, **G** on $\mathbf{G_O}$ and $\mathbf{G_C}$, and so on. The dichotomy optional/compulsory arises from the intensity of evaluation: favorable or disfavorable evaluation involves undoubtedly an ill-structured and subjective evaluation space, so that the chosen dichotomy serves to partition that space and thereby simplify the use of evaluations over individual **d**, **p**, **g**, **q**, or $\hat{\mathbf{q}}$. The actual intensity becomes of interest once tradeoffs appear – i.e., when it is necessary to determine which of two or more optional or compulsory, yet conflicting goals, plans, or otherwise, need to be satisfied. In case of conflict, we speak of preferences and do so regardless of optionality. Note that optionality is a means for comparing alternative specifications, whereby those that satisfy "more" among the optional **d**, **p**, **g**, **q**, or $\hat{\mathbf{q}}$ is more desired than another that satisfies "less" of these, all other things being equal.

Preference. Evaluation that compares two or more instances of the same concept in our ontology (e.g., two different goals) introduces an order between these, and amounts to what we call a preference (order). Ex.10, Ex.12, and Ex.13 establish preference orders as each compares two alternatives in terms of favor or disfavor. Choice of appropriate properties for the model of the preference order (e.g., whether it is transitive) is not of interest in this paper. It is important to note that preferences can be defined over preferences; this is needed when all preferences cannot be simultaneously satisfied. For instance, lower payment verification time may correlate with lower payment security in a flight booking system. When aiming to satisfy one preferred requirement negatively affects the ability to satisfy some other requirement, we say that the involved requirement preferences are conflicting. If the conflict cannot be alleviated through a different system design (e.g., new payment verification servers with better performance both in payment verification and security), the relative importance of the conflicting preferences should be determined. The importance relation between requirement preferences is merely another kind of preference where higher importance corresponds to higher desirability. The following two preference orders over goals give a conflict:

(Ex.14) *Issuing electronic flight tickets is preferred to issuing paper flight tickets.*

(Ex.15) *It is preferred that all travelers have paper tickets than only those who have had access to a printer after the booking.*

We must state which of the two preferences is more desirable to follow when choosing among alternative plans. If the preference obtained from the second statement is more important, then a system design which satisfies the first will be less desirable than a system design which satisfies the second but does not satisfy the first. When we say that a preference is satisfied, we mean that the most desirable alternative among the ordered ones is chosen.

4.4 Requirements Problem Revisited

Returning to Zave and Jackson's formulation, if we seek $\mathbf{S}$ such that for elicited $\mathbf{D}$ and $\mathbf{R}$ we have $\mathbf{D}, \mathbf{S} \vdash \mathbf{R}$, we implicitly assume that $\mathbf{D}$, $\mathbf{S}$, and $\mathbf{R}$ are precise and complete enough for the satisfaction relation to hold. We have noted earlier that this is hardly a plausible assumption in practice (§4.3.4), and we suggested to use a non-monotonic consequence $\mathbf{\sim}$ instead of a monotonic one $\vdash$. This adjustment does not respond to the problem of eliciting and using tentative evaluations – i.e., attitudes seem to be irrelevant as they are not covered in $\mathbf{D}, \mathbf{P} \mathbf{\sim} \mathbf{G}, \mathbf{Q}$. They, however, clearly are relevant: the quality of a system is evaluated by the stakeholders over the system's entire lifecycle, leaving the engineer to anticipate these evaluations by eliciting – during development – evaluations (i.e., attitudes) and assume that these are representative of the future evaluations.

We have introduced two ways in which attitudes intervene within our ontology, namely through optionality and preference. The former leads us to partition instances of the various ontology concepts into either optional or compulsory, whereas we will denote all preferences by $\mathbf{A}^{\succ}$. Given some preference order $\mathbf{a}^{\succ}$ in $\mathbf{A}^{\succ}$ stating saying that to satisfy some $\mathbf{g}'$ is preferred to satisfying the alternative $\mathbf{g}$, we prefer – all other things being equal – the specification $\mathbf{P}'$ in which plans lead to states described by $\mathbf{g}'$ to the specification $\mathbf{P}$ in which plans lead to states described by $\mathbf{g}$. We can then say that, given $\mathbf{A}^{\succ}$, we seek the most desirable feasible $\mathbf{P}$. Following the definitions outlined above, the most desirable $\mathbf{P}$ is one that brings about the desired and allowed states (that are delimited by requirements and domain assumptions) that rate highest on stakeholders' attitudes. We thus seek $\mathbf{P}$ that satisfies highest ranked requirements and domain assumptions. Since we can have preferences over conflicting preference orders, $\mathbf{P}$ will ideally satisfy highest ranked alternatives that these preferences define. Attitudes that concern the dichotomy optional/compulsory resolve a different problem with Zave and Jackson's formulation: if $\mathbf{D}, \mathbf{S} \vdash \mathbf{R}$ verifies, then all $\mathbf{R}$ obtains – in other words, writing $\mathbf{D}, \mathbf{S} \vdash \mathbf{R}$ makes all requirements compulsory. The optional status is impossible, even though it is clearly encountered in practice.

We can now provide the revised formulation of the requirements problem that accounts for all remarks and the ontology discussed in this paper. Below, $\mathbf{D}$ denotes the finite set of all communicated domain assumptions in a given project. Since we can have alternative domain assumptions (i.e., $\mathbf{d}$ is alternative to $\mathbf{d}'$ if there are no possible states in which both can hold – note that $\mathbf{d}$ and $\mathbf{d}'$ are inconsistent), we can partition $\mathbf{D}$ into a finite number of consistent subsets $\mathbf{D}^i$. Each $\mathbf{D}^i$ therefore contains one alternative for each set of alternatives in $\mathbf{D}$ (i.e., a $\mathbf{D}^i$ contains either $\mathbf{d}$ or $\mathbf{d}'$). We have the same notational convention for $\mathbf{P}$, $\mathbf{G}$, $\mathbf{Q}$, $\hat{\mathbf{Q}}$, and $\mathbf{A}$. The i-th consistent subset of the compulsory domain assumptions is thus denoted $\mathbf{D}^i_{\mathbf{C}}$.

Definition 4.9. *Starting from compulsory and optional domain assumptions ($\boldsymbol{D_C}$ and $\boldsymbol{D_O}$), goals ($\boldsymbol{G_C}$ and $\boldsymbol{G_O}$), quality constraints ($\boldsymbol{Q_C}$ and $\boldsymbol{Q_O}$), softgoals ($\hat{\boldsymbol{Q}}_{\boldsymbol{C}}$ and $\hat{\boldsymbol{Q}}_{\boldsymbol{O}}$), plans ($\boldsymbol{P_C}$ and $\boldsymbol{P_O}$), and attitudes ($\boldsymbol{A}^{\succ}_{\boldsymbol{C}}$ and $\boldsymbol{A}^{\succ}_{\boldsymbol{O}}$) communicated by the stakeholders, the requirements problem amounts to finding the specification (i.e., plans) ($\boldsymbol{P}^*$) such that:*

1. $\boldsymbol{D}^*, \boldsymbol{P}^* \mathbf{\sim} \boldsymbol{G}^*, \boldsymbol{Q}^*, \boldsymbol{A}^{\succ}$.
2. *Preferences in $\boldsymbol{A}^{\succ}$ indicate that that there is no other combination ($\boldsymbol{D}^i_{\boldsymbol{C}}, \boldsymbol{G}^i_{\boldsymbol{C}}, \boldsymbol{Q}^i_{\boldsymbol{C}}$) that is preferred to ($\boldsymbol{D}^*_{\boldsymbol{C}}, \boldsymbol{G}^*_{\boldsymbol{C}}, \boldsymbol{Q}^*_{\boldsymbol{C}}$).*
3. *There is no $\boldsymbol{P}^i$ different from $\boldsymbol{P}^*$ such that $\boldsymbol{D}^*_{\boldsymbol{C}}, \boldsymbol{P}^i \mathbf{\sim} \boldsymbol{G}^*_{\boldsymbol{C}}, \boldsymbol{Q}^*_{\boldsymbol{C}}, \boldsymbol{A}^{\succ}_{\boldsymbol{C}}$ verifies, and*
   a) *$\boldsymbol{D}^*_{\boldsymbol{C}}, \boldsymbol{D}^i_{\boldsymbol{O}}, \boldsymbol{P}^i \mathbf{\sim} \boldsymbol{G}^*_{\boldsymbol{C}}, \boldsymbol{G}^i_{\boldsymbol{O}}, \boldsymbol{Q}^*_{\boldsymbol{C}}, \boldsymbol{Q}^i_{\boldsymbol{O}}, \boldsymbol{A}^{\succ}_{\boldsymbol{C}}, \boldsymbol{A}^{\succ}_{\boldsymbol{O}}$ verifies, where $\boldsymbol{D}^i_{\boldsymbol{O}}$ is a larger subset of $\boldsymbol{D_O}$ than $\boldsymbol{D}^*_{\boldsymbol{O}}$, and/or $\boldsymbol{G}^i_{\boldsymbol{O}}$ is a larger subset of $\boldsymbol{G_O}$ than $\boldsymbol{G}^*_{\boldsymbol{O}}$, and/or $\boldsymbol{Q}^i_{\boldsymbol{O}}$ is a larger subset of $\boldsymbol{Q_O}$ than $\boldsymbol{Q}^*_{\boldsymbol{O}}$, and/or*
   b) *Preferences in $\boldsymbol{A}^{\succ}$ indicate that $\boldsymbol{D}^i_{\boldsymbol{O}}$ is preferred to $\boldsymbol{D}^*_{\boldsymbol{O}}$, and/or $\boldsymbol{G}^i_{\boldsymbol{O}}$ is preferred to $\boldsymbol{G}^*_{\boldsymbol{O}}$, and/or $\boldsymbol{Q}^i_{\boldsymbol{O}}$ is preferred to $\boldsymbol{Q}^*_{\boldsymbol{O}}$.*
4. *For each softgoal $\hat{\boldsymbol{q}}$ in $\hat{\boldsymbol{Q}}$, there is a quality constraint $\boldsymbol{q}$ that stands in the justified approximation relationship with the softgoal, i.e., $\boldsymbol{jApprox}(\hat{\boldsymbol{q}}, \boldsymbol{q})$.*
5. *For each preference order in $\boldsymbol{A}^{\succ}$ over softgoals, there is a preference order that maintains that same same ordering over quality constraints that stand in the justified approximation relationship to the given softgoals.*

Note that the definition assumes (e.g., point 2 in Definition 4.9) that preferences are combined to obtain an aggregate preference over subsets of $\mathbf{D_C}$, $\mathbf{G_C}$, and $\mathbf{Q_C}$. Observe that since we can define $\mathbf{\sim}$ in an informal logic (e.g., of argumentation [128]), the above applies if informal notation is used during RE. The engineer no longer seeks a specification which will merely satisfy compulsory requirements, but is expected to define a specification which comes as close as feasible to satisfying all compulsory

requirements and as many as feasible optional requirements, and this in the way that rates favorably as high as possible in terms of stakeholders attitudes.

4.5 Conclusions

Given the current confusion about the basic ontology in RE, this paper revisits the core concepts and proposes a series of definitions thereof. Definitions are written in a restricted vocabulary of the DOLCE foundational ontology with explicit speech acts and mental states. The scope of the core RE ontology is now determined by the kinds of speech acts that stakeholders communicate to the software engineer. The suggested concepts are shown to cover all speech acts, thereby ensuring that the software engineer can classify all content of stakeholders' communications as domain assumptions, plans, goals, quality constraints, softgoals, or attitudes.

The proposed definitions for goal, quality constraint, and softgoal concepts have several advantages. Together, these concepts cover the key taxonomy of functional and nonfunctional requirements. In contrast to previous work, it is clear which of the nonfunctional requirements are verifiable, and we provided precise conditions for the verifiability of nonfunctional requirements. All three concepts are grounded in an intuitive foundational ontology. It has not been particularly clear how measures fit within the goal and softgoal separation in RE – this issue is resolved herein. While we acknowledge – through quality constraints – the basic tenet that what cannot be measured cannot be managed, we also make it clear that not all can be directly defined or measured: convenience, security, safety, along with considerations such as fun amount to softgoals, which involve subjective and local quality spaces that are difficult to precisely describe and share. Our definitions are not specific to particular kinds of functional and nonfunctional requirements, but span much of ongoing software engineering research on these topics. We define the softgoal concept in a way that states precisely what a softgoal is within the communication between the engineers and stakeholders, and this without breaking off from the tradition of how softgoals are used in RE (established in [225] and later). We are also in line with research on software measurement and quality: it is indeed now well accepted that considerations such as security, safety, convenience, usability, maintainability, and so on, have no domain- and/or project-independent definition [175, 77]. Our departure from Zave and Jackson lies in the introduction of attitudes and our detailed discussion of functional and nonfunctional requirements – we shown (§4.4) that Zave and Jackson's **R** only includes our goals, and neither quality constraints nor softgoals. Recently, Glinz suggested that a "nonfunctional requirement is is an attribute of or a constraint on a system" [106], though it remains unclear what concepts the terms 'attribute' and 'constraint' denote. Our conceptualization has the advantages of being grounded in the actual context of RE (i.e., in the communication between the engineer and the stakeholders) and we are explicit on the meaning of our concepts.

The revised ontology leads to a new formulation of the requirements problem, that is, the fundamental problem of the RE field. We have argued that the new formulation fits intuition better and avoids the pitfalls of the previously established formulation from Zave and Jackson [339].

Established frameworks for RE lack capabilities needed for the representation of and reasoning about instances of all terms in the terminology. It is for instance unclear how to deal with the justification of approximation in methodological terms, how to elicit and analyze attitudes. Future effort is directed towards the extension of both conceptual and methodological parts of established RE frameworks to ensure that the full ontology proposed in the present paper is covered. This is a pressing concern for it is necessary to provide appropriate means for resolving the requirements problem in its revised form.

5

Achieving, Satisficing, and Excelling

Abstract. Definitions of the concepts derived from the goal concept (including functional and nonfunctional goal, hardgoal, and softgoal) used in requirements engineering are discussed, and precise (and, when appropriate, mathematical) definitions are suggested. The concept of satisficing, associated to softgoals is revisited. A softgoal is satisficed when thresholds of some precise criteria are reached. Satisficing does not cover situations in which continual improvement of thresholds is expected. The notion of *excelling* is suggested to cover such cases, along with the concept of *disposition* to represent and reason about excelling.

5.1 Outline

One motive for representing and reasoning about a system is to precisely understand the purpose thereof and subsequently use this information to identify, analyze, and select among alternative properties and behaviors needed of the system to fulfill its purpose. The *goal* concept stands out among the various abstractions proposed for the representation and reasoning about a system's purpose. It is now accepted that the concept is relevant for the elicitation, elaboration, structuring, specification, analysis, negotiation, documentation, and modification of stakeholders' requirements on a system [225, 71, 331, 11, 332, 276, 55, 197, 315, 50, 100, 198, 316].

Few contributors to the field of requirements engineering (RE) agree on a *precise* definition of the goal concept. Elasticity in definitions may facilitate the basic understanding of goal-based RE frameworks to non-experts: common knowledge substitutes for specialized RE knowledge, thus facilitating learning. Elasticity, however, also involves difficulties in communication, imprecision in intended meaning, and overuse and/or abuse of the terminology.

In response, we study definitions of the goal and its derived concepts, including hardgoal, softgoal, functional goal, and nonfunctional goal. We suggest precise definitions consistent with the literature; when appropriate, definitions are in formal logic. It is usually said that a hardgoal can be achieved, while a softgoal can only be satisficed. We revisit the concept of satisficing, commonly associated to the concept of softgoal. A softgoal is satisficed when thresholds of some precise criteria are reached. We argue that satisficing does not cover situations in which continual improvement of thresholds is preferred. We subsequently suggest the notion of *excelling* to cover such cases, along with the concept of *disposition* to represent and reason about excelling. The contributions of this paper are: (i) the set of precise definitions of goal and derived concepts of hardgoal, softgoal, functional goal, and nonfunctional goal; (ii) the notion of excelling intended to complement the concepts of achievement and satisficing in goal-oriented RE; and (iii) the concept of disposition intended for representing and reasoning about requirements to which the notion of excelling applies.

5.2 Goal Concept in RE Research

System development frameworks include, since the 1970s some form of analysis involving goals [315], among them context analysis, definition study, and participative analysis. Goals have become an essential part of any system's documentation through standards such as e.g., the IEEE-Std-830/1993. There is no established definition for the goal concept: consider Table 1, which lists informal definitions of the goal concept appearing in various goal-oriented RE frameworks. KAOS highlights the nonoperational

Table 1. Informal definitions of the *goal* concept in goal-oriented RE.

Framework	Informal definition of the goal (and derived) concepts
KAOS	"A goal is a nonoperational objective to be achieved by the composite system. Nonoperational means that the objective is not formulated in terms of objects and actions available to some agent in the system; in other words, a goal as it is formulated cannot be established through appropriate state transitions under control of one of the agents." [71] "A goal is a desired property about quantities in the environment." [197]
Tropos and i^*	"A goal is a condition or state of affairs in the world that the stakeholders would like to achieve." [331, 332, 50]
NFR	"Goals [represent] non-functional requirements, design decisions and arguments in support or against other goals." [225, 55]
REF	"According to the nature of a goal, a distinction is made between hard goals and soft goals. A goal is classified as hard when its achievement criterion is sharply defined [...]. For a soft goal, instead, it is up to the goal originator, or to an agreement between the involved agents, to decide when the goal is considered to have been achieved [...]. In comparison to hard goals, soft goals can be highly subjective and strictly related to a particular context; they enable the analysts to highlight quality issues [...] from the outset [...]" [76]
GDC	"An enterprise goal is a desired state of affairs that needs to be attained." [166]
GBRAM	"Goals are high level objectives of the business, organization or system. They capture the reasons why a system is needed and guide decisions at various levels within the enterprise." [11]
Lightswitch	"[A] maintenance goal is said to represent a condition that remains constant. [...] [An] achievement goal has definite pre and post-conditions. The pre-condition represents the interpretation that the state of affairs has drifted (or will drift) outside of the threshold associated with the norm [i.e., a variable of the system whose state the system attempts to maintain unchanged as defined by an observer]. The post condition is an interpretation that is within this threshold." [271]

nature of goals, pointing to the need for taking action to make goals precise by refinement (see, e.g., [71]). Broadly speaking, the KAOS definition is in line with those of Tropos, i^*, GDC, and Lightswitch: a goal designates desirable conditions on the system and/or its environment. Such condititions restrict the set of alternative system and environment states, so that it is appropriate to say that a goal describes desired states. A different conceptualization appears in NFR, where goals are employed for representing nonfunctional requirements, in addition to design decisions, and arguments for or against other goals. We can interpret "design decisions" as restricting potential desired system and environment states. Notions of argument and justification appear in NFR and GBRAM. We have discussed elsewhere [150] the relevance of argumentation and justification for goal-oriented RE, arguing and illustrating that it is more appropriate to maintain the notion of argument separate from the goal concept.

Regarding the use of goals for modeling nonfunctional requirements, two relevant goal taxonomies have been introduced since the seminal contributions in the NFR framework (see, e.g., [315, 151] for discussions).[1] *Functional* goals are distinguished from *nonfunctional* ones, and *softgoals* from *hard goals*. *Functional* goals have been used to represent services that the software is expected to deliver (i.e., *what* the software does), whereas *nonfunctional* goals refer to quality requirements that the software needs to satisfy while delivering the services (i.e., *how* the software provides services; e.g., securely, safely, rapidly, etc.). While it is common to equate nonfunctional goals and softgoals (e.g., [225]), it has been subsequently argued that *softgoals* belong to another taxonomy, in which they are distinguished from *hardgoals* [315]. According to the traditional definition, "a softgoal is similar to a (hard) goal except that the criteria for whether a softgoal is achieved are not clear-cut and a priori." [202] The definition used in the REF framework [76] adds details, as shown in Table 1.

While softgoal satisfaction cannot be established in a clear-cut sense [225], the satisfaction of a hardgoal is objective in that it can be established using (formal) verification techniques [71]. In this respect, a hardgoal is said to be *achievable*, whereas a softgoal is *satisficeable* [225, 55, 198]. The concept of *satis-*

[1] This paragraph follows our previous discussions on the subject [151].

ficing originates in H. Simon's work [290] in economics: to satisfice is to set a threshold, and accept any achievement above the threshold. In addition to involving satisficing, a softgoal has a subjective component, in that various stakeholders of the system will have different thresholds. We have worked on a more expressive definition of softgoals elsewhere [151], but we did not provide a mathematical definition.

The KAOS framework provides the most precise hardgoal conceptualization: a hardgoal is defined in terms of predicate patterns in a discrete linear temporal first-order logic (see, e.g., [209]).[2] A hardgoal is any one of the following [197]: an achieve hardgoal (pattern: $\phi \Rightarrow \diamond\psi$), a cease hardgoal ($\phi \Rightarrow \diamond\neg\psi$), a maintain hardgoal ($\phi \Rightarrow \Box\psi$), an avoid hardgoal ($\phi \Rightarrow \Box\neg\psi$). The same conceptualization is adopted in Formal Tropos [100], where patterns are used in the same way to define hardgoals. An informal interpretation of the said conceptualization is that a hardgoal is a constraint over system histories (i.e., behavior over time).

It is clear that the goal concept is intended to be rich in meaning. Instead then of seeking an all encompassing definition, we study derived concepts, obtained by crossing the hardgoal/softgoal and functional/nonfunctional taxonomies; we thus have: (i) *functional hardgoals*, which are hardgoals about what the system should do (e.g., in an email application, such a goal can be: "whenever an e-mail marked as important arrives, the user is informed with a pop-up window and a sound"); (ii) *nonfunctional hardgoals* which describe verifiable criteria for how the system should operate (e.g., "the user should be informed about important e-mail arrival within 1 second of arrival"); (iii) *functional softgoals* describe a subjective requirement of a stakeholder about what the system should do (e.g., "the user should be informed when an e-mail marked as important arrives"); and (iv) *nonfunctional softgoals* which indicate in a subjective and nonverifiable manner how the system should operate (e.g., "the user should be informed rapidly about the arrival of an e-mail marked as important").

In summary, there is no unique definition of goal. One reason for this is that the goal concept is intended to be rich in meaning. Variations in definitions are also due to slightly different uses of the concept in each framework. Whether a goal conceptualization is appropriate depends on how useful is the framework in which it is used. A prescriptive general definition thus seems excluded. It remains, however, of interest to seek a conceptual framework in which the derived concepts mentioned above can be used together, so that the benefits of these complementary concepts can be combined when representing and reasoning about requirements. A precise definition is already available for the hardgoal concept. We can now suggest a common ground for the cited derived concepts.

5.3 A Common Framework

Consider a toy system that has only two properties, p_1 and p_2. All possible combinations of allowed values for p_1 and p_2 define all possible states of the system. Let S_1, S_2, and S_3 be arbitrary system states, as shown in the bottom part of Figure 1. Assume that measurements are performed on the system in order to evaluate its quality. To perform measurement, we define two metrics d_1 and d_2. To relate what we observe in the system and the values of the metrics, we define mappings M_1, M_2, and M_3 between system states and value combinations of the two metrics. Since some minimum level of quality is expected, we define thresholds on metrics: in Figure 1, $t_1 \equiv d_1 \geq d_1^t$ and $t_2 \equiv d_2 \geq d_2^t$, so that the quality is above the minimal level only when the system is in state S_2 and not in the other two.

Taking the state-based conceptualization of the hardgoal concept, we define a hardgoal $hg_1 \equiv (\top \Rightarrow \Box(p_2 = p_2^*))$ as a value p_2^* of the property p_2. hg_1 is a functional hardgoal, since it says precisely what the system is expected to do (i.e., set property p_2 to value p_2^*). Returning to the informal understanding of the nonfunctional hardgoal given earlier (§5.2), we see that it cannot be defined over system properties, but on metrics defined for the system. We can thus define two nonfunctional hardgoals in Figure 1: $\tilde{hg}_1 \equiv (d_1 \geq d_1^t)$ and $\tilde{hg}_2 \equiv (d_2 \geq d_2^t)$.

Are there any softgoals in Figure 1? We know that a softgoal is used to model requirements at the earliest stages of an RE process (e.g., [225, 315, 50, 198, 151]). Usually, initial requirements, and consequently softgoals are imprecise, subjective, idealistic, and context-specific [151], meaning that we cannot have a softgoal in Figure 1—the figure is already too precise. Consequently, and in line with contributions in the RE field, a softgoal is here understood as an initial, early form of requirements about what the system should do and how "well" it should do it, from which one or more functional

[2] In publications on KAOS, what we call hardgoal here is called simply "goal". Note, however, that this conceptualization does not encompass the softgoal concept (which was introduced separately from KAOS): if we know a constraint, written in logic over system histories, we can check at any time if the actual history of the system respects or not the constraint.

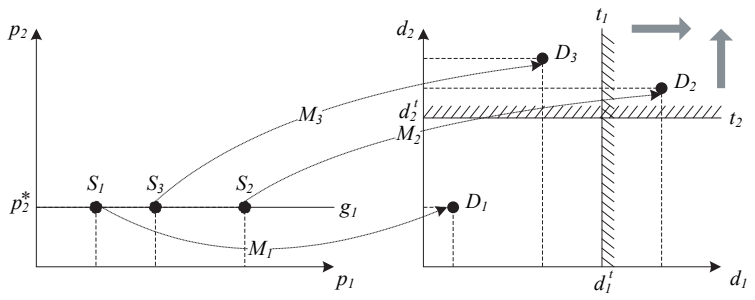


Fig. 1. Arbitrary functional and nonfunctional hardgoals in a toy system.

and/or nonfunctional hardgoals are extracted during the requirements process. This is appropriate, since we also know that one cannot manage (i.e., assure, control, or improve) what one cannot measure (e.g., [92, 176, 43, 89, 114]): if quality-related information contained in softgoals is to be used in decision-making during an RE process, we need to make a softgoal precise, agreed upon by various stakeholders, and realistic—that is, we need to convert it into nonfunctional hardgoals. Same applies for functional softgoals: we require precise, agreed upon, and realistic requirements about what services the system should deliver in order to be able to implement them in later stages of the system development process.

Having established that softgoals appear earlier in an RE process than hardgoals, recall that softgoals are associated to the concept of satisficing. Satisficing is the reason we specified our nonfunctional hardgoals as thresholds only, instead of, e.g., more elaborately stating the desired values of d_1 and d_2. Indeed, nonfunctional hardgoals derived from nonfunctional softgoals serve in RE as criteria for comparing alternative system structures (e.g., [225, 55, 315, 50]). A system structure is chosen over an alternative one if the former dominates the latter over a set of nonfunctional softgoals or the derived nonfunctional hardgoals. In our toy system, we would choose a system structure that is associated to higher values of the two metrics, over one associated with lower values; we would discard structures that are not above both thresholds. Satisficing, while clearly useful and reflecting the inability to identify *the* optimal system structure, does not cover requirements in which continuous improvement is sought. Indeed, satisficing does not go as far as to say what values above a threshold are preferred over other values, also above the threshold. That is, all values are equally desired, provided that they are above the threshold. In many actual cases, we do need to set thresholds, but we need not equally prefer all above-threshold structures. This is the case in particular for adaptable systems based on the agent or services paradigms. We encountered this need in an actual setting: we proposed elsewhere [149] an adaptable system in which above-threshold structures are learned. Therein, a "system structure" corresponds to a composition of web services that allows a service request (coming from a system user and specified in terms of requirements) to be filfiled. To form compositions, a composer web service observes other web services during execution and subsequently selects (for participation in a composition) only those that allow it to obtain more desired values over a given set of metrics. Compositions are revised, so that the quality to which same service requests are fulfilled improves over time. When specifying requirements on such a system, we are clearly not interested only in satisficing—if we did rely on satisficing only, we would not exploit the ability of the system to improve the fulfillment of service requets. We would not exploit the system's ability to adapt. Instead, we need to express that the system both needs to satisfice (so that below-threshold compositions be discarded) and to always improve compositions. We clearly cannot use the notion of satisficing to express requirements on continuous improvement: instead, we use the notion of *excelling* to do so. The limitation of satisficing that we highlight here is not novel: recently, J. L. Pollock proposed the concept of locally global planning, in which "any plan with a positive expected utility is defeasibly acceptable, but only defeasibly. If a better plan is discovered, it should supplant the original one. Satisficing would have us remain content with the original." [261] This discussion brings us to the following position: to use the concept of softgoal grounded in satisficing to express requirements that are associated to the concept of excelling is to extend the softgoal concept too far. We thus propose the concept of *disposition*. A disposition is a preference order defined over goals of the same type; we thus have the following taxonomy for the disposition concept: (i) *hard-functional disposition*, defined over functional hardgoals; (ii) *hard-nonfunctional disposition*, over nonfunctional hardgoals; (iii) *soft-functional disposition*, over functional softgoals; and (iv) *soft-nonfunctional disposition*, over nonfunctional softgoals. An example of a hard-nonfunctional disposition expressed informally is: "the user should be informed about important e-mail arrival within the least time possible" generalizes a preference order in which it is clear that the nonfunctional hardgoal "the user should be informed about important e-mail arrival within 0.5 second of

arrival" is preferred to "the user should be informed about important e-mail arrival within 1 second of arrival". Note the following:

- Do not mistake excelling for optimization: the latter applies if the email system is designed so that it always gives the optimal time (i.e., 0 seconds). This is clearly idealistic. Excelling is in a sense optimization over time and given resource boundedness; that is, the email system excels if it reduces time for informing the user at each email arrival compared to the time it needed on the last occasion an email arrived. Excelling applies even if the system does not continually improve (expecting this may be idealistic); what is important for excelling to apply is that, even if, in our example email system, notification time increases, it restablishes and goes down at some later and observable point (i.e., not indefinetly in the future).
- Not always can be a disposition so summarily expressed as in our notification time example for the email application. It may for instance happen that disjoint subsets of metric values are preferred, so that a disposition does not reduce to a formulation of desired direction for metric values. We explore in the remainder a simple notion of disposition mainly because we are introducing the concept here—extensions to its expressivity are of interest in current and future effort.

To express our various types of goals, we start from the multi-sorted first-order version of MITL [7, 125], a continuous real-time linear temporal logic. Some predefined sorts (e.g. real numbers) have a fixed interpretation that will be used to express metrics. Our logic starts from Φ, a first-order vocabulary, consisting of predicate and function symbols p, f. They can be declared as flexible (time-dependent) or rigid. As usual, constant symbols are viewed as 0-ary rigid function symbols. Starting with atomic formulas of first-order logic, we form more complex formulas as usual by closing off under truth-functional connectives (i.e., $\wedge$, $\vee$, $\neg$, and $\rightarrow$)[3], temporal operators (i.e., next $\bigcirc$, eventually $\diamond$, always $\Box$, until U and unless W) that can be indexed by a non-singular real-time constraint, existential ($\exists$) and universal ($\forall$) quantification. We denote the resulting language $\mathcal{L}$. We interpret formulas of $\mathcal{L}$ over the structure $\mathcal{T} \equiv \langle \mathcal{D}_{\mathcal{S}}, \mathcal{S}, \pi, \mathcal{H} \rangle$, where $\mathcal{D}$ gives a domain to interpret each sort, $\mathcal{S}$ is a set of states of the system, π is an interpretation assigning each predicate symbol and function symbol in Φ a predicate or function of the right arity over $\mathcal{D}$, if the symbol is declared rigid. If the symbol is declared flexible, it depends furthermore on the current state. $\mathcal{H}$ is a set of timed state sequences, $\mathcal{S}_0, I_0, \mathcal{S}_1, I_1 \ldots$ i.e. an infinite sequence of *states* and their associated interval of time, representig all possible executions of the system. These intervals I_i must partition the positive reals. To interpret first-order variables, we use a valuation function σ, which, given a variable of sort s returns its value, an element of $\mathcal{D}_{\int}$. Given a structure $\mathcal{T}$, an history h, a time t and a valuation σ, we can associate with every formula of $\mathcal{L}$ a truth value in the usual way. A formula holds in a structure if it yields true for all histories, valuations and times. We call the obtained logic $\mathbb{L}_{\mathcal{L}}$. We can now give a precise definition of functional hardgoal.

Definition 5.1. *A functional hardgoal is a formula in $\mathbb{L}_{\mathcal{L}}$ that restricts the possible histories of a given system only to those desired by system stakeholders.*

To express the nonfunctional hardgoal concept, we use *metrics*. A metric is a rigid function symbol, which will return values in a sort equipped with an order.

Definition 5.2. *A nonfunctional hardgoal is a formula in $\mathbb{L}_{\mathcal{L}}$ that restricts the values of metrics to those desired by system stakeholders.*

To relate the metrics and the behaviour of the system (recall Figure 1), we also need mappings between functional and nonfunctional hardgoals.

Definition 5.3. *A hardgoal mapping is a formula in $\mathbb{L}_{\mathcal{L}}$ over one or more functional hardgoals and one or more nonfunctional hardgoals.*

Taking the email application example, the following is a functional hardgoal, and is followed by an equivalent nonfunctional hardgoal:

$$hg \equiv [\forall m : Email\ (arrived(m) \wedge important(m)) \Rightarrow \\ \diamond_{\leq 1sec} \exists\ w : PopupWindow, s : NotifSound\ (display(w) \wedge play(s))]$$

$$\tilde{hg} \equiv [timeToNotification \leq 1sec]$$

[3] Usual abbreviations apply, e.g., $(\phi \Rightarrow \psi) \equiv \Box(\phi \rightarrow \psi)$.

We can then define a hardgoal mapping for the above as follows: $hg \equiv \tilde{hg}$.

In contrast, we understand softgoals as expressing dispositions, i.e. preference over goals of the same type. To accommodate dispositions, we extend $\mathbb{L}_{\mathcal{L}}$ in the following way. First, we add a new kind of formulas, *disposition formulas*, such that if ϕ and ψ are formulas of $\mathcal{L}$ then $\phi \succeq_d \psi$ is a disposition formula. We take here the simplest case, in which we do not allow the operator $\succeq_d$ to appear in, e.g., temporal formulas of the language. $\mathcal{L}$ extended with disposition formulas is denoted $\mathcal{L}^{\succeq_d}$. Second, we extend our structures $\mathcal{T}$ to interpret defeasible formulas: we add a function υ which maps a real number (informally understood as a utility value) to non-disposition formulae. υ is then evaluated as follows: if $\phi \succeq_d \psi$, then $\upsilon(\phi) \geq \upsilon(\psi)$, which means that ϕ is preferred to ψ.

Following the earlier example, we may have the following hard-nonfunctional disposition:

$$[timeToNotification \leq 0.5sec] \succeq_d [timeToNotification \leq 1sec]$$

We define a disposition in terms of hard and soft disposition as follows:

Definition 5.4. *A hard disposition is a disposition formula in $\mathbb{L}_{\mathcal{L}^{\succeq_d}}$ between hardgoals.*

Definition 5.5. *A soft disposition is a preference either between only functional softgoals or between only nonfunctional softgoals.*

5.4 Conclusions and Future Work

Definitions of the concepts derived from the goal concept (including functional and nonfunctional goal, hardgoal, and softgoal) used in RE are discussed, and precise (and, when appropriate, mathematical) definitions are suggested. The concept of satisficing, associated to softgoals is revisited. A softgoal is satisficed when thresholds of some precise criteria are reached. Satisficing does not cover situations in which continual improvement of thresholds is expected. The notion of *excelling* is suggested to cover such cases, along with the concept of *disposition* to represent and reason about excelling.

Although we have only presented here the simplest notion of disposition, we believe that the paper opens a particularly relevant discussion for goal-oriented RE. We hope it motivates similar efforts to ours in exploring more expressive concepts and techniques for RE. More elaborate expressions of dispositions need to be possible if excelling is to be properly accounted for; these include, e.g., conditional dispositions. We are working on extending the expressivity of the concept and are building a method to use the disposition concept in a systematic manner during the RE process. The method is intended to extend established goal-oriented RE frameworks. Tool support will be explored.

6

A More Expressive Softgoal Conceptualization for Quality Requirements Analysis

Abstract. Initial software quality requirements tend to be imprecise, subjective, idealistic, and context-specific. An extended characterization of the common Softgoal concept is proposed for representing and reasoning about such requirements during the early stages of the requirements engineering process. The types of information often implicitly contained in a Softgoal instance are highlighted to allow richer requirements to be obtained. On the basis of the revisited conceptual foundations, guidelines are suggested as to the techniques that need to be present in requirements modeling approaches that aim to employ the given Softgoal conceptualization.

6.1 Dealing with Software Quality Requirements

Ensuring the quality of software has become a major issue in software engineering research and practice since the 1970s [30]. As increasingly complex software plays a critical role in business, comprehensive and precise methods and tools are needed to create software products and services that are safe, dependable, and efficient [241].

Software quality is defined by the International Organization for Standardization [92] as the totality of features and characteristics of a software product that bear on its ability to satisfy stated or implied needs. Ensuring the quality of software therefore amounts to making sure that software behavior is in line with stated and implied needs.

It is widely acknowledged that quality needs to be taken into account early in the software development process [55, 315, 198]. Quality requires specifying stated and implied needs. Approaches focusing on ensuring quality during the development process by guiding functional requirements specification decisions by quality considerations, so that the latter justify the former, are termed process-oriented. In contrast, product-oriented approaches (e.g., [76, 139]) evaluate the quality of already developed software products, and are particularly relevant for, e.g., component selection [8].

Although a large body of work deals with quality assurance in a process-oriented manner, a non-negligible part of it relies on the usual Softgoal concept for the representation and reasoning about quality-related requirements. In doing so, procedural aspects of methods for dealing with quality during requirements engineering (RE) activities have been considerably developed, while conceptual foundations have not evolved in a notable manner. In particular, a more extensive view on the conceptualization and formalisms for representing and using quality requirements while taking into account their multi-facetted nature has not been proposed yet. We need to deal with requirements that are not only implicit, but also subjective, context-specific, imprecise, and ordered by preference.

The work presented in this paper is a step towards a more profound understanding of requirements that are expressed usually in requirements goal diagrams (such as, e.g., i^* [331]) as instances of the Softgoal concept. Overall, instances of the original Softgoal concept are seen as frequently containing information that is, not only subjective and context-specific (as assumed in the original definition), but also imprecise and involving preferences of the stakeholder who suggested the requirements modeled as the given softgoal. It is therefore suggested that the Softgoal is a multi-facetted concept that requires specialized techniques for dealing with its additional facets. This paper thus proves useful both in terms of advancing the understanding of a key concept in the RE modeling field, and in arguing that additional considerations need be taken into account when a RE method or framework that employs the Softgoal concept is being constructed and applied. Finally, the reader will undoubtedly notice that the discussion below is independent of a particular RE framework, which supports our arguments regarding the applicability of this discussion to many (at least goal-oriented) RE methods.

The paper is organized as follows. Part of the literature on the treatment of quality requirements, applicable to the discussion in this paper is first overviewed. The bulk of the paper, which discusses and revisits the original Softgoal conceptualization is then presented. A set of general guidelines on the characteristics of RE methods aiming to employ the suggested conceptualization are presented. Finally, conclusions are summarized and directions for future work are identified.

6.2 Related Work

To facilitate the discussion of related work, Table 1 gives a classification of process-oriented approaches. Formal approaches rely on formal notations such as temporal or fuzzy logic to specify nonfunctional requirements in a precise way, while semi-formal provide structured notations (unrelated to mathematical logic) that are used mainly to organize information about nonfunctional requirements. Qualitative approaches traditionally evaluate the degree of quality requirements satisfaction using subjective qualitative characterizations. In contrast, quantitative techniques focus on estimating the probability of failure of quality-related goals [198], or use informal measures of the degree to which software properties contribute to specific qualities [6]. Decision on placing some approaches in qualitative or quantitative category is based on the methods described in the cited papers; e.g., adapting a qualitative approach to use quantitative methods remains possible, but is not discussed in the literature.

Table 2. A classification of process-oriented approaches proposed in related work for ensuring quality during the software development process.

	Qualitative	*Quantitative*
Formal	[191, 203, 330]	[198, 228]
Informal	[55, 225, 278, 188]	[59, 6]

The NFR framework [225, 55] has been the first to propose the concept of Softgoal in the RE context (the original concept that is specialized for RE in NFR has a longer history—e.g., [291]) and a process for dealing with nonfunctional requirements. In NFR, Softgoals describe quality requirements in very abstract terms. They are related with contribution links to support qualitative reasoning about the degree to which alternative software properties satisfy the desired qualities. Their intuitiveness and ease of use have led to their integration in goal-oriented RE (GORE) frameworks: i^* [331], Tropos [50], GRL [202], and REF [76]. However, the Softgoal concept remains informally defined and used. Many GORE frameworks that have adopted NFR suffer from the same symptoms, as few extend the NFR Softgoal conceptualization. [59] adds a probabilistic layer to study the impact of requirements change on quality satisfaction. Others [6] use multi-criteria decision techniques to select among alternative software architectures.

Formal approaches have been proposed to provide systematic support when semi-formal techniques are considered inadequate. Instead of Softgoals, [198] is focused on software goals that are precise, but cannot be completely achieved by the software (i.e., they are idealistic). Quality variables are associated with all goals that can only be partially satisfied and objective functions are defined over these variables to indicate ideal software behavior. Quality variables seem to be metrics that measure performance of the behavior specified by the goal to which the variables are associated. Based on a sample of software operation, probabilities of satisfying a goal can be estimated—these probability values indicate the degree to which the goal is satisfied. Imprecise requirements are treated with fuzzy logic in [191, 203, 330, 228]. While fuzzy logic may be an interesting approach for formalizing imprecise requirements, it has been discussed mostly in isolation from typical RE activities, although it is not obvious how such formalisms can be integrated within existing, more extensive frameworks.

Discussion.

Expressive formalisms such as fuzzy logic have unfortunately been discussed somewhat separately from confirmed GORE methodologies and frameworks, making the us of the techniques proposed in [191, 203, 330, 228] impractical and difficult. The informed reader will also note that fuzzy logic is merely one among many approaches to imprecision. While quality requirements are indeed imprecise, they do have other characteristics that need to be accounted for during representation and reasoning. Partial satisfaction, extensively discussed in [198] is relevant, but is discussed with focus on precise goals. It should also

be noted that the NFR approach and other RE frameworks using the Softgoal concept fail to address situations in which systematic and formal treatment is required, even though the growing criticality of software increases the need for rigor.

The research presented in the remainder of this paper starts from a hypothesis that the informally defined Softgoal concept, extensively used in NFR, can be more valuable if its facets: subjectivity, context-specificity, idealism, impreciseness, and preference are characterized explicitly. Such a characterization will allow both a systematic treatment of the stated facets, and a closer integration of the proposed Softgoal analysis approach and later RE activities, such as functional goal specification. In this respect, the proposed conceptualization draw on the extensive body of related work to provide a more integrative view on the representation and manipulation of software quality information.

6.3 The Softgoal Concept Revisited

Softgoals provided by the stakeholders at the outset of the RE process, such as "the software should be fast", can be characterized as *imprecise*, *subjective*, *context-specific*, and *ideal*. Imprecision stems essentially from the inability to specify what "fast" mean, so that they could be measured. Subjectivity results from the fact that two people can evaluate the same software as being fast to a different extent. Context-specificity further entails that "fast", or "usable", "maintainable", "adaptable" (standard qualities of software [55]) will have a different meaning for each project. Finally, implicit preference information is hidden behind terms such as "fast software": various measures can be taken, but low values will be preferred. All of the above characteristics need to be taken into account to deal systematically with quality requirements. To use the Softgoal concept to represent and reason about such requirements, it is necessary to make its traditional definition more expressive and precise. The choice of the Softgoal concept is based on the illustrated usefulness of the underlying goal concept in RE activities, such as elicitation, elaboration, structuring, specification, analysis, negotiation, documentation, and modification of requirements [314].

6.3.1 Functional and Nonfunctional Goals vs. Hardgoals and Softgoals

A goal can be broadly defined as a constraint on software behavior that is desired by stakeholders involved in the software development project (e.g., [71]). Among the many proposed goal taxonomies (for an overview, see [315]), two are particularly relevant for quality requirements modeling. *Functional* goals have been used to represent services that the software is expected to deliver (i.e., *what* the software does), whereas *nonfunctional* goals refer to quality requirements that the software needs to satisfy while delivering the services (i.e., *how* the software provides services; e.g., securely, safely, rapidly, etc.). While it is common to equate nonfunctional goals and softgoals (e.g., [225]), it is suggested that *softgoals* belong to another taxonomy, in which they are opposed to *hardgoals* [315]. Although softgoal satisfaction cannot be established in a clear-cut sense [225], the satisfaction of a hardgoal is objective in that it can be established using (formal) verification techniques [71]. Consequently, there are: (i) *functional hardgoals*, which are objective goals about services that software needs to deliver (e.g., "whenever an e-mail marked as important arrives, the user is informed with a pop-up window and a sound"); (ii) *nonfunctional hardgoals* which describe objective criteria for how the services are to be delivered (e.g., "the user should be informed about important e-mail arrival within 1sec"); (iii) *functional softgoals* describe imprecisely stated software services (e.g., "the user should be informed when an e-mail marked as important arrives"); and finally, (iv) *nonfunctional softgoals* characterize imprecise statements for how a service is to be delivered (e.g., "the user should be informed rapidly about the arrival of an e-mail marked as important"). It is likely that statements about the needs that the software is to satisfy will be closer to nonfunctional softgoals than to functional hardgoals at the outset of the RE phase of software development. Notice that there is a large gap in precision between nonfunctional softgoals and functional hardgoals example: the former says nothing on how the user is to be informed, while the latter gives a specific context (the e-mail reader software) and process (e-mail arrives, pop-up is displayed, and a sound is played). Having clarified the informal meaning of Softgoal in relation to goal types, we proceed to its characterization.

6.3.2 Characterizing Softgoals

The traditional view of softgoals [202] focuses essentially on the subjectivity facet:

> "A softgoal is similar to a (hard) goal except that the criteria for whether a softgoal is achieved are not clear-cut and a priori."

A definition proposed in the REF framework [76] adds details:

> "For a soft goal [...] it is up to the goal originator [i.e., the agent wishing goal achievement], or to an agreement between the involved agents, to decide when the goal is considered to have been achieved [...]. In comparison to hard goals, soft goals can be highly subjective and strictly related to a particular context; they enable the analysts to highlight quality issues (e.g., the concept of a 'fast computer') from the outset, making explicit the semantics assigned to them by the stakeholders."

Softgoals therefore involve subjectivity because of a lack of objective achievement criteria, and the responsibility for evaluating their achievement falls on stakeholders. Notice that it is imprecise to say that quality considerations can mainly be modeled with softgoals, since quality refers to software behavior that can be both objectively and subjectively evaluated. However, there is more to quality requirements than the current softgoal conceptualization allows representing and reasoning about. Consider a simple example of a quality requirement often encountered in practice: "the software should be fast". By examining the stated and implied information contained in this statement, a notation can be proposed to model the various softgoal facets.

Subjectivity.

Since many stakeholders (i.e., parties being influenced by, or having an influence on the development project) are likely to be involved in the RE phase of software development, the specificity of each these parties' views on the software, the development process, and the environment in which the software will operate needs to be accounted for. The usefulness of separating stakeholders' concerns and the use of adapted, different notations is now widely accepted. Such multi-perspective requirements require techniques for making individual views consistent either by reconciling requirements specifications written in different specification languages (e.g., [230]), or written different styles, terminology, etc. (e.g., [141]). The resulting heterogeneous representations need to be integrated to ensure consistency [314], coordination and composition [231].

We argue that subjectivity in softgoals can be accounted for in a relatively straightforward manner by annotating the softgoal with an identifier of its stakeholder and the suggestion time. Then each stakeholder can refine his requirement (by answering, e.g.: "When is this software fast for you?") independently.

Softgoal: E-Mail reader should be fast.
Added on: 08Nov2005
Stakeholder: Mr. J. Smith
Refined into: An e-mail reader is fast if it opens quickly and creates new e-mail messages quickly.

If a similar softgoal is stated, our approach will see it as different:

Softgoal: E-Mail reader should be fast.
Added on: 08Nov2005
Stakeholder: Mr. J. Smith
Refined into: An e-mail reader is fast if it opens quickly and creates new e-mail messages quickly.

Context-Specificity.

At an abstract level, information about the context to which the quality requirement refers can be specific to: the software, the software development process, and the environment in which the software will operate (which can be the hardware environment, the human environment, etc.). For example, "development cost should be low" is a softgoal related to the software development process, whereas "the throughput should be high on the production line" is specific to the human environment in which the software will operate. The combination of the software and environment compose the *information system* (IS) [339]. To specify the context of a softgoal, an attribute *applies to* (with *software*, *environment*, *process* as allowed values) is added to the softgoal template:

Softgoal: E-Mail reader should be fast.
Added on: 08Nov2005
Stakeholder: Mr. J. Smith
Refined into: An e-mail reader is fast if it opens quickly and creates

new e-mail messages quickly.

Applies To: Software

Idealism and Preferences.

Quality requirements are often not clear-cut. It is thus beneficial to measure the degree to which a quality requirement, modeled as a softgoal, is satisfied. Metrics, called quality variables in [198], can be designed based on refined requirements. Consider the earlier Mr. J. Smith's Softgoal. It can be refined into two Softgoals, each having a quality variable and an objective function.

Softgoal: The E-Mail reader should open fast.

...

Preferences:

Objective Functions:

Name	Def	Type	Modal	Target	Threshold	Current
3SecOpen	P(TimeToOpen $<$ 3sec)	Prob	Max	80%	70%	*unknown*

Quality Variables:

TimeToOpen: *Duration*

Sample space: distribution of old e-mails, size of old e-mails,...

Definition: time between the input of the request to open the software and the moment the software functionalities can be used.

Softgoal: It should be possible to create new e-mail messages quickly.

...

Preferences:

Objective Functions:

Name	Def	Type	Modal	Target	Threshold	Current
2SecCreate	E(TeToCrMail) $<$ 2sec	Durat	Min	1Sec	2Sec	*unknown*

Quality Variables:

TimeToCrMail: *Duration*

Sample space: number of options available when writing an e-mail,...

Definition: time between the input of the request to create a new e-mail message and the moment its content can be written.

Quality variables are random variables whose distribution can be estimated using data collected by experimentation. Sample spaces can be, e.g., related to similar functionality in existing software. The estimated probability distribution functions are then used to estimate the probability of satisfying the softgoal to some desired level and questions such as, e.g., "What is the probability for the software to open in less than 2 seconds?" or "Under what time will the software open in 90% of cases?" can be answered. Objective functions are associated with quantifiable quality variables, and target levels of performance for each variable are specified. A modal (i.e., *max* or *min*) is also added to indicate the preferred direction. Because not all objective functions are stated in terms of probabilities (i.e., there are objective functions defined over quality variables), the tables used in [198] to specify objective functions are extended here with a *type* column, to give an indication on the type of variable used in the objective function. In addition, a threshold column is added to further distinguish acceptable from unacceptable degree of softgoal satisfaction.

Quality variables combined with objective functions as in [198] allow the degree of softgoal satisfaction to be measured in cases in which the degree of satisfaction is not under stakeholders' complete control (i.e., there is a probabilistic component in the events affecting softgoal satisfaction). While this is often the case, the RE phase will also involve preferences that can be perceived as deterministic. Assume that a stakeholder provides the following view of a softgoal:

Softgoal: Software should not take too much hardware resources.

...

Stakeholder's view: An e-mail reader will take little hardware resources if it does not occupy memory when not running (i.e., it does not run "in the background").

The stakeholder expresses preference in the quality requirement modeled with the above softgoal. Traditional economics preferences conceptualization (e.g., [186]) can be used to make the preference information from the stakeholders' view explicit. Implicitly, a statement of preference provides partial information about alternatives that are to be ordered using preference relations. We use the classical preference for-

malism to indicate strict, partial or indifference preference. Using this simple formalism, we can write:

Softgoal: Software should not take too much hardware resources.

...

Preferences:

Choice Preferences:

(run software when requested)$\succ$(software runs in background)

Modeling preferences using objective functions can refine the preference formalization given above. For example, "software should not take too much hardware resources" indicates that the degree of hardware resources used by the software would ideally be measured to determine the degree to which alternative software structures would satisfy the softgoal. Consequently, the above partial softgoal specification can be improved by adding a quality variable that can be used to quantify the "too much" term. Notice that the choice between alternatives specified in *choice preferences* influences the value of quality variables.

Imprecision.

Without an accurate notion of the stated and implied needs, the degree of quality satisfaction by software cannot be measured and software properties that could satisfy quality requirements cannot be determined. The use of fuzzy logic has been suggested to formalize imprecise requirements to allow for conflicts between them to be studied ([191, 203, 330, 228]) unfortunately outside the goal-oriented RE field. A key limitation of this approach (see, [198] for a discussion) is that the degree of imprecise requirements satisfaction, measured through a "satisfaction function" that maps software behavior to a degree (comprised between 0 and 1) to which it satisfies a fuzzy (in the sense of [335]) requirement is measure-independent. There are no specific metrics involved, and it is not obvious how the measurement could be made objective, as in [198]. It would be beneficial if objective metrics and fuzzy logic notation can be combined to express formally the information given in the softgoal template.

Imprecision is dealt with here in a procedural approach, consisting of progressively increasing the precision of initially imprecise information contained in a softgoal template. This is achieved through the application of a set of transformations that manipulate specialized formalisms defined to characterize the above discussed facets of quality requirements modeled as softgoals. The formalisms are necessary to assist stakeholders in representing and reasoning about quality requirements in a systematic manner. A softgoal formalization, based on the discussions above is as follows.

Formal Characterization of Softgoal.

We make explicit the facets of softgoals described above by modeling a softgoal $\mathcal{S}$ as a tuple:

$$\mathcal{S} = \langle n, t, St, v, c, P \rangle \tag{1}$$

where n is the softgoal identifier, t is the time of softgoal statement, St is the set of stakeholders that agree on the softgoal, v is the view of the softgoal shared by members of St, c is the context of the softgoal where $c \in software, process, environment$), and P is the preference information associated with the softgoal, including utility. The utilities are evaluated over a set of alternatives for softgoal operationalization (call this set B), each including a combination of the software, environment, and development process. The softgoals are then aggregated to produce the global utility, corresponding to the top softgoal of the project.

The preference information in a softgoal can be represented with a tuple P:

$$P : Obj \times Mod \times T \times Thr \times Curr \times U \tag{2}$$

where Obj is the objective function, Mod is the modality Min, Max of the objective function, T its target value, Thr its threshold value, $Curr$ the quality variable value of the existing alternative, U indicates whether the objective function can be considered as a local utility (see the classical utility theory [170]). The definition of the objective will often make use of auxiliary quality variables. They are defined by an expression, the metric function that (implicitly) depends on the alternative $b \in B$. An objective function is thus a metric function with an associated modality: $mod(m(b))$, where $mod \in Mod$. The modality indicates in which direction the metric function will influence the global utility.

The notation defined above allows the requirements engineer to compare alternatives by:

1. Defining an order among alternatives, as a first approximation.
2. State quality variables Qv to quantitatively compare alternative behaviors, as in [198] but not limited to random variables.

3. Defining metric functions $m(b)$ to associate alternatives b_i to metric values.
4. Combining metric functions $m(b)$ with modalities mod to construct objective functions $mod(m(b)) \in Obj$ which indicate preferred metric values T.
5. Refining metrics to local utilities, and aggregating them to obtain the global utility. Tradeoffs between degrees of satisfaction can be evaluated using the marginal rate of substitution (MRS), a concept taken from economics (e.g., [170]) which, in the terminology used here, indicates the maximal amount of a metric value that a stakeholder is willing to sacrifice for a unit increase in value of another metric. Techniques described in [330] can be reused here, although with caveats noted earlier.
6. Using linguistic facilities as in fuzzy logic: a value above threshold will be deemed *acceptable*, above target *good*.

The formalisms themselves do not eliminate imprecision, but point to information to look for and a way to organize it in order to reduce imprecision.

Imprecision can further be reduced by using logic to formalize the information about alternative behaviors contained in the softgoal. This allows closer integration with later steps of software development.

6.3.3 Formally Specifying Softgoals

To this point, the traditional softgoal concept has been enriched with templates that allow the expression of subjective, idealist, and context-specific facets of quality requirements information. Imprecision has been indicated, and treated with a simple formal model of the enriched softgoal concept. The model, while summarizing softgoal information in a precise way, does not alleviate imprecision. However, sources of imprecision have become clearer: the fuzzy set of behaviors and time-dependency of preferences which is derived from the fuzziness of the set of behaviors (i.e., preferences change because stakeholders learn about previously unknown behaviors during the development process). Both can receive further treatment: the former through formalization of behaviors, and the latter, through the transformation activities, presented in Sect. 4.

While behavior can be represented in various ways, the goal concept, discussed earlier, proves invaluable in the RE phase (e.g., [71, 198, 314]). It allows more freedom in the specifications, than, e.g., pre/post condition specification of state transitions used in [191, 203, 330]. As precise representation of behavior is needed, and since behavior represents *what* software or stakeholders do, functional goals (see, [315]) are used as a concept to model behavior. To remain general, the choice of formal acquisition language for functional goal specification is left to the requirements engineer. It is suggested that temporal logic be used for expressivity reasons. Softgoal formalization then consists of writing formal specification of behaviors $b \in B$ using the chosen acquisition language.

Return to the "software should not take too much hardware resources" softgoal in the previous subsection. Two behaviors appear in its choice preferences attribute. Using the KAOS framework (where a goal is defined as a constraint on behavior [71]), the two behaviors can be specified as KAOS goals (i.e., precise functional goals):

Goal: Maintain [SoftwRunsInBackgr]
Definition: The e-mail reader runs constantly.
Formal Def: $os : OperatSyst; mr : MailReader; os.status = on \Rightarrow mr.status = on$

Goal: Achieve [RunSoftwWhenRequest]
Definition: When the os receives a request to start the mail reader, the mail reader should start running.
Formal Def: $os.status = on \wedge mr.status = off \wedge os.start = mr \Rightarrow mr.status = on$

The above formalization is reflected in the softgoal template by adding a keyword becomes after the imprecise preference relation and rewriting that information using the identifiers for specified behaviors. The imprecise formulation is maintained for traceability reasons.

...
Preferences:
Choice Preferences:
(run software when requested)$\succ$(software runs in background)
becomes Achieve [RunSoftwWhenRequest] $\succ$ Maintain [SoftwRunsInBackgr]

In the NFR framework [225, 55] terminology, the above would be represented with a softgoal, two goals, and a contribution link between each of the goals and the softgoal. The preference relation can be translated into a positive and a negative contribution. However, NFR is less expressive, since metrics and most other facets of the softgoal concept presented above are missing.

6.4 General Guidelines for RE Frameworks

On the basis of the revisited conceptual foundations, guidelines can be suggested as to the transformations that can be applied to softgoals and that need to be present in requirements modeling approaches that aim to employ the given Softgoal conceptualization. Any such transformation activities need to be constructed so that they can deal with all of the four Softgoal facets identified above. Ideally, the transformations would allow initially imprecise, subjective, context-specific, and idealistic softgoals to be transformed into a consistent set of hardgoals (e.g., similar to those of the KAOS acquisition language [71]). We argue that two classes of transformation activities are useful—one for dealing with individual softgoals, and another for transforming several softgoals together.

Single-Softgoal Transformations

are aimed at arriving, for each softgoal, at a template in which the initially vague statement of need is made more precise, subjectivity is made explicit, objective metrics are found, and alternative behaviors influencing the degree of softgoal satisfaction are informally identified:

- *(T1) Build an initial softgoal template.* To discover softgoals, the requirements engineer will ask questions about *what* and *how* the software and the wider context should do, according to each stakeholder. The *what* and *how* questions are likely to result in informal and imprecise statements of needs that may be both related to behaviors (i.e., functional aspects) of the context, and how the behaviors need to be exhibited (i.e., nonfunctional aspects; e.g., rapidly, safely, securely, etc.). Consequently, the requirements engineer will need to fill in softgoal templates in a rather sketchy manner at first. As a result of T1, the template needs to contain information about the name of the softgoal (n), statement time (t), stakeholder identifier (St), stakeholder's view (v), and the context relevant to the softgoal (c).
- *(T2) Identify alternative behaviors that are likely to influence softgoal satisfaction.* The *what* and *how* questions will also lead stakeholders to indicate alternative behaviors whose execution will satisfy to a varying degree the softgoal. The set of identified alternative behaviors for a softgoal j (B_j) can be enlarged by looking at similar existing contexts (and observing, e.g., limitations, errors, etc.), seeking expert opinion on the specific problems, etc.
- *(T3) De-idealize the softgoal.* Stakeholders may express views which qualify or quantify behaviors in ideal ways (e.g., development cost should be lower than X—where X is simply impossible to achieve). De-idealization can be realized by further discussing alternative target values and/or behaviors, or by taking into account benchmarks, which would provide evidence on the idealistic nature of stated needs.
- *(T4-A) Construct objective measurements of softgoal satisfaction.* The aim is to find a set of quality variables (Qv), for the softgoal j. For each quality variable q_{jk}, there should be a metric function ($m_{jr}(b_i)$) to which a modal mod_{jr} is associated to form an objective function ($mod_{jr}(m_{jr}(b_i))$). A target value (t_{jr}) should be defined for the objective function. Quality variables can be derived from information contained in the stakeholders' view (as in the example in Sect. 3.2), in the parent softgoal (see transformation T7), and/or can be based on company-/industry-specific benchmarks. Metric functions can come from knowledge about the events generated by behaviors that are to be measured, from company-/industry-specific standards, and/or behavior categories (i.e., KAOS goal categories [198]). Benchmarks are an invaluable source of target values.
- *(T4-B) Establish preference relations over alternative behaviors.* Based on subjective indications of the stakeholder that has provided the information for softgoal j, alternative behaviors found by application of T2 can be related with preference relations. Preferences can also be established based on objective measurements, when, e.g., the current value for a metric is closer to the target value for a behavior over some other behavior. Notice that preference relations can be objectively constructed only when actual measurements exist (based, e.g., on similar systems) so that current values of quality variables under different behaviors can be observed.

The above transformations are likely to be given as a toolset to the requirements engineer. The order of application will probably be T1 to T4 initially, but iterations should not come as a surprise, especially when additional behaviors are suggested by the stakeholder or due to preference variability over time.

Many-Softgoal Transformations

are aimed at establishing relationships between two or more softgoals, to indicate inter-softgoal contribution and refinement. Contribution and refinement are based on widely accepted conceptualizations of

such relationships initially given in the NFR framework [225, 55], while relying here on a formal model for argumentation of contribution and refinement choices, which itself employs the formal softgoal model proposed above.

- *(T5) Negotiate to avoid conflict.* Contribution between softgoals indicates the degree to which a softgoal supports or obstructs the satisfaction of another softgoal. Contribution is interesting mainly when negative, or conflicting contribution exists between softgoals. Conflict between softgoals may appear in the form of inconsistencies resulting from different terminology (due to subjectivity and imprecision), conflicting preferences, and/or different target values of objective functions. Because of imprecision, the requirements engineer will not be able him/herself to resolve conflicts. Instead, negotiation can be used to lead stakeholders to common understanding, consensus, and closer terminology.
- *(T6) Argument modeling decisions.* Argumentation during negotiation can be recorded using a logical model of argument (for an overview of the research specific to logical models of argument, see [46]). Rigor in recording argumentation during the early phases is relevant not only for traceability reasons, but also because it confronts stakeholders to discuss quality requirements (allowing the requirements engineer to potentially find more information about preferences and alternative behaviors)..
- *(T7) Merge softgoals.* Negotiation will ideally lead stakeholders to a shared terminology and an agreement on quality requirements that have been initially perceived differently. Merging two or more softgoals consists of selecting a subset of preference information available in all softgoals to merge, while using a shared softgoal name, view, context, etc. Objective functions from merged softgoals can be aggregated, provided that quality variables they refer to be converted into compatible types.
- *(T8) Refine a softgoal.* A softgoal is refined if there are sub-softgoals whose joint partial satisfaction is considered equivalent to partially satisfying the refined softgoal. In practical terms, refinement can consist of, e.g., decomposing a softgoal according to some taxonomy (see, e.g., [10] for a privacy and [225] for an accuracy and a performance requirements taxonomies) into sub-softgoals, or making the softgoal more specific through each sub-softgoal. For example, "software should be fast" can be refined into a set of softgoals, e.g., "operation A should be fast",..., "operation Z should be fast".

The result of these transformations can be considered as completed when all of the following conditions hold: (i) a set of behaviors is associated with each leaf softgoal; (ii) there are no conflicting softgoals; (iii) all disagreements on softgoals have been resolved through negotiation; (iv) the set of softgoals is considered sufficiently complete by the stakeholders.

6.5 Conclusions and Future Work

The aim of the work presented in this paper is primarily a more profound understanding of the Softgoal concept that is commonly used to model requirements in the early stages of requirements engineering. It has been argued that there is more to the information commonly represented using Softgoal instances, than currently established Softgoal definitions seem to indicate. In particular, four facets of the Softgoal concept are identified—namely: imprecision, subjectivity, context-specificity, and idealism (which involves implicit preference orderings of the stakeholders who state the information represented using Softgoal instances). A tentative formalism for this extended Softgoal conceptualization is suggested, to summarize the information that we argue the requirements engineer can and should attempt to extract from a Softgoal instance (or, in relation to it). It is also illustrated how richer requirements can be obtained when the extended conceptualization is taken into account.

Although a rather simple example has been employed to illustrate the facets we consider relevant, we believe that a powerful insight comes from this paper: a more elaborate treatment of imprecise, subjective, context-specific, and idealistic requirements, usual at the outset of a RE project, can be realized if a commonly used Softgoal concept is extended. Ultimately, this is likely to lead to richer requirements specifications and more stakeholders who are satisfied with the performance of the systems built for them.

Important directions for future work include extending the Softgoal concept further, by possibly identifying additional facets. The formalism needs to be operationalized within already common specification languages. Additional transformation techniques, more effectively exploiting the extended conceptualization remain to be explored.

Part II

Techniques for Clarification, Argumentation, and Justification

7

Outline of Part II

The revised formulation of the requirements problem – introduced in Chapter 4 – acknowledges the importance of defeasible reasoning in solving the requirements problem. This leads – in Part II – to the study of how defeasible reasoning can be incorporated into established decision making processes involved in the identification and analysis of requirements.

Chapter 8 introduces the argumentation, justification, and clarification techniques as means for assisting the engineer in proceeding – through defeasible reasoning that underlies argumentation – to the construction and revision of models that represent the purpose of the IS. In light of the revised requirements problem in Chapter 4, the argumentation and justification techniques are means for arguing for and justifying the acceptance or rejection of models of IS purpose. These techniques thereby help the engineer in deciding the appropriateness of models and their systematic rejection or revision if this proves necessary. Rigorous argumentation and justification are suggested by drawing on artificial intelligence research. Clarification is introduced by drawing on linguistics, in the aim of detecting unclear – and thus unacceptable – arguments and their subsequent clarification. In particular, Chapter 8 organizes argumentation, justification, and clarification within a method – called Goal Argumentation Method – in which these techniques are organized around a systematic approach to decision making during the construction and revision of goal models.

Chapter 8 thus fits in the conceptual framework established in Part I by adding actual means for guiding the reasoning during the use of the core ontology for the modeling of the purpose of the IS. By relying on argumentation and justification, the Goal Argumentation Method is in line with the need for defeasible reasoning in RE, as acknowledged in Part I of the thesis.

8

Clear Justification of Modeling Decisions for Goal-Oriented Requirements Engineering

Abstract. Representation and reasoning about goals of an information system unavoidably involve the transformation of unclear stakeholder requirements into an instance of a goal model. If the requirements engineer does not justify why one clear form of requirements is chosen over others, the subsequent modeling decisions cannot be justified either. If arguments for clarification and modeling decisions are instead explicit, justifiably appropriate instances of goal models can be constructed and additional analyses applied to discover richer sets of requirements. The paper proposes the "Goal Argumentation Method (GAM)" to fulfil three roles: (i) GAM guides argumentation and justification of modeling choices during the construction or critique of goal model instances; (ii) it enables the detection of deficient argumentation within goal model instances; and (iii) it provides practical techniques for the engineer to ensure that requirements appearing both in arguments and in model instance elements is clear.

8.1 Introduction

Requirements engineering (RE) is a structured approach to the assessment of the role that a future information system (IS) is to have within a relatively well-delimited human and/or automated environment. It involves the identification of goals to be achieved by the IS, their operationalization into implementable IS services and constraints, the identification of resources required to perform those services and the assignment of responsibilities for the resulting requirements to agents, such as humans, devices, and software.

A usual starting point in RE is the elicitation of goals that the future IS will need to achieve once developed and deployed [315]. Goal modeling can be defined as the activity of representing and reasoning about IS goals using models, in which goals are related through relationships with other goals and/or other model elements, such as, e.g., actions that system agents are expected to execute, resources that they can use, or roles that they can occupy. With a number of currently established RE methods relying on goal models in the early stages of requirements analysis (e.g., [50, 55, 71, 76, 202]; see, [315] for overviews), there seems to be a consensus that goal models are useful in RE.

When an instance of a goal model (henceforth, "goal diagram") is constructed by few stakeholders having similar backgrounds, during a very limited amount of time, and consequently for a relatively simple system, there is generally no need to record the details of the decision process that has led to the final goal diagram. However, as the system under scrutiny gains in complexity—which tends to occur for most but toy systems, and is certainly true for IS employed in automating, e.g., government and healthcare services, air-traffic management, industrial production processes—the inherent inability of individual human stakeholders to grasp the full extent of interactions and interdependencies between system components, to predict with detail and/or certainty the future conditions in which the system is expected to operate and the influence of such conditions on its functioning, makes the construction of the goal diagram an intricate task, critical for the success of the subsequent IS development activities. Moreover, stakeholders' preferences are rarely absolute, relevant, stable, consistent, precise, or exogenous (e.g., [210] and later), thus making it difficult to choose among alternative requirements.

A prominent consequence of system complexity, environment unpredictability, and preference variability is that the requirements provided by the stakeholders will be, among others, ambiguous, overgeneral, and vague (overall: "unclear"), thus unavoidably leading the requirements engineer to transform such information into a clear form written in a goal diagram. This transformation consists essentially of the engineer *interpreting* the information provided by a stakeholder, relating the interpretation to the se-

mantics of the goal model, and finally, establishing how to represent it using the syntax of the model. In doing so, the engineer encounters two significant issues:

- With stakeholders' tendency to express expectations in natural language prone to misinterpretation, it is difficult for the engineer to be assured that her understanding corresponds to what the stakeholders intended to communicate.
- Although it is reasonable to assume that problem and solution knowledge relevant for the engineering of the future IS rests mostly with the stakeholders, system complexity and environment unpredictability make it necessary for the engineer to take an active role in shaping requirements, namely by questioning the rationale behind the expectations expressed by the stakeholders.

Failing to address the above in a structured manner has apparent consequences on the success of the RE phase in system development:

1. Traceability between expectations and their representation in a goal diagram is undoubtedly weak: why a particular natural language requirement is represented in a particular way and not another remains unknown. It is to expect then that stakeholders reviewing the diagram will not know why another stakeholder or the requirements engineer made the given modeling decision. The result may be an unnecessary review of the diagram, changes, or additional explaining. These activities require time and resources that could have been employed in a more productive manner.
2. A stakeholder cannot recall the reasons for making a modeling decision. While goal modeling is an iterative process, it is not uncommon to review the prior decisions because of imperfect recall of reasons leading to them in the first place. Future iterations could be better informed if rationale for prior ones is explicit.
3. The ideas, arguments, and assumptions underlying a decision remain implicit and/or are lost over time. Alternative ideas and confronting views that could have led to different, possibily more adequate modeling choices are lost as well. Both favor a poor understanding of the problem and solution domains. Empirical data suggest that this is an important cause of RE project failure [67].
4. There is no guarantee whatsoever that expectations are not misinterpreted by the engineer, as intuitive detection of ambiguities, vagueness, and so on in stakeholders' statements by itself gives no serious grounds for claiming proper understanding. If there are no checks for clarity of requirements statements, it is difficult for a stakeholder or the requirements engineer to establish whether a statement is unclear and how it can be clarified. As illustrated in the remainder, some such clarity checks are obvious, but many are not at all trivial. Leaving the clarity checking implicit is likely to lead to the application of trivial and intuitive checks, while non-trivial ones will be disregarded for simplicity.
5. Inconsistencies that could have been identified by clarifying initial requirements in a structured manner would remain hidden until later steps of the RE process. Inconsistencies identified early on help in eliciting additional requirements that would otherwise have been missed.

8.1.1 Contributions

One possible approach to reducing the issues 1–5 is to (i) externalize and document arguments that led the stakeholders to express some requirements as well as the arguments that led the requirements engineer to transform these requirements into goal diagram content; and, (ii) to analyze the clarity of these arguments and of the information given within the goal diagram, revising it if proves necessary. To facilitate tasks (i) and (ii), the present paper proposes the so-called "Goal Argumentation Method (GAM)", which integrates a decision process, an argumentation model, and techniques combined to enable the analysis of argument structure, justification of modeling choices, and clarification of information appearing in arguments and elsewhere in the the goal modeling decision process. Drawing on design rationale literature (see, [205] for an overview), the decision process suggests an intuitively acceptable organization of the goal modeling task. The argumentation model, inspired by work in artificial intelligence argument models (see, [46] for an overview), is introduced in the evaluative step of the decision process, allowing various degrees of structure and rigor in the provision of arguments for modeling choices. The analysis techniques serve for the justification of modeling choices, the study of argument interaction (e.g., defeat and counterargumentation), the detection of deficient argumentation, and the checking for unclear information and subsequent clarification.

An important characteristic of GAM is that it does not integrate a particular goal model; instead, it is independent of specific goal model syntax and semantics, allowing its combined use with any available RE framework which employs goal models. In conjunction with any available goal-oriented RE framework, GAM fulfils three roles:

1. GAM guides argumentation and justification of modeling choices during the construction or revision/critique of IS goal diagrams.
2. It enables the detection of deficient argumentation within available goal diagrams (which need not have been built using GAM).
3. It provides practical techniques for the requirements engineer to ensure (to a reasonable extent) that information appearing both in arguments and in diagram elements is clear (i.e., is not ambiguous, overgeneral, vague, among others).

The principal contributions of this paper to the RE field are the introduction of generic argumentation and clarification conceptualizations and techniques in goal modeling for RE. Hopefully, the present paper highlights that the activities of argumentation and clarification are method-independent concerns and therefore significant for RE at large. Following the initial presentation of GAM at the 14th International Requirements Engineering Conference (RE'06) [150], the present paper extends the method considerably, to include clarification of arguments and stakeholders' statements of requirements.

8.1.2 Case Studies

While the reader may assume from the above that GAM addresses issues arising only when the complexity of the future system surpasses some considerable threshold, it turns out that such a threshold is, in actual application, unexpectedly low. Having exemplified GAM initially [150] using the classical case study involving the engineering of requirements for a meeting scheduler system [313], GAM was shown to apply to a wide range of settings, including systems that are less complex than many to which GAM has initially been oriented. To ensure that the text is readable and that the main features of the method are salient, the same case study is maintained herein. Since the presentation of preliminary results at the RE'06 conference, GAM was applied to a realistic case study of considerable complexity, involving the clarification and justification of requirements for an air-traffic management (ATM) IS. Requirements are based on extensive documentation providing records of numerous meetings of the stakeholders involved in the European Organisation for the Safety of Air Navigation [83]. Part of the results obtained in the ATM case study are employed herein to illustrate the applicability of some specific features of GAM to the engineering of requirements for complex systems.

In the present paper GAM is used with the Tropos RE method [50, 100]. Tropos' goal model is based on the well-known i^* modeling framework (see, [331] and related). The i^* notation features a simple yet expressive notation, which is briefly overviewed below (§8.2). An advantage of adopting Tropos is that the modeling primitives of i^* are usually present in similar form within most goal-oriented RE frameworks (see, e.g., [315]), thus maintaining the discussion generic.

8.1.3 Organization

Problems of justification and clarification are first exemplified within the two case studies (§8.2). Examples are then used to illustrate the features and use of GAM, that is, the decision process (§8.4), the clarification techniques (§8.5), and the argumentation model and associated analysis techniques (§8.6). After reviewing related work (§8.7) and discussing limitations of GAM (§8.8), conclusions are drawn and directions for future work identified (§8.9). Each section presenting parts of the method provides and discusses a set of definitions of concepts, techniques employing these concepts, and examples illustrating the use of techniques in the case studies. The suggested techniques are derived from our experience in using GAM and the literature associated to concepts employed in GAM but introduced in related research.

8.2 Illustration of the Problem

Consider a system for scheduling meetings, similar to that described in [313, 332]. The meeting scheduler should try to select a convenient date and location, such that most potential participants participate effectively. Each meeting participant should provide acceptable and unacceptable meeting dates based on his/her agenda. The scheduler will suggest a meeting date that falls in as many sets of acceptable dates as possible, and is not in unacceptable date sets. The potential participants will agree on a meeting date once an acceptable date is suggested by the scheduler.

A goal diagram for such a system would be represented in Tropos as an instance of the i^* Strategic Rationale (SR) model. The i^* framework comprises, in addition to the SR model, the so-called "Strategic Dependency (SD)" model, which features a subset of the modeling primitives of the SR—the latter is

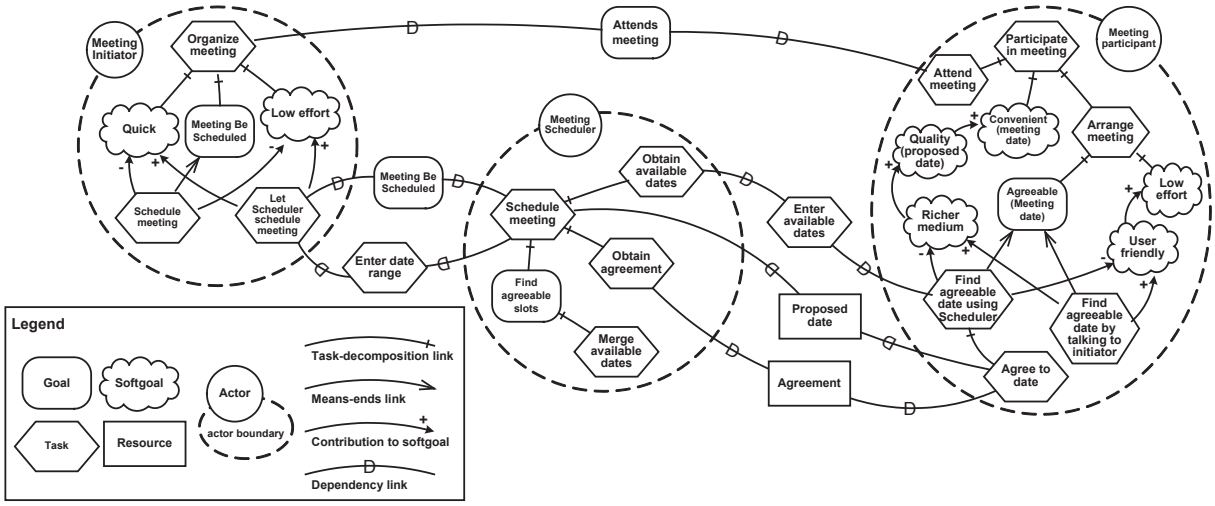


Fig. 2. An i^* Strategic Rationale diagram from [332].

therefore taken as the reference Tropos model in the remainder. An example SR diagram for the scheduler, taken as-is from [332], is reprinted in Fig.2. It shows actors such as *Meeting Scheduler* and *Meeting Participant*, their interdependencies in the achievement of goals, the execution of tasks, and the use of resources, and their internal rationale when participating in the given IS. For example, the *Meeting Be Scheduled* goal of the *Meeting Initiator* can be achieved (represented via a means-ends link) by scheduling meetings in a certain way, consisting of (represented via task-decomposition links): obtaining availability dates from participants, finding a suitable date (and time) slot, proposing a meeting date, and obtaining agreement from the participants. Cloud-shaped elements designate softgoals which differ from goals in that there are no crisp criteria for their satisfaction. Softgoals are commonly used to represent nonfunctional requirements in a goal diagram.

During the RE process, *the appropriateness of this goal diagram can be evaluated on the basis of arguments that support its content*. The appropriateness is defined here as the likelihood of the IS produced from the given diagram to satisfy stakeholder needs. If arguments are missing, alternative diagrams that could be produced by the stakeholders or requirements engineers for the same IS could be considered appropriate, provided that no errors are made when using the syntax and semantics of the relevant goal model. While the SR in Fig.2 serves as a valuable example to illustrate the syntax and semantics of the i^* framework in [332], it is difficult to accept without justification that diagram as more appropriate than another one in a RE project. Lack of arguments to support the diagram in Fig.2 lead a stakeholder to challenge it by pointing, e.g., to unnecessary extensiveness or to incompleteness, as in the following questions:

- How does the initiator inform participants that a meeting is being organized?
- Would it not be user friendly for the meeting initiator to inform participants about the meeting using the meeting scheduler?
- Would it not be user friendly if the scheduler looked available dates up in participants' electronic agendas?
- Does the scheduler remind participants of the meeting date? If yes, how/when does it do so? If no, why not?

If arguments were given explicitly for the diagram in Fig.2, stakeholders might know that, e.g., the initiator prefers to inform participants verbally, that different formats of electronic agendas make it costly to develop a scheduler that can communicate with each participant's software, and so on. Even if such questions are not asked, making it unnecessary for the requirements engineer to address them, ideas and assumptions that could have surfaced and led to additional requirements as a result of the questions would remain hidden. It would be easier to consider the goal diagram in Fig.2 appropriate if modeling decisions leading to it are justified.

Adding arguments alone certainly seems helpful in answering questions such as above, though it is a different issue altogether to ensure answers given in arguments are meaningful, and this in the same (or, at least very similar) way to all relevant stakeholders—it would otherwise be particularly difficult to agree on the relevance and acceptability of arguments given to support modeling decisions; in practice, this translates in extensive counterargumentation due primarily to lack of clarity. The same applies to

information already placed within the goal diagram—consider, e.g., the nonfunctional requirement *Quality of the proposed date* represented as a softgoal in Fig.2. The stakeholder could ask the following questions:

- What does "quality" of a meeting date mean?
- Who decides when the quality of the date is high/low?
- Under what conditions is the quality of the proposed date considered high or low?
- What if my criteria for meeting date quality are different from those assumed in the diagram?

The questions point to stakeholders' unsatisfactory understanding of the goal diagram, due to a lack of *clarity* of the information in the diagram. For a more elaborate illustration of clarity issues in statements of requirements, consider an actual requirements document [83] which is a result of consultations that took place from 1994 to 1998 of a number of stakeholders selected by and involved in the European Organization for the Safety of Air Navigation. The document compiles information about the needs the given parties expressed regarding the air traffic management strategy and information systems that would support it for the period after the year 2000. The following is an example of a requirement from the cited document (more such examples are given in §5):

> "For short haul and regional operations there is a need for ATM [i.e., Air Traffic Management] to make them wait on the ground rather than in the air in case of weather contingencies and to be treated fairly with respect to arriving long haul aircraft in the air while they have not taken off yet. Once given the takeoff clearance they should be able to fly directly to the destination." [83] (p.18)

A number of questions arise, some answerable through a specialized glossary (e.g., What are the conditions for a flight to qualify as short haul?) while other are more intricate (e.g., Under what conditions can be said that there is a weather contingency? What is a "fair" treatment? When can the fair treatment be overrun by human operators?). Provided that the system involved in satisfying the above requirement is partly automated and as argued earlier, the latter questions involve interpretation by a requirements engineer involved in the specification of the automated parts of such a system. There is seemingly no pressing issue here if the requirements engineer is also an expert in air traffic management, for this person would already know the precise answers to these questions and would know how to appropriately specify them in a requirements diagram. However, while air traffic is sufficiently safety-critical for society to warrant the existence of bodies that specialize in managing requirements acquisition, analysis, maintenance for ATM systems, in most situations the requirements engineering expertise lies not with the same people as expertise in the problem that the engineered system is expected to resolve. Empirically, the said separation of effort is observed to weigh on the success of RE projects [67]. Separation entails that the requirements engineer ought to take particular care when interpreting requirments stated by the stakeholders who are likely to be more knowledgable in the specific problem at hand. As observed in §8.7, it is surprisingly rare for available RE frameworks to suggest ways to raise the awareness of the requirements engineer to unclear information and the problems of interpretation of requirements. GAM addresses this issue through the concept of unclear information, introduced and discussed in §8.5.

The preceding discussion highlights essentially two issues:

1. It is desirable to argument and justify modeling decisions, as the notion of appropriateness of a goal diagram would lose all meaning—i.e., any diagram would be appropriate provided that no technical errors are made in modeling.
2. Information used and produced during RE can be unclear, thus increasing the probability that it will be misinterpreted by the requirements engineer and that the resulting diagram will be inappropriate.

The position adopted in the remainder is that the two problems are related: the arguments leading to specific modeling choices can be unclear *and* the information contained in a goal diagram can be unclear—both therefore require clarification. Hence the analysis of information clarity (§8.5). But as clarification of a piece of information can result in choosing among alternative clear forms of the given information (in case it is, e.g., ambiguous), and the apparent interest in arguing for some choices over other, understanding how to proceed to argumentation is of relevance as well.

Although it may be interesting to further claim that the clarity problem may arise in part from restrictive (or, e.g., ambiguous and incomplete) syntax and semantics of a goal model that is being employed, the present discussion does not go so far: the existing literature is followed in assuming the value of the chosen Tropos goal model for RE activities (e.g., [50, 100, 332]). Additional information, aimed at clarifying and arguing for modeling choices, is therefore given *with* (instead of *in*) diagrams to avoid changing the original model or the associated, wider RE framework.

A useful complement to a goal diagram contains arguments that justify or challenge decisions for clarifying and then modeling requirements in a particular way. Although argumentation and clarification

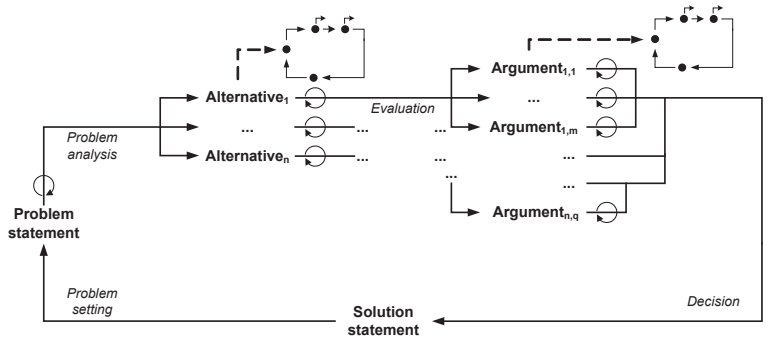


Fig. 3. Outline of the GAM decision process.

can be informal and without a particular structure, the following benefits can be gained by using a structured method:

- Of relevance to traceability, the use of a decision process explicitly adapted for argumentation allows each element introduced in a goal diagram to be related to a set of arguments that justify or criticize the modeling decision leading to the given representation of that element.
- When new information becomes available, the change of the diagram that it may require can be easier to understand if arguments for prior decisions are explicit. Otherwise, the engineer may not be capable of understanding the relationship between the new elements added to the diagram and the existing ones, leading the engineer to overlook additional changes for ensuring the consistency of information in the diagram.
- If arguments are explicit and structured according to a few simple rules §8.6, the justification process can be analyzed for inconsistency and preference over arguments can be established. If arguments are further formalized, the justification process for a goal diagram can be at least partly automated.
- In addition to qualifying some information as unclear, it is possible to determine the kind of information that clarifies it. Namely, it will be illustrated that information can be unclear in many *different* ways, so that no single clarification technique always applies. Accordingly, it is useful for the requirements engineer to determine in which way the information is unclear so as to apply specialized techniques.

8.3 Brief Overview of GAM

To address the problem outlined in §8.2, GAM combines three components:

- A **decision process** (§8.4) which highlights the key steps to take when creating a new or modifying an existing goal diagram, and gives an overall organization to the argumentation and clarification activities in relation to goal modeling.
- A set of **clarification techniques** (§8.5) to check the information used and produced during decision making for clarity, and clarify it in case it is judged unclear.
- An **argumentation model** (§8.6) which imposes restrictions on how pieces of information are composed into arguments, and how the latter combine to arrive at a justified decision.

While the decision process and the clarification techniques are essentially informal, the argumentation model is suggested in both an informal and a formal variant. The formalisms facilitate and make precise the presentation of the definitions of argument and dialectical trees. It should be noted that the formalization of arguments is not required for the remarks and suggestions made below to apply; however, avoiding a structured language (be it first-order or a propositional one) is not helpful when automated translation between goal diagrams and arguments is attempted (see, §8.6.1). The three following sections present and discuss respectively the decision process, the clarification techniques, and the argumentation model.

8.4 GAM Decision Process

GAM employs a decision making process to organize the activities involved in clarification and argumentation, and whose outcome is a change in the content of a goal diagram. Although systematic decision

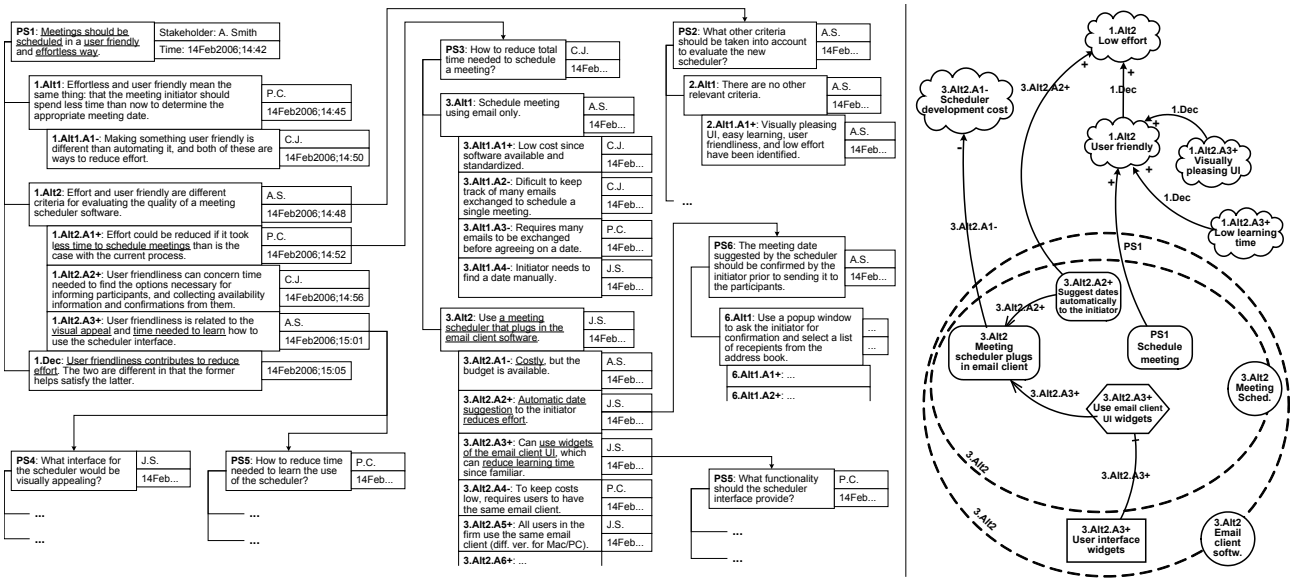


Fig. 4. Using the GAM decision process to build an i^* diagram for the meeting scheduler.

procedures are generally adapted for specific application domains, most involve a number of common steps shown in Fig.3. A decision maker generally proceeds by setting the problem to be resolved by application of the process. The problem statement provides initial directions for the identification of alternative solutions, which are in turn evaluated, by making qualitative or quantitative arguments to support or defeat each alternative. The chosen alternative takes the form of a solution statement. A new problem can then be tackled and the decision process repeated until the stakeholders involved in decision making consider that the relevant issues have been addressed.

The GAM decision process, given in Fig. 3 is purposefully generic, for its principal purpose is to highlight the steps at which the clarification and argumentation activities are to be executed within *any* decision process suggested in the engineer's RE framework of choice. The activities and deliverables appearing in Fig.3 are based on recomendations of design rationale research (e.g., [64, 286, 205]), which is concerned with assisting humans when reasoning about the rationale behind decisions that lead to the production of an artifact. A design rationale expresses elements of the reasoning which have been invested behind the design of the artifact [286]. The various design rationale approaches that have been suggested in the software engineering literature give a set of concepts and suggest ways in which these can be manipulated during a design activity (for an overview, see [205]). For example, the IBIS [64] approach consists of relating *issues* that need to be deliberated to *positions* that resolve issues, and *arguments*, that support or object to positions. More recently, [205] suggested the *reasoning loop model* (RLM), which integrates common characteristics of established design rationale approaches. It starts from a description of a problem which generates *goals* that characterize potential solutions. Then, *hypotheses* about potential solutions that satisfy goals are generated through problem analysis. Evaluation of alternative hypotheses leads to a *justification* of a selected alternative, which in turn leads to *deciding an action*. The result of an action is likely to lead to new goals, thus restarting the reasoning loop. It is not difficult to establish immediate links between the concepts given in Fig. 3 and the ones common in design rationale research:

- The *problem statement* designates any objective to be reached, demand to be satisfied, problem to be solved, issue to be discussed, in general anything one would like to achieve through problem resolution. It corresponds to the concepts of *issue* (in IBIS [64]), *goal* (RLM [205]), *requirement* (REMAP [266]), *decision problem* (DRL [192]), *question* or *criterion* (QOC [208]).
- The *alternatives* are potential solutions to the stated problem, and correspond to *positions* (IBIS, REMAP), *hypotheses* (RLM), *alternatives* (DRL), and *options* (QOC).
- The *argument* is a piece of information (e.g., a statement) that either provides support for, or is provided against choosing an *alternative*. It is conceptually close to *argument* (IBIS, REMAP, QOC), *justification* (RLM), and *claim* (DRL).
- The *solution statement* encompasses both the alternative chosen as the result of evaluation and the action taken to conform to the prescription given in the chosen alternative. As such, it is similar to *decision* (REMAP) and *design action* (RLM).

Technique 1. (Applying the GAM decision process) The decision process is applied as follows. Starting from an empty or already elaborate goal and/or decision diagram (the latter is an output of the decision process), stakeholders proceed by setting the problem, which consists of finding gaps between the desired and the observed in both diagrams. To describe the result of the problem setting activity, a problem statement is devised in natural language by stakeholders who consider the given diagrams' content inadequate. The *problem statement* should not be mistaken for a goal of the IS for which the requirements are being engineered: a problem statement may result in adding an IS goal to a goal diagram, but it may also lead to adding any other modeling element or changing any of the two diagrams in some other way. Stakeholders then suggest ideas and make proposals about the resolution of the stated problem. They generate a set of *alternatives* (shown as $Alternative_1$,..., $Alternative_n$ in Fig. 3). Alternatives are evaluated by providing one or more *arguments* to support or contest each alternative. When only one alternative remains justified and all others defeated, a decision is reached, and the diagram is changed according to the information contained in the justified alternative. The completion of a cycle in the reasoning process leads to the initiation of a new loop, until stakeholders agree that no further reasoning about a diagram is required. A loop must be closed, i.e., all activities executed—otherwise, the stated problem is not considered to be treated adequately.

Technique 2. (Initiating additional decison processes) The dashed arrows in Fig.3 indicate that additional decision process loops may be initiated from some of the specific decision deliverables. It is possible to identify new problem statements from alternatives (e.g., an alternative may highlight, be it is selected or not, the need for existing diagram elements to change in a way not anticipated in the problem statement) and arguments (e.g., some of the arguments provided for alternatives may be based on information already in the goal diagram, so that additional arguments that contest the former may point to problems in the goal diagram). There are no dashed arrows from the decision and problem statements, as any issue identified as a result of the decision statement generates a problem statement, which itself initiates a new instance of the decision process.

Technique 3. (Unstructured clarification in the decision process) The small circular arrows in Fig.3 indicate that there may be a need for clarification of information produced in the activities of the decision process. Information that requires clarification is written in a problem statement of a new reasoning loop, in which alternatives can be, e.g., different clear forms of the unclear statement. Until §8.5, clarification is left to the intuition of the engineer and the participating stakeholders.

Example 8.1. (Applying the GAM decision process) To illustrate the application of the decision process in GAM, part of an *i** SR diagram for the meeting scheduler is constructed starting from an empty diagram. The output of the decision process, i.e., the *decision diagram*, is on the left-hand side of Fig.4. Next to it is an incomplete SR derived from part of the information contained in the decision diagram. To relate SR diagram elements to those of the decision diagram, each SR element is annotated with the reference of the reasoning diagram element from which it is derived, and the expression from the decision diagram that is translated into the SR diagram is underlined. For example, the goal *Schedule Meeting* is marked with *PS1* indicating the decision element (here, the problem statement at the root of the reasoning diagram) that led to the introduction of the goal in the goal diagram. As a convention, arguments for an alternative are marked with *Ax*+/-, where *x* is the number of the argument in the list, and + (plus) and - (minus) symbols are used, respectively to indicate that the argument supports or contests the alternative. Clarification is the first activity that has been realized: the *PS1* problem statement has been considered unclear, in that the meanings of user friendly and effortless, as well as the potential relationship between the two are unclear. Two alternatives were suggested and a decision has been taken to adopt the second alternative (*1.Dec*) based on arguments given for the alternative. Some of the alternatives led to additional problem statements that were in turn subjected to the decision process. Each element of the decision diagram is labeled with the name of the stakeholder that suggested it, and the time of writing. Several additional observations can be made:

- There is information in the decision diagram that is not in the goal diagram (e.g., the reasoning behind the construction of the goal diagram), and there is information in the goal diagram that is not in the decision diagram (e.g., that some expression in an argument is interpreted as a task or a goal). The two diagrams are complementary, allowing a stakeholder to discover why a goal diagram has been constructed in a particular way by reading it in conjunction with the decision diagram.
- Because the construction of the (incomplete) diagrams in Fig.4 started from an empty sheet, the information in both of them is still unclear. For example, one could ask if the task marked *3.Alt2.A3+* is a decomposition of another goal diagram element, and if so, which one; or if the goal *Schedule*

Meeting could be made more precise and how, etc. Clarification will lead to additional decision loops (as was initially the case for *PS1*). As illustrated above, the result can be increasingly precise decision and goal diagrams.

- GAM can be used to document the decision making behind the use of specific goal analysis techniques, already established in RE (such as, e.g., goal refinement and operationalization [315]). For example, the modeler can document reasons leading to, e.g., a particular refinement or decomposition of the *Schedule Meeting* goal in Fig.4. This way, the decision process in GAM helps to fill a gap between abstract suggestions on how the goal modeling activity should be organized (e.g., elicit goals from available documentation and look further by asking why and how questions [315]) and very precise, formal techniques already available in RE methodologies (such as formal refinement, abstraction, operationalization [197, 315]), without requiring them to change to fit the design rationale approach.

The reader may question whether the GAM decision process should further incorporate concepts that tend to commonly appear in RE decision making: for instance, the REMAP [266] decision rationale approach features additional concepts such as "constraint" and "design object", as well as a number of specialized links to relate the different concepts, including, e.g., "generalizes", "specializes", "responds to", etc. It would then appear interesting to introduce the RE-specific concepts such as, "goal", "task", "actor" within the decision process model. It has, however, been observed that particularly close integration of design rationale with the specific artifacts (here, goal models) may result in the loss of the original vision of argumentative decision making [286]. Since GAM is intended to assist more extensive RE methods that already rely on specific techniques to organize the RE process as well as specific ontologies, it was important not to use an approach that requires adaptation of the established methods. Therefore, the direction of design rationale approaches that aim to be minimally prescriptive, lightweight, informal, non-intrusive on the design activity they complement, and place no restrictions on the artifact being produced are favored, so as to facilitate the practical applicability of GAM. It can also be observed that the concepts appearing in the GAM decision process purposefully leave much to interpretation, leaving emphasis on argumentation and clarification in choosing among alternative solutions. GAM is in this respect an argumentative approach to goal modeling: by involving and integrating argument and argumentation of alternatives as, respectively, a key concept and activity, the proposal follows the noted relevance of argumentation in dealing with problems that lack a precise, agreed-upon formulation or available plans of action (e.g., [273, 64]).

The construction of the example given in Fig. 4 uses neither the specific clarification techniques nor the argumentation model available in GAM. Arguments are given without verifying their clarity and without a particular structure. Moreover, no precise criteria have been applied to determine whether arguments are inconsistent and if some arguments defeat others. Clarification remained informal, grounded in the intuition of the engineer and the stakeholders while structured around the decision process, in which the unclear information is considered the problem, and alternatives its various clear forms. Clarification techniques are proposed in the following section to avoid ambiguity, overgenerality, synonymy, and vagueness of arguments and of information in the goal diagram.

8.5 Clarification

Even in safety-critical systems such as those used in air-traffic management, unclear information abounds, as the example below illustrates.

Example 8.2. (Unclear information in ATM documentation) Consider the following excerpt on stakeholders expectations about the ATM system [83]:

> "All users want total visibility on costs and charging (cost recovery) mechanisms regarding ATM system development and operation expenditures. Constant monitoring of ATM services costs and quality of service ('benchmarking') delivered to the client is needed to ensure the cost effective provision of such services." [83] (p.12)

The above seems ambiguous (does "total visibility" involve control of expenditures in additon to their transparency?), overgeneral (for it refers in a much too general way to the entities it concerns: e.g., Who are the "users"? What are the "cost and charging mechanisms"?), carries synonyms (is there a difference between "cost" and "expenditure"?), and vague (since "cost effective" relies on a gradeable adjective, admits borderline cases of application, and the Sorites paradox—see §8.5.5). Similar issues are found in the excerpt below:

> "...also applies to pre-tactical (day minus one) and tactical (same day) flight planning phases." [83] (p.17)

Where "day minus one" admits alternative and contraditory interpretations: day minus one can be measured in different ways, while "day" can mean 24h or a working day, thus requiring disambiguation.

> "...the flexibility of ATM to cope with unforeseen short term changes in demand or partial failures whilst ensuring that the repercussions for all airspace users remain acceptable." [83] (p.20)

Above, "acceptable" is vague, as is "excessive communication" below:

> "...eliminating excessive routine voice communications should apply to all flight phases in the ground/ground and air/ground communications... The pilot wants as few frequency changes as possible..." [83] (p.17)

Stakeholders may have a number of reasons to communicate statements of requirements or give arguments that are unclear in one way or another: their knowledge of problem and solution domains may be restricted; they learn as the project unfolds, thus establishing and/or changing their preferences over alternative (parts of) problem and solution statements; for ease, they are likely to avoid a rigorous representation language, expressing their needs in natural language. It can then be reasonably assumed that the production of a specification obliges the requirements engineer to clarify the stakeholders' statements either in a passive or an active way. A passive approach would be to consider a statement sufficiently clear if there are no stakeholders that explicitly question it. This may be unhelpful because: (i) a stakeholder may not question a statement since its understanding is based on implicit domain knowledge shared with other stakeholders; (ii) there may be few motives/rewards for a stakeholder to participate actively in the RE process; (iii) a stakeholder may consider either being incompetent to question the statement, or unaffected by the part of the system that the statement concerns. In case of (i), domain knowledge of the engineer is likely to remain limited, while important requirements are also likely to be overlooked if (ii) and/or (iii). An active approach, in which the requirements engineer questions stakeholders' statements openly by using specialized clarification techniques therefore appears desirable.

Unclear information is problematic because the engineer has difficulties in giving it a unique interpretation and consequently translating it into elements of a goal diagram or using it as useful arguments in justifying modeling decisions. Although the engineer may perceive some information as unclear, to act in order to clarify it, the engineer ought to know how to detect a lack of clarity and to identify directions for the enchancement of unclear information. Moreover, as ambiguity differs from vagueness, synonymy differs from each of the latter, and so on, there can be various distinct techniques for detecting lack of clarity and subsequent clarification: i.e., clarity is a multi-facetted construct. In practical terms, in addition to perceiving a piece of information as unclear, the method must enable the engineer to determine along which facets it is unclear, and clarify accordingly.

From there on, an active clarification process is conceptualized as a successive application of a set of basic *clarification techniques*, each being a transformation of information initially considered unclear into that perceived as clear by the stakeholder(s). The aim of the requirements engineer is to move on each dimension towards a direction assumed desirable: e.g., moving from "more" to "less" ambiguity, from more to less vagueness, etc. In addition to clarification techniques, clarity checking techniques are required to detect if some information is unclear along a particular facet. It follows that a way of helping the engineer in an active approach is to provide a rich catalog of clarity facets, to define each facet for easier identification, and to suggest clarity checking and clarification techniques to be applied when a facet is identified.

The catalog of four facets—ambiguity (§8.5.2), overgenerality (§8.5.3), synonymy (§8.5.4), and vagueness(§8.5.5)—introduced herein is not meant to be complete and its extension is encouraged: it is impossible, knowing the extent of the literature on linguistic phenomena such as ambiguity, to provide a full account herein. Practicality has therefore been the focus, with discussion and careful reuse of established results in linguistics (e.g., [105, 268, 173]), philosophy (e.g., [328, 298, 15, 110]), and artificial intelligence (e.g., [23, 116]). The proposed facet classification, along with the clarity checking and clarification techniques make no attempt at settling debates on the essence of concepts such as vagueness or ambiguity. Instead, the proposal draws on various literatures, taking as given some of the existing philosophical and AI results while introducing techniques specialized for the problem at hand.

8.5.1 Introduction to Clarity Checking and Clarification

As argued above, the aim of the requirements engineer is to know whether a piece of information is unclear along a particular facet, and if so, to know how to clarify it. Therefore, each facet is associated with one or more domain-independent clarity checking and clarification techniques. Both are in essence informal, for no solid conceptual and formal foundations exist for the various concepts (e.g., vagueness, ambiguity) that underlie the facets. In the present paper at least, and in absence of fully automated requirements

acquisition frameworks, the informal treatment of the present issue is deemed sufficient. The problem of providing a formal acquisition language which would allow explicit representation and reasoning about all of the four facets is not to be underestimated.

Because lack of clarity can appear in any fragment of information in a goal diagram or argument or elsewhere (i.e., information used as a source for goal modeling and argumentation), clarification need be applicable to any part of thereof. The basic approach consists of a labeling technique outlined below, involving the checking of information for lack of clarity, and subsequent application of clarification techniques associated to ambiguity, overgenerality, synonymy, and vagueness. All information obtained through clarification is written in a thesaurus.

Technique 4. (Clarification by labeling) Clarification techniques associated to the cited four facets share a common approach to the labeling of unclear information, albeit employing different techniques for the identification of elements to mark and their subsequent clarification. Labeling proceeds by the following steps:

1. Choose a word or expression and test it for lack of clarity along a particular clarity facet. Choice is not arbitrary, but guided by stakeholders' or engineer's questions about what a word or expression is intended to mean.
2. If unclear along one or more clarity facet (see, §8.5.2–8.5.5), place brackets around it and label it accordingly (below, assume that the fragment of interest in a sentence is of the form: "... word(s) ..."):
   a) If ambiguous, then "... ^{A_i}[word(s)]^{A_i} ...", where A is to refer to the ambiguity facet, and i is a number to ensure a unique reference to the given label.
   b) If overgeneral, then "... ^{G_i}[word(s)]^{G_i} ...".
   c) If synonymous, then "... ^{S_i}[word(s)]^{S_i} ..." with the same label (i.e., S_i) applied to all words synonymous with "word(s)".
   d) If vague, then "... ^{V_i}[word(s)]^{V_i} ...".
3. Clarify the element in brackets it by applying a technique suggested for the given facet.
4. Transfer the result of the clarification into the thesaurus, and enforce the result of clarification over other artifacts so that the agreed meaning is maintained accross the project.

Example 8.3. (Clarification by labeling) Returning to the excerpt used in Ex.8.2, labeling leads to the following:

> "All ^{G_1}[users]^{G_1} want ^{A_1}[total visibility]^{A_1} on ^{G_2}[^{S_1}[costs]^{S_1} and charging (cost recovery) mechanisms]^{G_2} regarding ATM system development and operation ^{S_1}[expenditures]^{S_1}. Constant monitoring of ATM services ^{S_1}[costs]^{S_1} and quality of service ('benchmarking') delivered to the ^{G_3}[client]^{G_3} is needed to ensure the ^{V_1}[cost effective]^{V_1} provision of such services." [83] (p.12)

Labels—A for ambiguity, G overgenerality, S synonymy, and $V$for vagueness—are thus introduced for each of the four facets, applied to unclear information according to results of checks for clarity, and are then ready for analysis using clarification techniques outlined in the following subsections.

Dealing with these four facets requires the understanding of linguistic phenomena that ambiguity, overgenerality, synonymy, and vagueness refer to, as discussed below.

8.5.2 Ambiguity

An encyclopedic entry [15] suggests that a word or an expression (i.e., several related words) is ambiguous if it has more than one meaning. Examples include words, such as "light" (which can mean not heavy or not dark), or phrases, such as "user's agendas provide their availability" (Whose availability is provided?). While people seem capable in many cases to intuitively detect the occurrence of ambiguity, a useful criterion for doing so seems elusive (e.g., [272, 105]). The aim at present is to suggest a practical criterion for ambiguity that is acceptable in many cases. In addition, it is required that ambiguity is distinguished from the other three facets. For instance, separating ambiguity from overgenerality can be difficult if a word designating a class is considered ambiguous if the designated class is divisible onto subclasses. If this is accepted as a necessary condition for ambiguity, and knowing that a subclass contains instances of the subdivided class (see, §8.5.3), all ambiguous words would be considered as general. However, this is not appropriate: Hospers [130] has argued that a word is ambiguous neither because (i) the class of objects it refers to can be broken down into smaller classes or subclasses, nor (ii) when it may have many instances of use. Moreover, it is now accepted that ambiguity does not require generality [105].

Ambiguity is multiplicity of meaning of an expression, regardless of whether it originates from polysemy [268] of individual words or from multiplicity of structural analyses [105]. Polysemy occurs when a word, taken out of context, admits multiple meanings [268]—e.g., "run" as a verb has 29 distinct meanings and 125 sub-meanings according to the Webster's dictionary. Context of use tends to resolve problems of polysemy in communication [268], in that a word which is taken with other words in an expression loses most of its alternative senses. It is, however, not necessary to have a polysemous word in an expression for it to be ambiguous, as the expression "in airports, the police cannot shoot suspects with guns" illustrates. In this latter case, it is not polysemy that generates ambiguity, but the fact that the proposition admits different structural analyses [105]—each structural analysis leads roughly to one alternative reading of this proposition. The reader should note that matter is more elaborate than the above indicates: for instance, negation with "not" in English is often ambiguous for it leaves open whether it is the truth or the assertability of a proposition that is negated.

It is therefore difficult to propose a unique and general clarity check for ambiguity. For instance, it is useful to check if there is a state of affairs in which the ambiguous expression can both be affirmed and denied. For instance, in a state in which "armed police cannot shoot unarmed suspects on airport grounds" and "armed police can shoot armed suspects on airport grounds" both hold, the expression "in airports, the police cannot shoot suspects with guns" can both be affirmed true and false: in the last expression, the reader cannot know who carries guns (and is thus armed)—the police or the suspects, or both. A true reading of the given expression is, e.g., "in airports, the armed police cannot shoot unarmed suspects", while a false one is "in airports, the police cannot shoot armed suspects". Tautologies and contradictions cannot, however, be addressed by the given clarity check. Moreover, this check does not point to the source of ambiguity—finding it requires additional knowledge about the language being used, and thus varies accross languages (e.g., [105]).

Although imperfect, structural analysis and word polisemy can be first used to detect potential for ambiguity. Once detected, the above clarity check is applied to verify if there is a state in which some alternative readings can be affirmed and others denied.

Definition 8.4. *An expression e is **ambiguous** if there are at least two of its alternative readings e_j and e_k such that one can be affirmed and the other denied within the body of knowledge (denoted $\mathcal{A}$, and referred to in the remainder as domain knowledge) contained in the project artifacts (i.e., goal diagram, arguments, thesaurus, stakeholders' requirements documentation) in which the expression is employed: ambiguous(e) $\not\models \bot$ iff:*

1. *There is a non-empty set E which contains alternative readings of e.*
2. *There are two readings e_j and e_k in E such that one gives rise to inconsistency given $\mathcal{A}$, while the other does not: $\exists e_j, e_k \in E$ s.t. $(\mathcal{A}, e_i) \models \bot$ and $(\mathcal{A}, e_j) \not\models \bot$.*[1]

Technique 5. (Brute force ambiguity check) Identify as many alternative readings as feasible, and search for information relevant to the given body of knowledge which when combined with the readings is consistent with some but inconsistent with other. While it may seem cumbersome, this form of clarity checking is often feasible, for many alternative readings can be intuitively eliminated, whereas the remaining few can be subjected to scrutiny to relevant stakeholders through informal discussion.

Technique 6. (Ambiguity resolution in the thesaurus) The ambiguous element is labeled and introduced in the thesaurus with a list of alternative readings elicited by the engineer. Resolving ambiguity amounts to choosing one of the available readings and enforcing it throughout other fragments of the requirements specification through a thesaurus $\mathcal{D}$, where the the ambiguous word is carried over and accompanied with its chosen reading.

Example 8.5. Consider the following excerpt from a stakeholder expectation about the ATM:

> "...also applies to the pre-tactical (^{A_i}[day minus one]^{A_i}..." [83] (p.17)

Speaking of duration in terms of days engenders ambiguity by polysemy, as different and contradictory interpretations are available for "day" in "day minus one"[2]—among others: 24h before takeoff, and working day before takeoff. Because this seems to be a case of ambiguity from polysemy, the Techn.5 above applies. The identified alternative readings are written down in the thesaurus:

[1] No restrictions are placed on the way domain knowledge is represented—both informal and formal representations of stakeholders' knowledge about the domain are allowed in $\mathcal{A}$.

[2] Similar can be said for "day" in "same day" but the example focuses on "day minus one" only.

(Ambiguity, A_i) **Day minus one:** 1. 24h before takeoff; •
2. One working day before takeoff;

The ambiguous expression "day minus one" has been labeled according to Techn.4. The expression, along with the alternative readings and the decision (i.e., the choice of a reading—decorated with "•") is transferred to the thesaurus. Choosing randomly one of the interpretations is inappropriate, for the probable impact an inadequate choice would have—e.g., scheduling problems, which are bound to result in lack of efficiency in airport operations.

Technique 7. (Structural analysis ambiguity check) In English, knowing that a "noun phrase can have complementary propositional phrases" and a "verb phrase can contain just a verb and propositional phrase" [105] allows showing that "in airports, the police cannot shoot suspects with guns" admits two structural analyses: 1) in airports, the police [[cannot shoot]$^{\text{verb}}$ [suspects [with guns]$^{\text{prop. phrase}}$]$^{\text{noun phrase}}$]$^{\text{verb phrase}}$; and 2) in airports, the police [cannot shoot [suspects [with guns]$^{\text{prop. phrase}}$]$^{\text{noun phrase}}$]$^{\text{verb phrase}}$.

Example 8.6. For the meeting scheduler, the following is ambiguous:

"^{A_j}[User's agendas provide their availability.]^{A_j}"

as the following two readings can be identified using structural analysis:

(Ambiguity, A_j) **User's agendas provide their availability:**
1. Agendas that belong to the users provide the availability of the users; •
2. Agendas that belong to the users provide information on whether they (the agendas) are available for fulfilling a request;

8.5.3 Overgenerality

Overgenerality is introduced to characterize a relationship present between expressions such as "provide awareness of traffic in the areas involving aircraft movement" and "provide awareness of traffic in the areas involving helicopter movement", where the first is general with regards to the second if helicopter is understood to designate a subclass of aircraft.

Definition 8.7. *A word g is **overgeneral** if domain knowledge $\mathcal{A}$ contains an expression about an instance or specialization of g, say p, whereby the expression with p contradicts with an expression containing g. As a convention, let $F(a) \in \mathcal{A}$ in the definition indicate that an expression $F(a)$ containing a word of interest a is in domain knowledge $\mathcal{A}$. $F(a) \in \mathcal{A}$, that is, there is an affirmable expression $F(a)$ in domain knowledge, where a is a word that occurs once in $F(a)$ and can be replaced with the particular word p or the general word g. Then, by convention, $F(a)/g$ denotes the affirmable expression obtained by replacing the word a with the word g. We have overgeneral$(g) \not\models \bot$ iff:*

1. *The expressions $F(a)/g$ and $F(a)/p$ are inconsistent given the domain knowledge in $\mathcal{A}$: $\mathcal{A}, (F(a)/g), (F(a)/p) \models \bot$.*
2. *general(p, g) holds, i.e., it must be verified that g is a general of p. A word g is general with respect to a word p iff for an affirmable expression $F(a)$, the expression $\neg(F(a)/g) \wedge (F(a)/p)$ is a contradiction.*

The given clarity check for generality was suggested in [211] and defended against alternative checks in [105]—it is taken as appropriate for the purpose herein. In the ATM case study for instance, g can stand for the word "aircraft" and p for "helicopter", see, Ex.8.8.

Technique 8. (Criteria-based resolution of overgenerality) Define criteria for distinguishing entities that fall in, from those that do not fall in the set designated by the expression $F(a)$ in which substitution for g and p gives inconsistency. The entry in the thesaurus explicates the criteria used to discriminate what particulars can be used as substitutes for a in $F(a)$.

Example 8.8. If the general piece of information is "aircraft landing is allowed on strip Z", if helicopter is considered a particular of aircraft, and if the particular appears in an expression, e.g., "helicopter landing is not permitted on strip Z", then the two expressions are inconsistent, making the expression "aircraft landing is allowed on strip Z" overgeneral. A criterion that may be used to clarify over the overgenerality facet in this case would express that helicopters do not land on airstrips.

Example 8.9. The following fragment expresses stakeholders expectations about aircraft movement:

> "The main goal of an A-SMGCS is to provide all 'control authorities' ... with positive situation awareness of traffic evolving in the areas used for ^{G_i}[aircraft movement]^{G_i}, the runway strip..." [83], (p.22)

It is expressed elsewhere in the documentation that aircraft movement inside hangars and repair areas is not to be communicated to all control authorities. Applying the Techn.8 results in adding the expression labeled above as an entry in the thesaurus and providing a criterion which accounts for the exception:

(Overgenerality, G_i) **Aircraft movement:** excludes aircraft movement inside hangars and repair areas.

Example 8.10. Simplistic expressions of expectations tend to contain overgeneral information, e.g.:

> "^{G_j}[Any user of the meeting scheduler can initiate a meeting]^{G_j}."

(Overgenerality, G_j) **Any user of the meeting scheduler can initiate a meeting:** Full-time employees that have at least the managerial status in the research department can initiate meetings.

8.5.4 Synonymy

Synonymy treats the use of common terminology in an inconsistent manner—it highlights the use of syntactically different terms, which are given the same semantics.

Definition 8.11. *Words w_1 and w_2 are synonymous within $\mathcal{A}$ if they are can be used interchangeably in $\mathcal{A}$: synonymous(w_1, w_2) holds iff:*

1. *Both words appear in $\mathcal{A}$: $w_1, w_2 \in \mathcal{A}$;*
2. *w_1 appears in expression $F_1(w_1)$ and w_2 appears in expression $F_2(w_2)$, and interchanging the two words within the expressions maintains the meaning of the new expressions equivalent with original ones. That is, $F_1(w_1)/w_2$ has the same meaning as $F_1(w_1)$, and $F_2(w_2)/w_1$ has the same meaning as $F_2(w_2)$.*

Technique 9. (Synonymy by interchangeability check) Intuitively, words used within similar expressions are candidates for synonymy. Clarity checking for synonymy proceeds by replacing a word in an expression with another word within one or more expressions in which the first appears. For instance, if $F(a)$ and $G(b)$ are two expressions appearing in domain knowledge, and the requirements engineer believes that both expressions refer to the same properties or behaviors of the IS, then if the engineer understands $F(a)/b$ (i.e., the expression $F(a)$ in which each occurrence of the word a is replaced with the word b) in the same way as $F(a)$, and $G(b)/a$ in the same way as $G(b)$, then a and b are synonymous for the given expressions.

Example 8.12. Consider the following potential synonymy:

> "For ^{S_i}[incident]^{S_i} and ^{S_i}[accident]^{S_i} investigation purposes the ATM network must provide mechanisms to record and make available any data..." [83] (p.23)

By applying Techn.9 over the above and other requirements fragments, it has been established that the two expressions "incident investigation" and "accident investigation" are not synonymous, leading to:

(Synonymy, S_i) **Accident investigation $\neq$ Incident investigation:** distinct processes, which is executed depends on the gravity of the event.

Example 8.13. The excerpt below illustrates how synonymy can be involved in inconsistency—resolving it expectedly resolves the inconsistency:

> "^{S_j}[A meeting participant]^{S_j} is anyone who has confirmed attendance at the meeting... ^{S_j}[A participant]^{S_j} is any the person who attends the meeting, except the initiator."

(Synonymy, S_j) **Meeting participant = participant:** A participant is any the person who has confirmed attendance and attends the meeting, except the meeting initiator.

Technique 10. (Resolving synonymy by equivalence or criteria in case synonymy is absent) In case two words are checked for synonymy, resulting in the conclusion that they are nor synonymous, the criteria for their distinction are provided in the corresponding thesaurus entry to avoid any further questioning. Otherwise, if words are considered synonymous, their equivalence is written down in the thesaurus as well.

8.5.5 Vagueness

According to [17], "any grammatical element whose contribution to truth conditions requires perception, categorization, or judgement of gradient contingent facts—including tense, aspect, and plurality suffers from an incurable susceptibility to vague uncertainty". Consider the expression:

> "All users want total visibility on costs and charging (cost recovery) mechanisms regarding ATM system development and operation expenditures. Constant monitoring of ATM services costs and quality of service ('benchmarking') delivered to the client is needed to ensure the cost effective provision of such services." [83] (p.12)

The above is vague in that what exactly means to count as "cost effective" is indeterminate. According to linguists (e.g., [173] and related), such expressions have three distinguishing characteristics:

1. *Truth conditional variability:* The truth valuation of the expression depends on the context in which it is used.
2. *Existence of borderline cases:* Whatever the context of use, there will generally be sets of entities to which "is cost effective" either clearly applies or not, but there will also be entities for which it is difficult or impossible to determine whether the said predicate applies or not.
3. *The Sorites Paradox:* When employed within a particular form of argument, the predicate will give rise to the Sorites paradox: e.g., if a $2 million project is cost effective, and any project that costs 1 cent less than a cost effective one is still cost effective, then any project is cost effective. In such a line of reasoning, the argument appears valid, premises true, yet the conclusion false.

Admitting vagueness in a requirements specification leaves room for questioning the degree to which the goals are satisfied *after* the corresponding software has been built. Leaving vague expectations thus leaves room for misunderstanding. An immediate consequence thereof is the difficulty with translating expectations into development or evaluation decisions, that is, difficulty to operationalize the requirements specification. This is precisely because calling a part of the software, e.g., "highly usable" will depend on the stakeholder, or, more importantly, will depend on what the stakeholder knows about usability and about the system in question: it will depend on the conditions in which the evaluation is to be made. The more the engineer and the stakeholders agree on the content of this set of information, the more likely that vagueness will be reduced.

The locus of vagueness in many vague expressions, just as the one above, is the presence of a predicate headed by a *gradeable adjective*, (above: "effective"). Such predicate designates a property of having a degree of cost that is at least as great as some standard of comparison of cost, that itself is not part of the meaning of "effective" but is determined by the context in which the said adjective is used. From there on, truth assignment can change as the standard changes (and as context changes).

This matter is, however, more intricate, as setting a standard of comparison seems to eliminate borderline cases altogether (and subsequently the Sorites paradox). While attractive, this seems removed from reality: assuming e.g., that $2.5 million is a mean cost for similar projects (and, say, the standard for comparison), then some stakeholders may still refuse to accept that the given project of $2 million is cost effective, for they have witnessed similar projects in which cost was significantly lower (i.e., the variance in the sample from which the mean was computed can be considered high). A solution to this is suggested in [110], where for a borderline case to be described truthfully with the given vague predicate, it is necessary for it not to exceed the standard without exceeding it by a *significant* amount—in practice, two very similar cases along the scale of measurement associated to the vague predicate will be taken same (i.e., will carry the same truth valuation) if the cost of discriminating between them outweighs the benefits of doing so. They will count as the same for the given purposes [110]. Unfortunately, it seems that how much significant it is, is itself vague—the only realistic solution then remains seeking stakeholders' agreement on the standard and its enforcement throughout a chosen context.

These established positions on gradable adjectives (e.g., [173, 110][3]) already provide relevant practical indications for the problem at hand.

Definition 8.14. *An adjective e is gradeable and can be assumed giving rise to* ***vagueness*** *within the expression in which it appears, if the following conditions are met (it is said then that vague(e) holds):*

1. *1st assumption on the gradeable adjective: The adjective maps its arguments onto abstract representations of measurement, or degrees.*

[3] The reader is reminded that the present work is not one focused on linguistics, so that no specific references will be given beyond overview and extensive discussions from the aforementioned field. For instance, no minority positions are mentioned herein. For details, the reader will refer to the works cited within the given references.

2. *2nd assumption on the gradeable adjective: The set of degrees totally ordered with respect to some dimension (e.g., cost, size, etc.) constitute a scale. The 1st and 2nd condition together give an ontology for gradeable adjectives which provides indications on what adjectives to consider when clarity checking for vagueness.*
3. *Presence of context-dependent standard of comparison: The adjective itself does not entail a standard for comparison, so that such a standard varies with context.*
4. *Presence of borderline cases: It should be possible to identify borderline cases of application of the given adjective.*
5. *Truth conditional variability: The truth of the expression in which the adjective appears should vary with the change of standard which is accepted to distinguish when the adjective applies from when it does not.*
6. *The Sorites Paradox: The predicate generated by the adjective can be used in lines of reasoning that follow the one taken in the Sorites paradox (see above).*

Technique 11. (Vagueness check) See the conditions for the presence of vagueness in Def.8.14.

Technique 12. (Resolving vagueness from gradeable adjectives) For gradeable adjectives that generate vagueness, the desired solution is to specify the standard of comparison, and to treat borderline cases individually. Within the specification process, the identified source of vagueness is placed between brackets $[\ldots]^V$, transferred to the thesaurus, and associated with a sharp criterion.

Example 8.15. Returning to the example cite above, consider the following fragment:

> "...quality of service ('benchmarking') delivered to the client is needed to ensure the ^{V_i}[cost effective]^{V_i} provision of such services." [83] (p.12)

"(Cost) effective" is a gradeable adjective—applying Techn.12 results in the definition of a criterion or standard of comparison, as follows:

(Vagueness, V_i) **Cost effective:** Not cost effective if above the budget permitted at the outset of the development project.

If feasible, the above thesaurus entry can be extended to include explicit indications on the scale to employ when measuring the degree of cost effectiveness—e.g., if a global indicator of stakeholder satisfaction is available, it can be combined to the difference between actual project cost and the budget.

Example 8.16. In the following statement about the meeting scheduler system, the word "few" generates vagueness:

> "The scheduler should send a reminder ^{V_j}[a few days before the meeting date]^{V_j}."

(Vagueness, V_j) **A few days before the meeting date:** The meeting initiator should have a choice of automatically informing the participants 1, 2, ..., 7 days before the meeting date.

Following the above discussion, it appears that many softgoals identified early on in the Tropos RE process can be qualified as vague (as discussed in [151]). In this respect, softgoals can be dealt with at the very early stages using GAM in a relatively novel manner, without harming the applicability of established RE approaches that employ the softgoal concept. For instance, instead of leaving a softgoal "1.Alt2 User friendly" of the meeting scheduler as is and then establishing contribution links between this and other goal diagram elements, taking this softgoal as vague in GAM would require the application of clarification techniques, which would likely result in more detail as to what is actually meant and agreed upon to be the meaning of the given softgoal. The information obtained by clarification and written in the thesaurus can subsequently serve as the source for finding ways to operationalize the clarified softgoal.

8.5.6 Clarification and Argumentation

As suggested above, the given techniques can be applied to clarify information used and produced when using the GAM decision process, being therefore useful both to clarify the information to be placed in a goal diagram and the information employed when justifying the modeling choices. In both cases, arguments can be compared over specificity (as usual in the argumentation modeling literature—see, §8.6), but also according to their relative clarity: the following section introduces preferences orderings over arguments based on the presence or absence of clarity therein. For instance, an argument can be rejected in favor of another one if the former is unclear and the latter clear along a clarity facet. Clarification thus provides additional informal criteria for the comparison and discrimination between alternative arguments.

8.6 Argumentation and Justification

With clarification, the information used and produced when applying the GAM decision process can be checked for clarity and clarified to avoid misunderstanding. Returning to the evaluation activity of the decision process, argumentation and justification are introduced to provide a structured approach to the discussion, documentation, and revision of rationale behind modeling decisions—in the evaluation step, techniques presented below serve as an integrative approach to qualitative selection among alternative modeling choices.

Argumentation modeling literature [46] in the artificial intelligence field focuses on formalizing commonsense reasoning in the aim of automation. An *argumentation model* [310] is a static representation of an *argumentation process*, which can be seen as a search for arguments, where an argument consists of a set of rules chained to reach a conclusion. Each rule can be rebutted by another rule based on new information. To formalize such defeasible reasoning, elaborate syntax and semantics have been developed (e.g., [46, 289, 27]) commonly involving a logic to formally represent the argumentation process and reason about argument interaction.

A structured argumentation system (i.e., a model and processes employing the model) is introduced in GAM for a number of reasons: (i) it is needed for a rigorous justification process in the *Evaluation* step of the GAM decision process; (ii) content of the decision diagram can be more closely related to the content of the goal diagram than was illustrated in the previous section; and, (iii) the structured approach is a basis for automating some of the argumentation-specific analyses in GAM.

Example 8.17. To further illustrate the need for an argumentation system in GAM, consider the example in Fig.4. In the problem statement 3 (*PS3*), assume that a decision has been made to adopt alternative *3.Alt2* and not *3.Alt1*. Such a decision can be considered as justified only because the clearly unsupportive argument in *3.Alt2* (i.e., *3.Alt2.A4-*) is rebutted by *3.Alt2.A5+*, while the other negative argument *3.Alt2.A1-* is written in such way that its second part provides support against its first part. The requirements engineer could overlook the ambiguity in *3.Alt2.A1-* and conclude that there are no arguments that interfere with *3.Alt2*, accepting it then as justified. In contrast, because there are no arguments that interfere with the negative ones, which in turn interfere with *3.Alt1* (i.e., *3.Alt1.A2-* to *3.Alt1.A4-*), choosing *3.Alt1* is not justified. In presence of an argumentation system, it is required that the arguments for each alternative be more precise in terms of their structure, and their relationships explicit for more rigor in justification.

To arrive at a structured argumentation system, the concept of argument is first defined below, followed by a set of argument relationships, and the justification process.

Definition 8.18. *An argument is defined recursively as follows:*

1. *Any information of the form $P \therefore c$ is an argument, where c is called "conclusion" and is a speech act of any type (i.e., assertive, directive, commissive, expressive, or declarative [283]), $\therefore$ is a conclusion indicator (read it "therefore"), and P is a set of assertive propositions suggested to support the conclusion (such a proposition is called "premise").*
2. *If for A and B such that $A = P_A \therefore c_A$ and $B = P_B \therefore c_B$, $P_A \subseteq P_B$ then A is a subargument of B.*
3. *An argument cannot have its conclusion as a premise for its conclusion: if $A = P_A \therefore c_A$ and $c_A \in P_A$ then A is not an argument.*
4. *There can be no inconsistent propositions in P.*
5. *Nothing is an argument unless it obeys the rules above.*

The suggested definition is common in philosophy (see, e.g., [128] for a discussion) and AI (for an overview, see [46]). It has the desirable properties of avoiding inconsistent information as support for a conclusion, allows complex arguments, in which a premise can be a conclusion of another argument, and bans cyclical argumentation.

The above definition can be formalized as follows. Assuming a first-order language $\mathcal{L}$ defined as usual, let $\mathcal{K}$ represent a consistent set of formulae (i.e., $\mathcal{K} \nvdash \bot$), each being a piece of information about the universe of discourse, and let $\mathcal{K} \equiv \mathcal{K}_N \cup \mathcal{K}_C$. Members of the set $\mathcal{K}_N$, called *necessary knowledge*, represent facts about the universe of discourse and are taken to be formulae which contain variables, thus stating relationships. Necessary knowledge is taken at face value (i.e., is assumed unquestionable). The set $\mathcal{K}_C$, called *contingent knowledge* are information that can be put in question or argued for. It is then said that the knowledge an agent a (here, a stakeholder of the RE project) can use in argumentation is given by the pair (K_a, Δ_a), where K_a is a consistent subset of $\mathcal{K}$ (i.e., $K \subset \mathcal{K}$ and $K \nvdash \bot$), and Δ_a is a finite set of *defeasible rules*. A defeasible rule has the form $\alpha \rightsquigarrow \beta$. The relation $\rightsquigarrow$ between formulae α and β is understood to express that "reasons to believe in the antecedent α provide reasons to believe in the consequent β". In short, $\alpha \rightsquigarrow \beta$ reads "α is reason for β".

Definition 8.19. *Let A a set of agents (e.g., stakeholders), $K \equiv \bigcup_{a \in A} K_a$, and $\Delta \equiv \bigcup_{a \in A} \Delta_a$. Given (K_a, Δ_a) and $P \subset \Delta_a^{\downarrow}$, where $\Delta_a^{\downarrow}$ is a set of formulae from Δ_a instantiated over constants of the formal language (i.e., variables appearing in these formulae are replaced with specific values), P is an* ***argument*** *for $c \in \mathcal{K}_C$, denoted $\langle P, c \rangle_K$, if and only if:*

1. $K \cup P \mathrel{|\!\sim} c$ *(K and P derive c)*
2. $K \cup P \nvdash \bot$ *(K and P are consistent)*
3. $\nexists P' \subset P, K \cup P' \mathrel{|\!\sim} c$ *(P is minimal for K)*

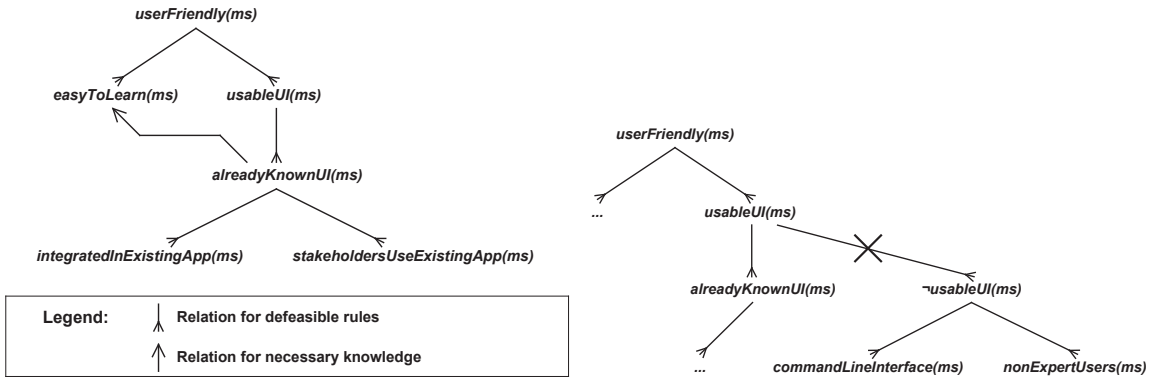
Where "$\mathrel{|\!\sim}$" is called the defeasible consequence [289] and is defined as follows. Define $\Phi = \{\phi_1, \ldots, \phi_n\}$ such that for any $\phi_i \in \Phi$, $\phi_i \in K \cup \Delta^{\downarrow}$. A formula ϕ is a defeasible consequence of Φ (i.e., $\Phi \mathrel{|\!\sim} \phi$) if and only if there exists a sequence $B_1, \ldots, B_m$ such that $\phi = B_m$, and, for each $B_i \in \{B_1, \ldots, B_m\}$, either B_i is an axiom of $\mathcal{L}$, or B_i is in Φ, or B_i is a direct consequence of the preceding members of the sequence using modus ponens or instantiation of a universally quantified sentence.

The formal argument definition given above is well-understood and established in the AI literature. Looking at alternative approaches (for extensive overviews, see, [46, 264]), the principal difference is in the choice of "$\mathrel{|\!\sim}$", that is, of the assumptions made on how the premises formally relate to the conclusion. For instance, [27] situate their discussion within classical propositional logic and take classical logical consequence "$\vdash$" instead. Choosing one over other formal approaches to argumentation (here, [289]) does not impose significant restrictions on the present discussion—the intuition behind the definitions proposed here are well known and do not differ within the AI argumentation literature.[4] Observe that the formal and the intuitive definition of argument fit—for an argument $P \therefore c$, the formal definition writes $\langle P, c \rangle_K$[5] so that the $\therefore$ is replaced with binary relationships (being either "$\rightarrow$" in case of necessary knowledge or the "$\rightsquigarrow$" for defeasible knowledge) between premises in P; also, definitions of subargument do not differ, minimality ensures that there is no circular argumentation, and consistency of premises is ensured in both definitions.

While an argument can be constructed by combining explicitly expressed knowledge (e.g., from a knowledge base, as is often the case in AI argumentation modeling literature), the aim with GAM is to start from a conclusion and build arguments that support it from the knowledge that stakeholders provide, and that can be related to the conclusion. An important observation about the above definitions is that an argument $P \therefore c$ or, equivalently, $\langle P, c \rangle$ can be seen as a tree in which the root is the conclusion c and the leaves are members of P. A subargument is then a subtree in the tree of the argument. The evaluation activity of the GAM decision process amounts to combining two techniques: the argumentation of each alternative, which consists of constructing an *argument tree* AT for each alternative, whereby the root of the tree is the alternative, and the nodes are premises proposed to support the given alternative; and the comparison of alternatives, which is the identification of arguments that counterargue the arguments which appear in argument trees.

[4] It is, however, significant to note that the choice of derivation *is* critical if the aim is to build arguments automatically from a knowledge base: in case, e.g., arguments for requirements are to be obtained automatically from a knowledge base, the derived arguments will differ depending on the chosen derivation. It is obvious that following the above suggestions would require a knowledge base which contains defeasible rules.

[5] In the remainder, the subscript K will be omitted, since no knowledge base other than K (which is taken here to contain any knowledge that the stakeholders can provide) will be used.



(a) Argumentation (b) Counterargumentation

Fig. 5. In (a), an argument tree with premises supporting the conclusion userFriendlyUI(ms). In (b), argument concluding ¬usableUI(ms) counterargues the argument concluding userFriendlyUI(ms), at usableUI(ms).

Technique 13. (Argumentation of an alternative) Supporting an alternative *Alt* consists of recursively defining an argument tree AT_{Alt} as follows:

1. Define *Alt* as the root of the tree AT_{Alt} and set $c = Alt$.
2. Let $\langle P, c\rangle$. Identify $p_1, \ldots, p_n$ such that $\{p_1, \ldots, p_n\} = P$, $P \subseteq K \cup \Delta^{\downarrow}$.
3. Define a node for each premise $p_i \in P$ and define an edge from that node to c. Draw the edge "$\longrightarrow$" if $p \in K$, or as shown in Fig.5(a) in case $p \in \Delta^{\downarrow}$ (i.e., notice that the symbol for defeasible rule relation "$\rightsquigarrow$" is changed in the argumentation tree in Fig.5).

Each premise can be in turn argued for, that is, steps 1 to 3 above can be repeated for each premise $p_i \in P$ by letting p_i be a conclusion of a new argument, e.g., $\langle P_i, p_i\rangle$. The tree built for $\langle P_i, p_i\rangle$ is a subtree of the tree built for $\langle P, c\rangle$. By building subtrees subsequently for premises of $\langle P_i, p_i\rangle$ and this for some or all $p_i \in P$, the argumentation can proceed until the stakeholders and/or the requirements engineer judge that the argument tree for *Alt* has been constructed to a satisfactory extent.

Example 8.20. The following example illustrates how the definitions above are used to build an argument in GAM. Consider the suggestion (see, Fig.4): "*1.Alt2.A3+: (The Meeting Scheduler is) user friendly if well integrated in the user interface of the email client.*" For a meeting scheduler *ms*, a formula *c* for which we wish to argue or interfere with can be written: *userFriendly(ms)*. Stakeholders may then suggest a set of defeasible rules:

$\Delta = \{$ *easyToLearn(x)* $\wedge$ *usableUI(x)* $\rightsquigarrow$ *userFriendly(x)*, *standardizedUI(x)* $\vee$ *alreadyKnownUI(x)* $\rightsquigarrow$ *usableUI(x)*, *integratedInExistingApp(x)* $\wedge$ *stakeholdersUseExistingApp(x)* $\rightsquigarrow$ *alreadyKnownUI(x)* $\}$

And the following necessary knowledge, where "$\rightarrow$" is the standard implication operator:

$K = \{$ *alreadyKnownUI(x)* $\rightarrow$ *easyToLearn(x)* $\}$

An argument tree that supports *userFriendly(ms)* can be constructed using the given knowledge and defeasible rules, and represented using a tree-like structure shown in Fig.5(a). The root of the tree is the sentence for which the argument structure provides support. The defeasible and necessary knowledge have been instantiated over the meeting scheduler *ms*. The argument in Fig.5(a) can be written:[6]

$\langle\{$ *integratedInExistingApp(ms)* $\rightsquigarrow$ *alreadyKnownUI(ms)*, *stakeholdersUseExistingApp(ms)* $\rightsquigarrow$ *alreadyKnownUI(ms)*, *alreadyKnownUI(ms)* $\rightsquigarrow$ *usableUI(ms)*, *usableUI(ms)* $\rightsquigarrow$ *userFriendly(ms)*, *easyToLearn(ms)* $\rightsquigarrow$ *userFriendly(ms)* $\}$, *userFriendly(ms)* $\rangle$

Where:

$\langle\{$ *integratedInExistingApp(ms)* $\rightsquigarrow$ *alreadyKnownUI(ms)*, *stakeholdersUseExistingApp(ms)* $\rightsquigarrow$ *alreadyKnownUI(ms)* $\}$, *alreadyKnownUI(ms)* $\rangle$

is one of the subarguments of the argument shown in Fig. 5(a).

[6] Note that the argument is extracted from the tree by ensuring minimality, according to Def.8.19, so that some branches of the tree need not be maintained.

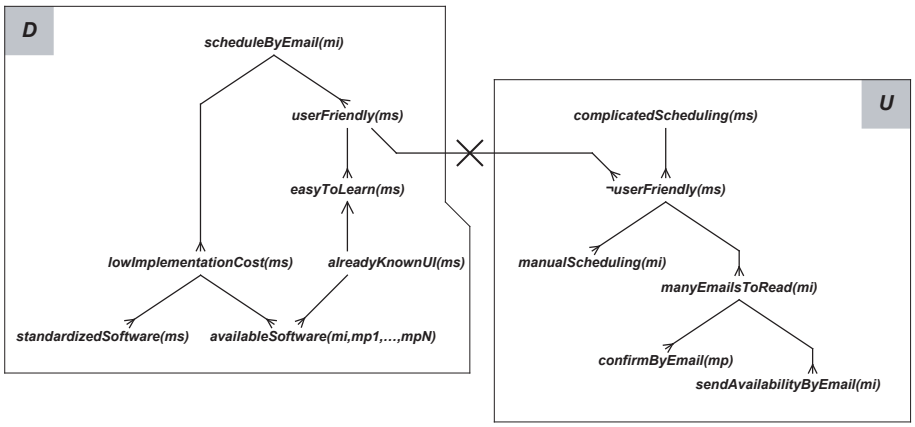


Fig. 6. A dialectical tree for 3.Alt1 "Schedule meeting using email only".

The argument tree (or, simply argument) shown in Fig.5(a) contains only information that supports the conclusion that the meeting scheduler is user friendly. Albeit being useful as it is, a particular interest in argumentation is in confronting arguments and rejecting some conclusion in favor of other. It is therefore necessary to define relationships between arguments.

Definition 8.21. *Two arguments* $\langle P_1, c_1\rangle$ *and* $\langle P_2, c_2\rangle$ ***disagree****, denoted by:*

$$\langle P_1, c_1\rangle \bowtie_{\mathcal{K}} \langle P_2, c_2\rangle$$

if and only if $\mathcal{K} \cup \{c_1, c_2\} \vdash \bot$.

Definition 8.22. *Instead of seeking contradiction of conclusions, the* ***counterargument*** *relation looks at the incompatibility of an argument's conclusion with the conclusion of a subargument of another argument. An argument* $\langle P_1, c_1\rangle$ *counterargues at c the argument* $\langle P_2, c_2\rangle$*, denoted by:*

$$\langle P_1, c_1\rangle \not\rightsquigarrow^{c} \langle P_2, c_2\rangle$$

if and only if there is a subargument $\langle P, c\rangle$ *of* $\langle P_2, c_2\rangle$ *such that* $\langle P_1, c_1\rangle \bowtie_{\mathcal{K}} \langle P, c\rangle$.

Example 8.23. The following argument counterargues at *usableUI(ms)* the argument in Fig.5(a):

$\langle\{$ *commandLineInterface(ms)* $\wedge$ *nonExpertUsers(ms)* $\rightsquigarrow \neg$*usableUI(ms)* $\}, \neg$*usableUI(ms)* $\rangle$

Counterargumentation is represented by a crossed line, as in Fig. 5(b), directed from the root (i.e., conclusion) of the argument that counterargues to the node which is countered.

Definition 8.24. *In case two arguments are such that one counterargues the other, it is necessary to determine which of the two is to be maintained. An argument* $\langle P_1, c_1\rangle$ ***defeats at c*** *an argument* $\langle P_2, c_2\rangle$*, denoted by:*

$$\langle P_1, c_1\rangle \gg^{c} \langle P_2, c_2\rangle$$

if and only if:

1. $\langle P_1, c_1\rangle \not\rightsquigarrow^{c} \langle P_2, c_2\rangle$*; that is,* $\langle P_1, c_1\rangle$ *counterargues* $\langle P_2, c_2\rangle$ *at c;*
2. *there is a subargument* $\langle P, c\rangle$ *of* $\langle P_2, c_2\rangle$ *such that* $\langle P_1, c_1\rangle \succ^{spec} \langle P, c\rangle$*; i.e.,* $\langle P_1, c_1\rangle$ *is more specific than* $\langle P, c\rangle$.

Definition 8.25. *The specificity relation "*$\succ^{spec}$*" is an order relation over arguments, defined so that arguments containing more information, i.e., which are more specific, are preferred over other. An argument* $\langle P_1, c_1\rangle$ *is strictly more specific than* $\langle P_2, c_2\rangle$*, denoted by:*

$$\langle P_1, c_1\rangle \succ^{spec} \langle P_2, c_2\rangle$$

if and only if:

1. $\forall e \in \mathcal{K}_C$ *such that* $\mathcal{K}_N \cup \{e\} \cup P_1 \mathrel{|\!\sim} c_1$ *and* $\mathcal{K}_N \cup \{e\} \not\mathrel{|\!\sim} c_1$*, also* $\mathcal{K}_N \cup \{e\} \cup P_2 \mathrel{|\!\sim} c_2$*, and*

2. $\exists e \in \mathcal{K}_C$ *such that:*
a) $\mathcal{K}_N \cup \{e\} \cup P_2 \mid\sim c_2$
b) $\mathcal{K}_N \cup \{e\} \cup P_1 \not\mid\sim c_1$
c) $\mathcal{K}_N \cup \{e\} \not\vdash c_2$

Example 8.26. The argument:

$\langle\{$ *easyToLearn(ms)* $\wedge$ *usableUI(ms)* $\rightsquigarrow$ *userFriendly(ms)* $\}$, *userFriendly(ms)* $\rangle$

is more specific than:

$\langle\{$ *easyToLearn(ms)* $\rightsquigarrow \neg$*userFriendly(ms)* $\}$, $\neg$*userFriendly(ms)* $\rangle$

because although *easyToLearn(ms)* alone is taken as sufficient for arguing the conclusion $\neg$*userFriendly(ms)*, it cannot by itself be sufficient to argue *userFriendly(ms)*. If *easyToLearn(ms)* $\wedge$ *usableUI(ms)* alone is used to argue *userFriendly(ms)*, the argument that supports $\neg$*userFriendly(ms)* can also be concluded. In other words, the former argument contains at least the same premises as the latter—the additional information makes it more specific.

Technique 14. (Comparison of arguments over clarity) Arguments employed in the examples above are constructed of premises and conclusions made of primitive propositions, which in practice are references to parts of the specification or initial documentation, or are replaced with sentences of natural language which express in a rich manner the given premise or conclusion. There is therefore no particular guarantee that the content of premises or conclusions carries clear content. Consequently, premises and conclusions can be clarified over ambiguity, overgenerality, synonymy, and vaguenes using the techniques presented earlier. In addition then to comparing arguments using the ordering relation defined by specificity, the following order relations can be defined:

- Arguments containing non-ambiguous information are preferred to those containing ambiguous information:

$$\langle P_1, c_1\rangle \succ^A \langle P_2, c_2\rangle$$

iff $\exists e \in P_2 \cup \{c_2\}$ s.t. *ambiguous*$(e) \not\vdash \bot$ and $\forall e' \in P_1 \cup \{c_1\}$, *ambiguous*$(e') \vdash \bot$.
- Arguments containing non-overgeneral information are preferred to those containing overgeneral information:

$$\langle P_1, c_1\rangle \succ^G \langle P_2, c_2\rangle$$

iff $\exists e \in P_2 \cup \{c_2\}$ s.t. *overgeneral*$(e) \not\vdash \bot$ and $\forall e' \in P_1 \cup \{c_1\}$, *overgeneral*$(e') \vdash \bot$.
- Arguments containing non-synonymous information are preferred to those containing synonymous information:

$$\langle P_1, c_1\rangle \succ^S \langle P_2, c_2\rangle$$

iff $\exists e_i, e_j \in P_2 \cup \{c_2\}$ s.t. *synonymous*$(e_i, e_j) \not\vdash \bot$ and $\forall e_k, e_l \in P_1 \cup \{c_1\}$, *synonymous*$(e_k, e_l) \vdash \bot$.
- Arguments containing non-vague information are preferred to those containing vague information:

$$\langle P_1, c_1\rangle \succ^V \langle P_2, c_2\rangle$$

iff $\exists e \in P_2 \cup \{c_2\}$ s.t. *vague*$(e) \not\vdash \bot$ and $\forall e' \in P_1 \cup \{c_1\}$, *vague*$(e') \vdash \bot$.

Technique 15. (Justification) The *justification process* consists of recursively defining and labeling a *dialectical tree* $\mathcal{T}\langle P, c\rangle$ as follows:

1. A single node containing the argument $\langle P, c\rangle$ with no defeaters is by itself a dialectical tree for $\langle P, c\rangle$. This node is also the root of the tree.
2. Suppose that $\langle P_1, c_1\rangle, \ldots, \langle P_n, c_n\rangle$ each defeats $\langle P, c\rangle$. Then the dialectical tree $\mathcal{T}\langle P, c\rangle$ for $\langle P, c\rangle$ is built by placing $\langle P, c\rangle$ at the root of the tree and by making this node the parent node of roots of dialectical trees rooted respectively in $\langle P_1, c_1\rangle, \ldots, \langle P_n, c_n\rangle$. One way of finding arguments $\langle P_1, c_1\rangle, \ldots, \langle P_n, c_n\rangle$ that defeat $\langle P, c\rangle$ is to look for arguments that support the negation of a premise in P or the negation of the conclusion c.

$$
\begin{aligned}
\mathit{goal(Name)} &\overset{\approx \text{goal}}{\longleftrightarrow} \mathit{achieve(f)} \mid \mathit{maintain(f)} \mid \mathit{achieve\&maintain(f)} \mid \mathit{avoid(f)} \\
\mathit{task(Name)} &\overset{\approx \text{task}}{\longleftrightarrow} \mathit{do(f)} \\
\mathit{resource(Name)} &\overset{\approx \text{resource}}{\longleftrightarrow} \mathit{use(var)} \mid \mathit{provide(var)} \\
\mathit{softgoal(Name)} &\overset{\approx \text{softgoal}}{\longleftrightarrow} \mathit{optimize(f)} \\
\mathit{actor(Name)} &\overset{\approx \text{actor}}{\longleftrightarrow} \mathit{var} \\
\mathit{cgmodel} &\overset{def}{=} \mathit{goal(Name)} \mid \mathit{task(Name)} \mid \mathit{resource(Name)} \\
&\quad \mid \mathit{softgoal(Name)} \mid \mathit{actor(Name)} \\
\mathit{cgmtype} &\overset{def}{=} \mathit{goal} \mid \mathit{task} \mid \mathit{resource} \mid \mathit{softgoal} \\
\mathit{ckeyword} &\overset{def}{=} \mathit{achieve(f)} \mid \mathit{maintain(f)} \mid \mathit{achieve\&maintain(f)} \mid \mathit{avoid(f)} \mid \mathit{do(f)} \\
&\quad \mid \mathit{use(var)} \mid \mathit{provide(var)} \mid \mathit{optimize(f)}
\end{aligned}
$$

Fig. 7. GAM/Tropos translation rules.

$$
\begin{aligned}
\mathit{contribution[+](cgmodel_1,\ cgmodel_2)} &\overset{\approx \text{contribute[+]}}{\longleftrightarrow} (\mathit{cgmodel_1} \overset{\approx \text{cgmtype}}{\longleftrightarrow} \mathit{ckeyword_1} \\
&\quad \wedge \mathit{cgmodel_2} \overset{\approx \text{cgmtype}}{\longleftrightarrow} \mathit{ckeyword_2} \\
&\quad \wedge \mathit{ckeyword_2} \rightsquigarrow \mathit{ckeyword_1}) \\
\mathit{contribution[-](cgmodel_1,\ cgmodel_2)} &\overset{\approx \text{contribute[-]}}{\longleftrightarrow} (\mathit{cgmodel_1} \overset{\approx \text{cgmtype}}{\longleftrightarrow} \mathit{ckeyword_1} \\
&\quad \wedge \mathit{cgmodel_2} \overset{\approx \text{cgmtype}}{\longleftrightarrow} \mathit{ckeyword_2} \\
&\quad \wedge \exists \langle P_1, c_1 \rangle, \langle P_2, c_2 \rangle \text{ such that } c_1 \text{ is the formula in } \mathit{ckeyword_1} \\
&\quad \text{and } c_2 \text{ is the formula in } \mathit{ckeyword_2} \\
&\quad \text{and } \langle P_2, c_2 \rangle \gg^{c} \langle P_1, c_1 \rangle) \\
\mathit{task\text{-}decomposition(task(Name),\ cgmodel)} &\overset{\approx \text{task-decomp}}{\longleftrightarrow} (\mathit{cgmodel} \overset{\approx \text{cgmtype}}{\longleftrightarrow} \mathit{ckeyword} \wedge \mathit{task(Name)} \overset{\approx \text{task}}{\longleftrightarrow} \mathit{do(f)} \\
&\quad \wedge \mathit{do(f)} \rightsquigarrow \mathit{ckeyword}) \\
\mathit{means\text{-}ends(goal(Name),\ cgmodel)} &\overset{\approx \text{means-ends}}{\longleftrightarrow} (\mathit{cgmodel} \overset{\approx \text{cgmtype}}{\longleftrightarrow} \mathit{ckeyword} \\
&\quad \wedge (\mathit{goal(Name)} \overset{\approx \text{goal}}{\longleftrightarrow} \mathit{achieve(f)} \mid \ldots \mid \mathit{avoid(f)}) \\
&\quad \wedge (\mathit{achieve(f)} \mid \ldots \mid \mathit{avoid(f)} \rightsquigarrow \mathit{ckeyword})) \\
\mathit{dependency(mel_1,\ cgmodel,\ mel_2)} &\overset{\approx \text{dependency}}{\longleftrightarrow} (\mathit{cgmodel} \overset{\approx \text{cgmtype}}{\longleftrightarrow} \mathit{ckeyword} \\
&\quad \wedge (\forall i = 1, 2, \mathit{mel_i} = (\mathit{cgmodel_{i,1}}, \ldots, \mathit{cgmodel_{i,r}}) \\
&\quad \wedge \forall 1 \leq k \leq r, r > 0, \mathit{cgmodel_{i,k}} \overset{\approx \text{cgmtype}}{\longleftrightarrow} \mathit{ckeyword_{i,k}} \\
&\quad \text{with } \mathit{cgmodel_{i,1}} \overset{\approx \text{actor}}{\longleftrightarrow} \mathit{var_i} \\
&\quad \wedge \forall i, \mathit{dep_i} = \bigwedge_m \mathit{ckeyword_{i,m}}, 2 \leq m \leq r) \\
&\quad (\mathit{ckeyword} \wedge \mathit{dep_1}) \rightsquigarrow \mathit{depend(var_1,\ ckeyword,\ var_2)} \\
&\quad \wedge \mathit{depend(var_1,\ ckeyword,\ var_2)} \rightsquigarrow \mathit{dep_2})
\end{aligned}
$$

Fig. 8. Continued from Fig.7: GAM/Tropos translation rules.

3. When the tree has been constructed to a satisfactory extent by recursive application of steps 1) and 2) above, label the leaves of the tree *undefeated* (U). For any inner node, label it undefeated if and only if every child of that node is a *defeated* (D) node. An inner node will be a defeated node if and only if it has at least one U node as a child. Do step 4) below after the entire dialectical tree is labeled.
4. $\langle P, c \rangle$ is a *justification* (or, P justifies c) if and only if the node $\langle P, c \rangle$ is labelled U.

Technique 16. (Justification in the GAM decision process) The GAM decision process is combined with the justification process in the following way. First, when *problem analysis* leads to the identification of a set of alternatives, a dialectical tree is built for each alternative, whereby the root of the dialectical tree

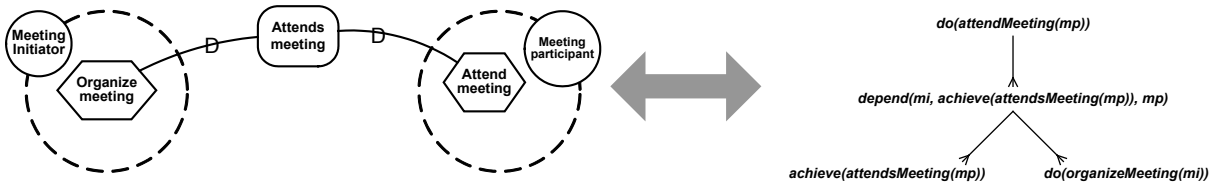


Fig. 9. Example illustrating the use of the GAM/Tropos transition rule for dependencies.

Element in the Tropos goal diagram	*Intermediary language*	*Labeled well-formed formula in a dialectical tree*
Schedule meeting	*goal(Schedule meeting)*	*achieve(schedule_meeting(ms))*
Low effort	*softgoal(Low effort)*	*optimize(schedule_meeting(ms))*
Ask for confirmation via popup	*task(Ask for confirmation via popup)*	*do(ask_for_confirmation(ms))*
User agenda data	*resource(User agenda data)*	*provide(user_data_agenda(email_client))*
Meeting Particip.	*actor(Meeting Participant)*	*meeting_participant* or **ms** *(a variable, not a wff)*
Grab dates without user intervention (+) User friendly	*contribution[+](task(Grab dates withour user intervention), softgoal(user friendly))*	*do(grab_dates_automatically(ms))* ↑ *optimize(user_friendly(ms))*
Grab dates without user intervention (-) User privacy	*contribution[-](task(Grab dates withour user intervention), softgoal(user privacy))*	*do(grab_dates_automatically(ms))* ↑ *¬optimize(user_privacy(ms))*
Grab dates without user intervention (I) User agenda data	*task-decomposition(task(Grab dates withour user intervention), resource(User agenda data))*	*provide(user_data_agenda(email_client))* ↑ *do(grab_dates_automatically(ms))*
Ask for confirmation via popup → Confirm potential meeting date before sending	*means-ends(task(Ask for confirmation via popup), goal(Confirm potential meeting date before sending))*	*do(ask_confirmation_via_popup(ms))* ↑ *achieve(confirm_before_sending(mi))*
Meeting Particip. (D) mel1; Schedule meeting (D) mel2; Meeting Sched.	*dependency(mel1, goal(schedule_meeting), mel2)* (see GAM/Tropos translation rules for meaning of **mel1** and **mel2**)	(see transl. rules) ↑ *depend(mp, achieve(schedule_meeting(ms)), ms)* ↑ *achieve(schedule_meeting(ms))* (see transl. rules)

Fig. 10. Exemplified GAM/Tropos translation rules.

is the argument tree of the given alternative. The *evaluation* decision activity consists of labeling each dialectical tree, and accepting the one justified alternative, this being the alternative whose dialectical tree is such that the root node is labeled *undefeated* U. *At most one alternative must remain justified*—in case more than one alternative appear justified, additional arguments need to be added as leaf nodes to each alternative's dialectical tree until only one alternative remains justified. The *decision* decision activity amounts to choosing the justified alternative, and acting upon it in terms of changing the associated goal diagram.

Example 8.27. For illustration, Fig.6 shows the dialectical tree for *3.Alt1 "Schedule meeting using email only"*. To simplify the representation, the dialectical tree is built with formulae. The justification process showed that the alternative is unjustified and therefore cannot be accepted. Argument trees for *scheduleByEmail(mi)* and *complicatedScheduling(ms)* are shown. The argument $\langle\{\ldots\}, complicatedScheduling(ms)\rangle$ defeats $\langle\{\ldots\}, scheduleByEmail(mi)\rangle$ at $\neg userFriendly(ms)$ with to the subargument:

$$\langle\{\ manualScheduling(ms) \wedge manyEmailsToRead(mi) \rightsquigarrow \neg userFriendly(ms)\ \}, \neg userFriendly(ms)\ \rangle$$

of $\langle\{\ldots\}, complicatedScheduling(ms)\rangle$.
Observe that the given dialectical tree suggests alternative 3.Alt1 should to be rejected as it does not satisfy user friendliness. It appears, however, that the same alternative satisfies low implementation cost, for

Fig. 11. Translation of the Tropos SR diagram in Fig. 2 using GAM/Tropos translation rules.

there are no arguments that defeat *lowImplementationCost(ms)*. Rejecting the alternative assumes that user friendliness is preferred by the decision makers to low implementation cost—preferences orderings ought to be specified outside GAM, i.e., within the goal diagram constructed using the RE framework that GAM complements. Integrating preferences in justification is a matter that requires a separate treatment (for an introduction, see, e.g., [157]).

8.6.1 Analyzing Arguments in Existing Goal Diagrams

The definitions given earlier can be applied to analyze goal diagrams constructed without using the GAM decision process, thus allowing GAM to be applied in a relatively straightforward manner to study the arguments implicit behind modeling decisions of which only the solution statements are available to the requirements engineer.

This is realized by defining a mapping for translation between a goal diagram and a dialectical tree. In the present paper, only the rules for translating between a Tropos Goal Diagram (TGD) and a dialectical tree (DT) are suggested. Similar rules can be defined when using GAM with other goal-oriented RE frameworks. The DT can thus be submitted to Tropos-specific analyses, provided it is translated into a TGD; and a TGD can be subjected to the analysis of the justification behind the modeling decisions that led to it.

Mapping is defined using an intermediary language and a set of keywords illustrated with the meeting scheduler case study examples in Fig.10. The translation rules are bidirectional. When translating from

a TGD to a DT (moving from left to right in Fig.10) the intermediary language is used to write down th structure of the TGD. The obtained TGD specification is then translated into formulae labeled with a restricted set of keywords. The rules used for translating between the intermediary language and the DT are referred to as the "GAM/Tropos translation rules" and are formalized as follows. The operator "$\overset{\approx \text{label}}{\longleftrightarrow}$" denotes translation rules, with *label* indicating the name of the rule being used in the translation (f stands for "formula")—definitions are given in Figures 7 and 8.

Observe that the relationships in the goal model are interpreted as defeasible rules. Intuitively, this seems adequate: e.g., if a task is decomposed into a resource in a TGD, the need to provide a resource can be interpreted to exist because that resource is used when executing the given task; in a negative contribution, the link in the TGD is directed from an element that contributes negatively to the target softgoal, whereas in a DT, a negative contribution exists between a defeater argument and the argument it defeats. The dependency relationship is interpreted as a chain of two defeasible rules (i.e., $\alpha \rightsquigarrow \beta \wedge \beta \rightsquigarrow \chi$). In the first, β marks the dependency between actors and α is the dependum of the dependency (because, e.g., in a goal dependency, the dependum goal is the reason for the dependency to exist: the depender cannot achieve the goal without the dependee). In addition, α can contain one or more *ckeyword* to indicate why the depender alone is unable to obtain the dependum (this is required if a Tropos SR is being translated, whereas it is often unknown in a Tropos SD). In $\beta \rightsquigarrow \chi$, χ expresses goals / tasks / ... softgoals that the dependee is expected to, respectively achieve / ... / optimize in order to assist the depender in obtaining the dependum (χ is often unknown in a Tropos SD, whereas it is available in a Tropos SR). The use of the tranisition rule for dependencies is illustrated in Fig.9.

To translate a DT to a TGD, the formulae appearing in the DT are transformed into *labeled* formulae. Labels are used to derive a goal diagram element from a formula (i.e., a label maps to one or more concepts in the ontology of the goal model). For formulae that are to be translated into goals in a TGD, the Tropos goal taxonomy [100] is employed, giving four labels: *achieve(f)*, *maintain(f)*, *achieve&maintain(f)*, and *avoid(f)*. For formulae that will result in resources, the label *provide(f)* is used when a resource is being provided by an actor, whereas the label *use(f)* is applied when the resource is to be used by an actor. Figures 7, 10, and 8 give other labels along with their corresponding TGD representation.

Example 8.28. The arguments in Fig.11 have been obtained by applying the GAM/Tropos translation rules to the Tropos SR in Fig.2. Using the obtained dialectical tree (shown in its developed form, that is, with detail of its component arguments, in Fig.11), the stakeholders can, e.g., question modeling choices by providing new arguments that defeat existing ones and lead to changes in the tree. For example, it can be observed that there is cyclical argumentation in two cases: *do(agreeToDate(mp))* $\rightsquigarrow$ *depend(mp, provide(proposedDate(ms)), ms)*, and *do(scheduleMeeting(ms))* $\rightsquigarrow$ *depend(ms, do(enterDateRange(mi)), mi)*. As this is undersirable, the stakeholders may consider adding defeasible rules so that either the antecedent or the consequent is defeated or replaced. The same applies to arguments that some stakeholders may consider inappropriate—in response, they could add defeating arguments in order to request a change in the goal diagram.

While the discussion and the translation rules presented in this subsection remain Tropos-specific, the intention is not to provide within this paper a library of translation rules between GAM and various RE frameworks' goal models. Instead, the above illustrates how mappings can be defined and used, and therefore constitute exemplified guidelines for requirements engineers aiming to use GAM with their favorite framework. Moreover, while the translation rules suggested in Fig.7 seem to fit intuition, we cannot suggest that the given rules are definite and cannot be adjusted as the requirements engineer deems appropriate.

8.7 Related Work

The need for justifications of modeling choices has not been overlooked in various RE methods that employ goal models. In particular, high-level goals in a diagram can be understood as reasons for representing lower-level goals (i.e., the need for the latter is justified by the presence of the former) and any other element in a goal diagram. Informal and formal AND/OR refinement and decomposition techniques, widespread in RE (for an overview, see [315]), can therefore be seen as incorporating argumentation and justification, in that sub-goals could be understood as arguments supporting parent goals. A refinement alternative would then be justified if there are no conflicts between sub-goals (i.e., it is consistent), as few obstacles as possible harm sub-goal achievement, there are no superfluous sub-goals (the refinement is minimal), and the achievement of sub-goals can be verified to lead to achieving the parent goal (if

refinement is formal [72]). While this interpretation may seem satisfactory, argumentation and justification processes differ from and are complementary to refinement in several respects:

1. Regardless of clarification, argumentation anterior to modeling uses and produces richer information than is contained in a refinement recorded in a goal diagram, due to the relatively strict syntax and semantics of the underlying goal models. To leave such information unrecorded is likely to lead to issues 1–3 highlighted in §8.1 and not analyzing it would lead to 4–5. This has been argued and illustrated throughout the paper.
2. Historically, weak refinement links have been proposed in the NFR goal model [225], which has been inspired in part by work in informal design rationale approaches that involve argumentation (e.g., [193]). One result has been the integration of a class of argumentation goals to justify modeling choices. Any argumentation goal in NFR is of the sort "claim" (with subsorts "FormalClaim" and "InformalClaim") and represents "evidence or counter-evidence for other goals or goal refinements" [225]: taking the example from [225], the following is an argumentation goal
   $InformalClaim$["Rigorous examination is recommended for publication by employees"]
   which supports this argumentation goal:
   $FormalClaim[\exists e : ValidatedBy[e, attributes(Employee)] \wedge EmpStatus(e, SecI)]$
   The formalism above states that class I secretaries ($SecI$) review employee data. Notice that an argumentation goal is not by itself an argument, since there is no conclusion indicator and only one piece of information is given (i.e., one does not know if the content of the argumentation goal is a premise or a conclusion). In GAM, the above two argumentation goals are written as the following argument:
   $\langle\{$ ["Rigorous examination is recommended for publication by employees"] $\rightsquigarrow [\exists e : ValidatedBy[e, attributes(Employee)] \wedge EmpStatus(e, SecI)]\}, [\exists e : ValidatedBy[e, attributes(Employee)] \wedge EmpStatus(e, SecI)] \rangle$
   Argumentation goals produced using NFR can thus be reused in GAM. As a rigorous argumentation and justification process employing argumentation goals has not been proposed, GAM can be usefully combined with NFR to provide a rigorous argumentation and justification process. Another difficulty with weak refinements may be that semantics of weak refinement links (called contribution links) between goals remain, according to [198] "too vague for deep, accurate understanding of the [softgoal] model", leading to recent critiques in [198] on their meaning and subsequent applicability when rigorous qualitative analysis is required.
3. Formal goal refinement has clear, but limited semantics: It remains difficult to compare alternative refinements in a rigorous qualitative way (thorough quantitative comparison is possible using probabilistic measures of goal satisfaction [198], although it is applicable to already clear requirements). It has been suggested to use a temporal logic [197] to compare alternative refinements for e.g., minimality, conflict, and obstacles. Such techniques are RE method-specific and applicable when goals are already clear, leading [315] to argue that systematic approaches for alternative comparison are still not available.

A different approach, explored in this paper, is to allow qualitative and quantitative information by constructing and analyzing arguments. One novelty of GAM is thus to combine a design rationale approach to organize commonsense reasoning with an argumentation model to document the arguments leading to a modeling decision and allow structured justification to take place, while allowing informal and formal use with any goal model. As arguments accept both qualitative and quantitative information, the importance of quantitative evidence can be acknowledged, along with qualitative, subjective, and defeasible information.

Regarding design rationale and argumentation modeling literature, GAM adopts and adapts existing approaches, adjusting them for combined application in RE activities. In this respect, an important characteristic of this work is its integrative aim and its appropriateness to RE. As argued above, the GAM decision process is not a novelty in itself, and maps nicely to existing design rationale approaches; however, it is adapted to the purpose to which it has been put in the present paper.

Apart from the contribution of the integrative approach to justification in relation to goal models, another contribution is the identification of the clarity facets. While a considerable amount of work on, e.g., vagueness and ambiguity exists in various fields (as noted in §8.5), its interpretation in the context of RE has not been proposed, and an extensive treatment has not been realized. Although the issues raised in §8.5 are pressing for RE, in that the reader will undoubtedly agree that initial requirements tend to be, among others, ambiguous and vague, the research in RE comparable to the one here is limited. Ambiguities arising from the presence of coordination conjunctions *and*, *or*, and *and/or* in natural

language requirements have been studied [52]. Text appearing in instances of modeling primitives in goal models normally does not contain coordination conjunctions since these are avoided with goal modeling constructs such as AND/OR refinement or decomposition (see, e.g., [71, 332, 50]). The approach suggested in [52] can be combined with GAM: while GAM focuses on goal diagram content, the said approach can be applied before goal modeling, on information represented in textual documentation from which the engineer is expected to derive goal diagram elements. Ambiguity of natural language requirements has been discussed and techniques suggested for detecting ambiguity without discussing clarification in [161]. Their approach does not consider ambiguity detectable by structural analysis. Others [25] have concentrated on ambiguity that may arise from the use of plural in natural language requirements. Fuzzy logic has been suggested for dealing with vague requirements (e.g., [203] among others—a more elaborate discussion is given in [151]). However, §8.5 clearly illustrates that vagueness is merely one among many facets. Somewhat close to our effort here is [256], where three dimensions along which any RE project or framework can be described, are identified. It is suggested there that the overall aim in any RE project is to move from informal to formal representations of requirements (along the *representation* dimension), from individual to shared views (the *agreement* dimension), and from an opaque understanding of the system to its complete specification (the *specification* dimension). It may be interesting in such a framework to perceive clarity facets as refining the three dimensions with a number of dimensions, and in particular, enriching the specification dimension by disaggregating it onto many dimensions that we called clarity facets.

8.8 Discussion

In real world RE projects, GAM can be applied to guide the construction, questioning, and critique of (fragments of) a goal diagram in three ways:

1. When only the GAM decision process (§8.4) is put to use, clarification and argumentation are unstructured and left to intuition of the requirements engineer and of the stakeholders. Although the discussions of clarification (§8.5) and argumentation (§8.6) illustrate that neither of these is trivial, using the GAM decision process alone helps in organizing the goal modeling activity. In addition to being grounded in established results in design rationale research, the decision process is intuitive: adding a new fragment or revising an existing part of the goal diagram is normally due to particular reasons (described in the problem statement); there are often alternative ways of changing the diagram to solve the stated problem (the alternatives); each alternative has its merits and drawbacks (i.e., arguments for and against each alternative), whereby merits and drawbacks need to be compared to select one alternative over others (the evaluation activity); finally, a decision is reached (decision statement) and the goal diagram is modified accordingly. As highlighted at the outset of the paper, this or a very similar approach is usually implicitly performed by the engineer. It is when the system to engineer is of non-trivial complexity and when stakeholders participate actively in modeling that it becomes useful to have a structured and shared approach to goal modeling. Having a sequence of clearly identifiable steps allows each participant to know how others take part in the modeling activity and how to intervene to question others' modeling decisions.
   Applying a rationale approach to structure the design process (regardless of whether the rationale is recorded or not) is not trivial. Fitting the thinking involved in the design activity to a framework can be perceived as unnatural and thus involves further effort from the participants [64, 286]. In GAM, this issue is tempered by using a small set of imposed concepts and activities, thus facilitating the learning of the decision process.
   Whether the information produced in the various activities of the GAM decision process is recorded is a project-specific choice. Recording gives rise to the following difficulties:
   a) As already observed in design rationale research (e.g., [205]), recording rationale requires additional effort from the engineer and the stakeholders participating in the design activity.
   b) It is noted in literature on design rationale that the artifacts containing records of design rationale tend to become rapidly complex and thus difficult to use.
   c) Allowing stakeholders to modify records of design rationale requires training.

   Issue (1a) above is tempered in GAM by using a small set of intuitively acceptable concepts and techniques, as in the reasoning loop model [205] on which the GAM decision process draws. In addition, the visual syntax of the decision diagram remains flexible so that various available tools can be used (e.g., gIBIS [64], Euclid [296], Belvedere [51], Compendium [63], Hermes [163]). Annotations of the decision and goal diagrams also allow the use of simpler tools, such as common word processors to record

the decision process. When applying the GAM decision process to guide and record the details of the rationale invested in building a goal diagram, elaborate decision diagrams (see, issue (1b) above) are obtained even for goal diagrams of limited size (see, e.g., Fig.4). Again, annotations provide a simple means to relate fragments of decision and goal diagrams to focus questioning and revision on such fragments only. Regarding issue (1c), training is necessary both if meetings are organized to collaboratively apply GAM and if distributed acquisition of information for the decision process is allowed. Distributed acquisition supported by simple tools was sufficient for the case studies: for instance, a web log (i.e., blog) platform (in which each post and comment is annotated as in the decision and goal diagrams) is not to underestimate for recording and making available to participants the content of the decision diagram at all times, with the engineer updating the goal diagram using a standard diagramming solution and posting the updated versions on the blog. Keeping the participants posted of new additions to the blog by automatically generated email messages proved a simple and useful solution, avoiding the need for learning a new tool. This approach, however, encountered difficulties in meetings (more suitable for brainstorming than distributed acquisition) as the engineer is expected to act as the facilitator and record changes to the goal diagram.

2. A second way to apply GAM is to combine the GAM decision process (§8.4) with the clarification (§8.5) and argumentation (§8.6) techniques.
   The engineer applies clarification techniques on information recorded in the design rationale. Candidate information for clarification has one or more of the following characteristics: (i) the engineer does not understand the information; (ii) a stakeholder indicates a difficulty in understanding the information; and (iii) the information gave rise to arguments and counterarguments in which it appears that the confrontation results from lack of shared understanding of the given information. Clarification requires the writing and updating of the thesaurus in order to enforce clarification choices throughout the goal diagram and accompanying documentation. In the case studies, a cross-referenced document was created and maintained with a word processor, while markup and cross-referencing was performed with Atlas.ti [13], which allows markup and cross-referencing of both text and graphical (including diagrams) artifacts. A more advanced tool would automatically verify internal thesaurus consistency and automate the enforcement of clarification decisions in associated artifacts (i.e., decision and goal diagram). It remains for future work. In practice, clarification techniques proved particularly useful in raising awareness of clarity issues in stakeholders' statements and various documentation used in the case studies. This lead the participants to rephrase pieces of information in the goal and decision diagrams and the thesaurus so as to check whether they understand them appropriately. When rephrased information appeared different than the original, clarification techniques were applied to check what clarity issue is present and subsequently resolve it. Clarification is time consuming: one potential direction for improvement lies in linking the thesaurus to lexicons such as, e.g., WordNet [87] in order to automatically generate lists of potential synonyms and facilitate the writing of definitions, which is useful in dealing with overgenerality.
   Using the argumentation techniques does not necessarily require the formalization of argument or goal diagram content (see point (3) below). Applying techniques such as the argumentation of an alternative and justification already render more rigorous the Evaluation activity in the GAM decision process. Such rigor is needed to avoid issues which stem from unstructured argumentation and justification, and described in Ex.8.17 (e.g., the limited justification of alternatives). It has been empirically observed that nonmonotonic reasoning is hard for humans [94]. Effort involved in finding arguments in GAM is considerable, which is in line the cited empirical result. Some techniques derived from theory are particularly hard to apply in practice: for instance, comparing arguments for specificity appeared counterintuitive and was thus seldom used. The suggested informal orderings based on clarity of arguments appeared, however, practical for rejecting unclear arguments in favor of clear ones. Because these orderings are derived from clarification techniques, clarifying arguments was combined to argument comparison over clarity. To help stakeholders' in their search for arguments, it is useful to question why they believe in the information they provide (thus expecting answers in which they give reasons for what they suggest). Prior experience and resources about the debated domain are relevant sources of arguments, so that referring to these is suggested. Although the difficulties are considerable when applying argumentation and justification, a significant benefit is that these techniques lead to the externalization of information usually left implicit in goal modeling. Abstract entities such as goals of the goal diagram are thus associated with more precise information that led the participants to introduce the given goal in the first place, and that can subsequently be used when operationalizing goals. Moreover, the information made explicit is available to a number of stakeholders who can, through argumentation and justification, question and revise the goal modeling

decisions. Lessons can be learned from past modeling problems as sources of these issues (such as, e.g., fallacious argumentation) can be identified by going back to the recorded design rationale and the arguments therein.

3. A third, most rigorous and resource intensive way of applying GAM is to formalize the content of arguments. If the engineer intends to arrive at a formal specification of the goal diagram, it is useful to formalize premises and conclusions using the formal logic of the chosen goal modeling framework: for instance, in Tropos, a temporal first order logic is used, so that the premises and conclusions would be written in that logic, while any automated detection of argument relationships would be performed using the theory of the argumentation framework (§8.6). However, as there is normally much more information describing the design rationale than is in the product of the design activity, formalizing the former because the latter needs to be formalized may be difficult to justify in terms of cost and benefits. When GAM is applied to an already elaborate goal diagram (not necessarily constructed with GAM) which is accompanied by a formal specification, argument formalization is facilitated by reusing fragments of the specification. The choice of formalizing depends also on the characteristics of the system: formalizing arguments and justifications of requirements for a safety-critical system can prove relevant, as it is likely that using formal instead of natural language would give rise to less clarity issues. The choice of formalizing or not remains a difficult one (some general suggestions can be found in [35]) in particular since current support for automated analysis of formalized arguments is limited. This, however, does not limit the benefit of introducing the argumentation and justification activities in GAM by using a formal notation, as it simplified the discussions and helped make the presentation precise.

8.9 Conclusions and Future Work

Goal modeling unavoidably involves the transformation of, often unclear requirements expressed by stakeholders, inexperienced in RE goal modeling techniques, into more precise information written often in an instance of a goal model. Difficulty further arises when many stakeholders with different backgrounds participate in the engineering of requirements.

The suggested Goal Argumentation Method (GAM) addresses these issues by advancing the available research in the following ways:

1. Techniques for identifying and resolving problems of clarity in expectations expressed by the stakeholders are suggested. The issue of clarity is identified and treated in GAM as one of central problems that appear when dealing with initial statements of requirements. Lack of clarity is treated in a systematic manner, relying on precise and operational definitions of the clarity facets, along with specialized techniques for clarifying information to arrive at more elaborate understanding of stakeholders' expectations.
2. Argumentation of modeling choices is considered critical in obtaining justifiably appropriate instances of requirements models. Argumentation and justification allow straightforward, yet rigorous discussion, revision, and settling on the goal modeling choices.
3. Combination of clarity facets and argumentation allowed the definition of novel ordering relations based on the analysis of argument content to prefer non-ambiguous, -overgeneral, -synonymous, and -vague arguments to other.

The proposed framework advances the state of the art both on the argumentation and clarification issues. While GAM is method-independent, aiming for complementarity to established and potentially future RE methods, its use has been illustrated in combination with the Tropos goal model. The meeting scheduler and air-traffic management case studies served for illustration.

8.9.1 Future Work

Both argumentation and clarification can be refined and studied further than has been presented in this paper. For instance, justification could be partly automated when GAM is employed with formal RE methods, such as KAOS [71, 197], where arguments could be constructed automatically from already formalized requirements. In this respect, automated translation between goal diagrams and argument trees would allow novel automated checking of goal diagrams.

One important direction of future effort is the extension of GAM beyond goal models, and the use of argumentation and clarification with standardized modeling languages, such as the UML. While GAM

has been initially developed and presented herein with goal modeling in mind, it is difficult to accept that UML diagrams are necessarily well argued for or always contain clear information.

Finally, clarification can be further studied. In its current form, a formalism for requirements clarification is unavailable. As mentioned above, arriving at such a formalism is difficult, as it requires the integration of already well discussed, but also still debated topics in logics and AI, such as the formalization of vagueness and generality. Any specification language that would claim to allow the formal representation and reasoning about clarity facets would need to be based on a well-structured formal system, while remaining usable. Arriving at such a formal system would be a significant contribution, for it would take RE a step closer to increased automation of requirements acquisition, whereby such automated requirements assistants would interactively clarify initial statements of requirements. Following the discussion in §8.5, such a formalism would allow novel precise and structured representation and reasoning about quality and nonfunctional requirements which tend to involve vague and ambiguous information, an open and pressing issue in requirements and software engineering during the last four decades.

Part III

Applying Conceptual Foundations and Techniques to the Modeling and Analysis of Requirements

9

Outline of Part III

Spread over four chapters, Part III of the thesis shows the use of the theoretical contributions from Parts I and II to address specific issues in IS engineering. Each of these chapters adds specific contributions to the respective communities in which they are presented.

In Chapter 10, we show that principles underlying GAM (introduced in Chapter 5) apply to the design of classical enterprise IS, and in particular enable a new approach to the traceability problem. Neglecting traceability — i.e., the ability to describe and follow the life of a requirement — is known to entail misunderstanding and miscommunication, leading to the engineering of poor quality systems. Following the simple principles that (a) changes to instances of requirements models obtained with the Unified Modeling Language (UML) ought be justified to the stakeholders, (b) justification should proceed in a structured manner to ensure rigor in discussions, critique, and revisions of model instances, and (c) the concept of argument instantiated in a justification process ought to be well defined and understood, we used the underlying principles in GAM to enable the traceability of design rationale in UML while allowing the appropriateness of model changes to be checked by analysis of the structure of the arguments provided to justify such changes.

In Chapter 11, we adapt GAM and its underlying principles to the engineering of requirements for Adaptable and Open Service-oriented Systems (AOSS). Such systems pose novel problems in decision-making during RE. It is not feasible to specify requirements for AOSS by applying the commonly accepted generic approach in RE, which consists of moving from the elicitation, representation, and analysis of abstract stakeholder expectations towards a detailed specification of the entire system's behavior before deploying the system. Openness and adaptability allow new services to appear at run-time so that ways in, and degrees to which the initial stakeholders' requirements will be satisfied may vary at runtime. The initial requirements specification produced at development time may therefore lose relevance at runtime in two respects. First, the initial specification may be overly constraining, limiting adaptability by restricting the set of potential services that may participate at run-time. Second, the initial specification will not reflect the runtime variation in how and to what extent the stakeholders' requirements are satisfied. To maintain the initial requirements specification relevant at AOSS runtime, it is necessary to (1) determine how extensive the initial specification, produced at development-time, ought to be, (2) what parts thereof are to be updated at runtime to reflect system adaptation, and (3) how to perform such updates. We propose [154] a solution to these issues. Depending on the frequency of updates, we separate the requirements engineering (RE) of AOSS onto the RE for: individual services (Service RE), service coordination mechanisms (Coordination RE), and quality parameters and constraints guiding service composition (Client RE). To assist existing RE methodologies in dealing with Client RE, the Dynamic Requirements Adaptation Method (DRAM) is proposed. DRAM updates a requirements specification at runtime to reflect change due to adaptability and openness.

The discussion of Chapter 12 is situated in the field of service-oriented computing. We consider the problem of selecting among competing services those that are capable of satisfying functional and non-functional (i.e., quality) requirements. We combine a simple architecture with a novel algorithm to enable openness, distribution, and multi-criteria-driven service composition at runtime. The service-oriented architecture involves mediator web services coordinating other web services into compositions necessary to fulfil user requests. By basing mediator services' behavior on a novel multicriteria-driven (including quality of service, deadline, reputation, cost, and user preferences) reinforcement learning algorithm, which integrates the exploitation of acquired knowledge with optimal, undirected, continual exploration, we ensure that the system is responsive to changes in the availability of web services. The reported experiments indicate the algorithm behaves as expected and outperforms two standard approaches. The input of the

algorithm are service requests that indicate stakeholders' functional and nonfunctional requirements. We use the terminology outlined in Part 1 of the thesis as the foundation for the structure of the service requests.

The discussion in Chapter 13 is situated in the field of agent-oriented computing. In particular, we are concerned with the engineering of multi-agent systems (MAS), which involves heterogenous agents designed by, and distributed across various providers. Norms (i.e., obligations, prohibitions, and permissions), sanctions, and incentives are enforced within the norm-governed MAS to ensure that agents act and interact only in ways that satisfy stakeholders' requirements. Based on the terminological framework introduced in Part 1 of the thesis, we propose a framework, called Requirements-driven Contracting (RdC) for deriving executable norms, sanctions, and incentives from the requirements and associated relevant information. RdC allows the use of RE concepts and methods to govern the norm-governed MAS. RdC ensures that all requirements, along with runtime changes of requirements are reflected in the norms, sanctions, and incentives regulating the behavior of agents in an open MAS.

10

Tracing the Rationale behind UML Model Change through Argumentation

Abstract Neglecting traceability—i.e., the ability to describe and follow the life of a requirement—is known to entail misunderstanding and miscommunication, leading to the engineering of poor quality systems. Following the simple principles that (a) changes to UML model instances ought be justified to the stakeholders, (b) justification should proceed in a structured manner to ensure rigor in discussions, critique, and revisions of model instances, and (c) the concept of argument instantiated in a justification process ought to be well defined and understood, the present paper introduces the UML Traceability through Argumentation Method (UML-TAM) to enable the traceability of design rationale in UML while allowing the appropriateness of model changes to be checked by analysis of the structure of the arguments provided to justify such changes.

10.1 Introduction

In a noted discussion of the traceability problem [109], Gotel and Finkelstein define traceability as follows:

> "Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)."

Ensuring proper traceability through specialized concepts, techniques, and methods is argued to reduce the number of iterations in the construction and change of requirements engineering (RE) artifacts, thus helping keep the software development project under time, budget, and other constraints. However, if traceability is neglected, misunderstanding and miscommunication are bound to appear, compounding the loss of implicit information guiding requirements change and increasing the risk of poor project results [75, 257, 265].

This paper focuses on the problem of tracing the rationale behind changes local to one or spanning across several different kinds of models in the Unified Modeling Language (UML) [112]. To address the problem, the UML Traceability through Argumentation Method (UML-TAM) is suggested to enable the traceability of design rationale in UML while allowing the appropriateness of model changes to be checked by analysis of the structure of the arguments provided to justify such changes.

As the related research efforts are numerous, the following section (§10.2) first positions the present work within the relevant literature. The problem of interest is then identified and contributions outlined (§10.3), and is followed by a description of the case study (§10.4). The conceptual basis of UML-TAM is then presented (§10.5). It is followed by an illustration of its use in the case study (§10.6). The paper closes with conclusions and indications on directions for future effort (§10.7).

10.2 Background and Related Work

Complexity of the traceability problem, its span over the various activities in software development, along with the trade-off between extensive traceability and budget and time constraints make elusive the construction of an encompassing traceability approach still applicable to realistic settings—methods specialized for particular traceability sub-problems, combined with domain-specific expertise on when and how to apply them in a given project seem to be the choice in research and industry. In light

of the various methods suggested in related research efforts, situating the results of the present paper is facilitated by classification over five taxonomic dimensions: traceability data types, scope, degree of automation, conceptual foundations, and framework specificity. Each is considered in turn below.

10.2.1 Traceability Data Types

Traceability data types, as suggested by Dömges and Pohl [75], distinguish methods according to the content of traceability information being recorded:

- *Bi-directional links* between the stakeholder expectations, derived requirements, and software components enable validation of system functionality by stakeholders and impact analysis of requirements change on the system. Ramesh and colleagues [265] indicate that such benefits can be achieved, although at high initial cost of implementing and applying traceability policies. A framework allowing the capture of bi-directional links has been proposed by Pohl [258] and later extended to allow configuration to project-specific traceability needs [259], in both cases focusing on the recording of what changes are made, by whom, when, and how.
- *Contribution structures* aim at clearly relating the requirements to stakeholders to facilitate negotiation, search for additional information, and revision. Gotel [108] introduced contribution structures in RE to allow the recording of detailed information on stakeholders and the requirements they provide, hence ensuring traceability of the requirements to the people and systems from which these emanate.
- *Design rationale* records the reasoning that led to particular modeling and other software development decisions, in the aim of arriving at a shared understanding of models and other artifacts, and their purpose in the given project. Usually, a design rationale approach is employed to record such traceability information (Louridas and Loucopoulos give an overview [205]).
- *Process data* which relates to the planning and control of activities in the software development project.

10.2.2 Scope

Gotel and Finkelstein [109] introduce a separation of pre-Requirements Specification (pre-RS) from post-RS traceability. *Pre-RS*, which concerns the life of stakeholder expectations until they are converted to requirements, has been treated in the various RE frameworks proposed over the last decade—for instance, the introduction of goals in requirements models facilitates traceability, for goals make explicit (at least in part) the rationale for the inclusion of more specific requirements [315]. *Post-RS* focuses on the evolution of requirements in the steps following RE, i.e., the various activities involved in deploying the requirements. Automated traceability methods (below) focus on post-RS.

10.2.3 Degree of Automation

The degree of automation concerns the support allowed by or provided with a traceability method to reduce manual effort and facilitate analyses of trace information. Haumer and colleagues [121] and Jackson [142] both suggest manual traceability techniques focused on simplicity, while allowing rich trace recording (e.g., video, audio, etc.). Such an approach becomes difficult to manage efficiently for realistic systems, leading to, among other, Egyed's proposal [81] where models and software are aligned using traces generated by observation of software operation through the running of various test scenarios. Antoniol and colleagues [12] and Pinheiro and Goguen [253] both rely on formal methods for traceability, with the difficulty of avoiding obsolescence of formal trace specifications.

10.2.4 Conceptual foundations

Conceptual foundations discriminate according to the main concepts employed in recording traceability information (e.g., goals, scenarios, aspects). Egyed [81] generates design traceability information by iteratively running test *scenarios* on already operational software, so as to verify whether the models implementing the tested functionality correspond to the behavior of the observed system. A preliminary proposal from Naslavsky and colleagues [227] focuses on traceability between scenarios and the use thereof to relate requirements to code. Ubayashi and colleagues [312] propose a method for dealing with model evolution using model transformations based on *aspect* orientation, the main benefit thereof being the separation of concerns over traceability information. Torenzo and Castro [309] also seem to separate

concerns, albeit through specialized *views* and not aspects. In an overview of goal-oriented RE [315], Van Lamsweerde observes that the refinement links in goal refinement trees, in which an abstract goal is made more precise through refinement, can be read as traceability links making *goal*-orientation a favorable approach to aligning abstract and precise, operational information about a system. The concept of *argument* appears in design rationale approaches (for an overview, see [205]) which enable the recording of reasoning behind decisions. For instance, Ramesh and Dhar [266] suggest an approach involving concepts specialized for the RE: in addition to classical concepts—position, argument, issue—introduced in IBIS [64], REMAP [266] integrates the notion of requirement, design object, decision, and constraint.

10.2.5 Framework specificity

Framework specificity classifies approaches according to whether they are specialized or not for a particular software development framework. Briand and colleagues [42] suggest bi-directional links be extracted automatically from changes in UML models, whereby each identified type of UML model refinement (each refinement being a kind of model change) has associated traceability information, thus facilitating impact analysis in model evolution. Letelier [196] suggests a roughly defined metamodel of traceability information to collect when working with UML and requirements expressed in textual form; the aim is to ensure that bi-directional links are known during UML modeling, while very limited support is provided for design rationale recording.

10.3 Problem Outline and Contributions

The work presented in the remainder enables the recording of design rationale behind changes local to one or spanning across several different kinds of UML models. It is thus framework-specific and both pre- and post-RS (this depending on how UML is employed), while relying on the concept of argument. Because informally or formally expressed information is allowed into arguments to allow adaptability of the method to project specificities, automation is limited, this entailing selective application of the method. The present work is a response to the following observations, each highlighting a difficulty in current research:

- UML traceability rarely aims to record the rationale behind modeling decisions, and when this is attempted, as in Letelier's work [196], very limited attention is given to what kind of rationale information is to be recorded and how, and if/how it can be analyzed.
- Automated traceability by taxonomies of UML change/refinement types lacks the recording of design rationale—in the efforts cited in §10.2, traceability information answers *what* changes are made, but not *why* they are made. It is therefore possible to determine who, when, and how made a particular appropriate or inappropriate decision, but it is difficult/impossible to determine why the decision is made, hence limiting the potential to learn from mistakes or reinforce appropriate modeling behavior.
- Framework-independent traceability methods that use arguments in recording design rationale, such as REMAP [266] only provide techniques for trace capture—how to analyze such information remains unknown.

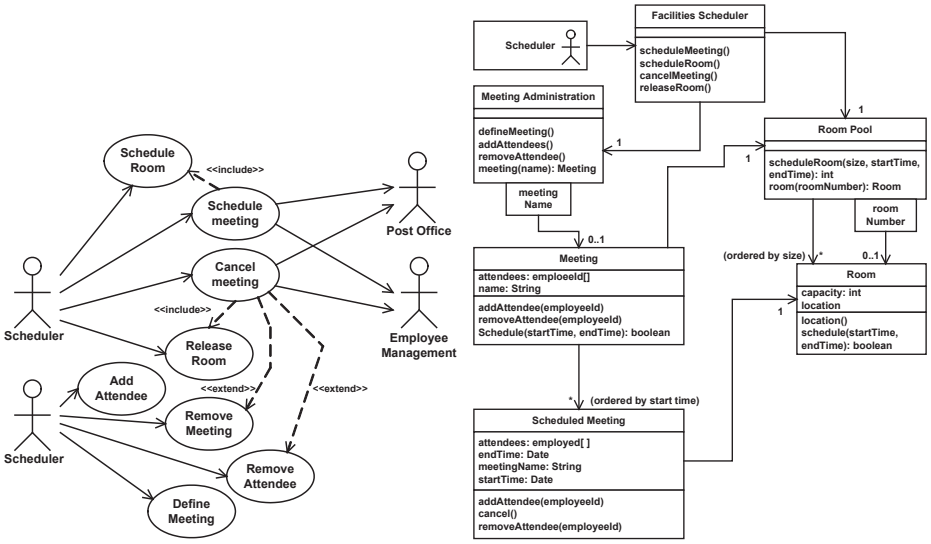
Following the simple principles that (a) changes to UML model instances ought be justified to the stakeholders, (b) justification should proceed in a structured manner to ensure rigor in discussions, critique, and revisions of model instances, and (c) the concept of argument instantiated in a justification process ought to be well defined and understood, the present paper introduces the UML Traceability through Argumentation Method (UML-TAM) for capturing and analyzing design rationale in UML modeling. The salient properties of the method are:

- *Adaptability.* Both informal and formal, and qualitative and quantitative information is allowed into arguments, to ensure that few constraints are placed on the stakeholders employing it to record design rationale.
- *Active rationale analysis.* Where available methods focus on ensuring design rationale is recorded (passive rationale traceability), UML-TAM provides specialized analyses for confronting arguments and avoiding ill-structured rationale which unavoidably leads to inappropriate modeling choices.
- *Sound conceptual foundations.* By relying on formal definitions of the concept of argument established in AI, and using it as a central concept, UML-TAM avoids ambiguity and aims to facilitate the learning of the method to the stakeholders (it merely requires the understanding of the notion of argument and the argumentation and justification processes).

- *Justification of modeling choices.* While recording arguments is certainly relevant, confronting them through a justification process to discriminate among alternative changes of model instances is critical. Justification thus provides a means for selecting among alternative sets of arguments to arrive at justifiably appropriate modeling choices.

10.4 Case Study

Following the classical meeting scheduler case study [313], a variant serves herein to illustrate the salient features of the method.[1] The aim is to design a system for scheduling meetings and meeting rooms. A user can request a meeting room of a chosen size and for a chosen period of time, and can schedule a meeting. A user can cancel any of the mentioned two until the beginning of the meeting time. An email is sent to participants any time the meeting is scheduled or canceled. When defining a meeting, the user provides a list of attendees, meeting time and room, and gives a brief description of the topic. It is further assumed that there is a Post Office package which delivers messages to designated users, and an Employee Management package which provides employee reference and email address. Fig.12(a) shows the initial use case which represents most of the described functionality but is incomplete and serves as a starting point in moving toward a more extensive use case diagram to illustrate the use of UML-TAM in tracing rationale for change. Fig.12(b) gives an initial class diagram, and is used in the remainder to illustrate traceability within class diagram with UML-TAM.



(a) An initial use case diagram (b) An initial class diagram.

Fig. 12. Some initial UML diagrams for the case study.

10.5 Traceability through Argumentation in UML-TAM

Returning to the initial use case digram in Fig.12(a), it is not difficult to notice it is incomplete at least with regards to the following plausible situations:

- If the room of requested size is not available at the requested period, various alternative responses by the system can be identified: e.g., it may record a failed request for a room for statistics on room availability; another option is to communicate the unavailability to the user and ask for a different period.

[1] As noted above, UML-TAM is not intended for recording rationale behind all modeling decisions for it is not automated and thus impractical—contributions are primarily conceptual and not related to efficiency per se in the present paper. An accessible case study, appropriate for the constraints of the present format, thus introduces the method, while scalability and cost to industrial projects are under study.

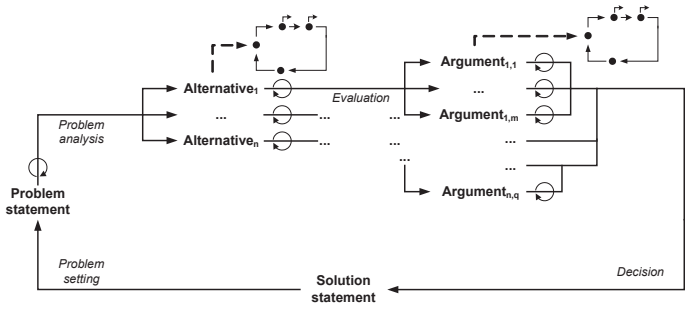


Fig. 13. Overview of the UML-TAM design rationale process.

- If an attendee is added as a participant to a recurrent meeting, should the system assume that this person is to attend all occurrences of the meeting in the future, or should the user specify this? Same applies when an attendee of a recurrent meeting is removed from the list of participants—does the removal apply for all occurrences of the meeting or only the next one?
- A participant informed of a meeting may have another engagement for the same period. Fig.12(a) gives no explicit mechanisms for ensuring the scheduler knows what participants to expect. The system could, e.g., connect to employees' electronic agendas and return availability information when the scheduler adds a participant and the meeting period.

It should be apparent from the above that providing a revised use case diagram alone—i.e., without additional information on why that particular revision is more adequate than another one—may be appropriate only in case the stakeholders are of similar background, share a precise idea of what the system is expected to do, and so on. In most realistic settings, however, this is not satisfactory, for various stakeholders would participate, each bringing a different perspective on the system grounded in different backgrounds and interests. The very presence of alternatives in both system functionality and of options in the representation of functionality (e.g., at the level of use cases: what to wrap in an existing use case, what requires an additional use case, and so on) makes it appropriate to make explicit the reasons (i.e., arguments) that aim to justify the functionality and representation decisions. One thus observes that three components are needed for ensuring traceability of rationale in UML: (1) a design rationale approach (below: TAM-Design Rationale, TAM-DR), which indicates when and how the engineer proceeds to making explicit the alternatives in functionality and/or modeling; (2) an argumentation framework, which, as soon as the alternatives are known, enables the argumentation of each alternative, the confrontation and comparison of arguments, ending in a justified choice of one alternative; (3) specialized means for connecting the content of UML diagrams with the content of rationale traces (referred to in the remainder as TAM-Connectors) produced through the use of the design rationale approach and associated argumentation and justification techniques.

10.5.1 UML-TAM Design Rationale

Having identified an engineering problem, design rationale literature (and as usual in problem solving) suggests the engineer should identify alternative solutions, compare them according to some relevant criteria, subsequently choose one alternative, and act upon the prescription given in the alternative. In the classical IBIS approach [64], the aforementioned problem is termed *issue* whereas *positions* (i.e., alternative solutions) resolve issues, and *arguments* support or object to positions. A problem in the present setting appears whenever alternative system structures can be chosen to translate stakeholder expectations into a UML representation, or when several modeling options exist for a chosen alternative system structure (i.e., one knows what to model, but syntax and semantics of the model permit various ways of modeling this). Based on work from Louridas and Loucopoulos [205], which integrates common characteristics of established design rationale approaches, a design rationale approach specialized for rationale traceability in UML-TAM involves the following steps (see, Fig.13):

1. *Problem setting* consists of identifying a discrepancy between the content of the given UML model instance and the content it should represent—e.g., some newly acquired information is not represented therein, or the given representation uses questionable modeling choices.
2. Based on the problem statement produced in 1 above, *problem analysis* leads to the identification of *alternative solutions.*

3. *Evaluation* then consists of providing arguments for or against each alternative solution. Such *argumentation* is followed by a *justification* of a choice of (i.e., *Decision* on) a particular alternative.
4. Having selected the alternative, the affected UML model instances need to be changed according to the adopted solution. The process is reinitiated as new problems are identified.

As shown in Fig.13, content of alternatives and arguments can give itself rise to new problem statements. Activities of the given process rely mainly on the domain- and problem-specific knowledge of the stakeholders. Argumentation and justification activities require specialized concepts and techniques outlined in §10.5.2 and §10.5.3. The use of the given concepts and techniques is exemplified in §10.6.

10.5.2 UML-TAM Argumentation Framework

Argumentation modeling literature [46] in the artificial intelligence field focuses on formalizing commonsense reasoning in the aim of automation. An *argumentation model* is a static representation of an *argumentation process*, which can be seen as a search for arguments, where an argument consists of a set of rules chained to reach a conclusion. Each rule can be rebutted by another rule based on new information. To formalize such defeasible reasoning, elaborate syntax and semantics have been developed (e.g., [46, 289, 27]) commonly involving a logic to formally represent the argumentation process and reason about argument interaction. A structured argumentation framework (i.e., a model and processes employing the model) is needed herein for a rigorous justification process in the *Evaluation* step of TAM-DR. To arrive at a structured argumentation system, the concept of argument is first defined below, followed by a set of argument relationships, and the justification process.

Argument. Assuming a first-order language $\mathcal{L}$ defined as usual, let $\mathcal{K}$ be a consistent set of formulae (i.e., $\mathcal{K} \nvdash \bot$), each a piece of information, and let $\mathcal{K} \equiv \mathcal{K}_N \cup \mathcal{K}_C$. Members of the set $\mathcal{K}_N$, called *necessary knowledge*, represent facts about the universe of discourse and are taken to be formulae which contain variables. Necessary knowledge is assumed unquestionable. The set $\mathcal{K}_C$, called *contingent knowledge*, are information that can be put in question or argued for. It is then said that the knowledge a stakeholder a can use in argumentation is given by the pair (K_a, Δ_a), where K_a is a consistent subset of $\mathcal{K}$ (i.e., $K_a \subset \mathcal{K}$ and $K_a \nvdash \bot$), and Δ_a is a finite set of *defeasible rules* of the form $\alpha \hookrightarrow \beta$. The relation $\hookrightarrow$ between formulae α and β is understood to express that "reasons to believe in the antecedent α provide reasons to believe in the consequent β". In short, $\alpha \hookrightarrow \beta$ reads "α is reason for β".

Let A a set of stakeholders, $K \equiv \bigcup_{a \in A} K_a$, and $\Delta \equiv \bigcup_{a \in A} \Delta_a$. Given (K_a, Δ_a) and $P \subset \Delta_a^{\downarrow}$, where $\Delta_a^{\downarrow}$ is a set of formulae from Δ_a instantiated over constants of the formal language, P is an *argument* for $c \in \mathcal{K}_C$, denoted $\langle P, c \rangle_K$, if and only if: 1) $K \cup P \mathrel{|\!\sim} c$ (K and P derive c); 2) $K \cup P \nvdash \bot$ (K and P are consistent); and 3) $\nexists P' \subset P, K \cup P' \mathrel{|\!\sim} c$ (P is minimal for K). Where "$\mathrel{|\!\sim}$" is called the *defeasible consequence* [289] and is defined as follows. Define $\Phi = \{\phi_1, \ldots, \phi_n\}$ such that for any $\phi_i \in \Phi$, $\phi_i \in K \cup \Delta^{\downarrow}$. A formula ϕ is a defeasible consequence of Φ (i.e., $\Phi \mathrel{|\!\sim} \phi$) if and only if there exists a sequence $B_1, \ldots, B_m$ such that $\phi = B_m$, and, for each $B_i \in \{B_1, \ldots, B_m\}$, either B_i is an axiom of $\mathcal{L}$, or B_i is in Φ, or B_i is a direct consequence of the preceding members of the sequence using modus ponens or instantiation of a universally quantified sentence. This argument definition is well-understood in the AI literature [46, 264].

Argumentation. While an argument can be constructed by combining explicitly expressed knowledge (e.g., from a knowledge base), the aim here is to start from a conclusion and build arguments that support it from the knowledge that stakeholders provide and that can be related to the conclusion. Argumentation of a conclusion R consists of recursively defining an argument tree AT_R as follows:

1. Define R as the root of the tree AT_R and set $c = R$;
2. Let $\langle P, c \rangle$. Identify $p_1, \ldots, p_n$ s.t. $\{p_1, \ldots, p_n\} = P$, $P \subseteq K \cup \Delta^{\downarrow}$;
3. Define a node for each premise $p_i \in P$ and define an edge from that node to c. Draw the edge "$\longrightarrow$" if $p \in K$, or "$\longmapsto$" in case $p \in \Delta^{\downarrow}$;
4. Set $c = p_i$ and repeat steps 2 and 3 for each $i = 1, \ldots, n$, until the argument tree has been constructed to a satisfactory extent.

Argument Relationships. Of particular interest in argumentation is to confront arguments and reject some conclusion in favor of other. It is therefore necessary to define several simple relationships between arguments.

Two arguments $\langle P_1, c_1 \rangle$ and $\langle P_2, c_2 \rangle$ *disagree*, denoted by $\langle P_1, c_1 \rangle \bowtie_{\mathcal{K}} \langle P_2, c_2 \rangle$, if and only if $\mathcal{K} \cup \{c_1, c_2\} \vdash \bot$.

Instead of seeking contradiction of conclusions, a *counterargument* relation looks for incompatibility of a conclusion with the conclusion of a subargument of another argument. $\langle P_1, c_1 \rangle$ *counterargues at* c

the argument $\langle P_2, c_2\rangle$, denoted by $\langle P_1, c_1\rangle \not\rightarrow^c \langle P_2, c_2\rangle$, if and only if there is a subargument $\langle P, c\rangle$ of $\langle P_2, c_2\rangle$ such that $\langle P_2, c_2\rangle \bowtie_{\mathcal{K}} \langle P, c\rangle$ (i.e., $\langle P, c\rangle \subset \langle P_2, c_2\rangle$ and $\mathcal{K} \cup \{c_1, c\} \vdash \bot$).

In case two arguments are such that one counterargues the other, it is necessary to determine which of the two is to be maintained. An argument $\langle P_1, c_1\rangle$ *defeats at* c an argument $\langle P_2, c_2\rangle$, denoted by $\langle P_1, c_1\rangle \gg^c \langle P_2, c_2\rangle$, if and only if there is a subargument $\langle P, c\rangle$ of $\langle P_1, c_1\rangle$ such that (1) $\langle P_1, c_1\rangle \not\rightarrow^c \langle P_2, c_2\rangle$ (that is, $\langle P_1, c_1\rangle$ counterargues $\langle P_2, c_2\rangle$ at c); and (2) $\langle P_1, c_1\rangle \succ^c \langle P, c\rangle$ ($\langle P_1, c_1\rangle$ is more *specific* than $\langle P, c\rangle$). In a dialectical tree (see below), defeat is represented by "$\not\longrightarrow$" directed from the conclusion of the argument that defeats to the node which is defeated. The specificity relation "$\succ^c$" is an order relation over arguments, defined so that arguments containing more information, i.e., which are more specific, are preferred over other. An argument $\langle P_1, c_1\rangle$ is *strictly more specific than* $\langle P_2, c_2\rangle$, denoted by $\langle P_1, c_1\rangle \succ^c \langle P_2, c_2\rangle$ if and only if (1) $\forall e \in \mathcal{K}_C$ such that $\mathcal{K}_N \cup \{e\} \cup P_1 \mathrel{|\!\sim} c_1$ and $\mathcal{K}_N \cup \{e\} \not\mathrel{|\!\sim} c_1$, also $\mathcal{K}_N \cup \{e\} \cup P_2 \mathrel{|\!\sim} c_2$; and (2) $\exists e \in \mathcal{K}_C$ such that: (2.1) $\mathcal{K}_N \cup \{e\} \cup P_2 \mathrel{|\!\sim} c_2$; (2.2) $\mathcal{K}_N \cup \{e\} \cup P_1 \not\mathrel{|\!\sim} c_1$; (2.3) $\mathcal{K}_N \cup \{e\} \not\mathrel{|\!\sim} c_2$.

Justification. Argument defeat is employed when attempting to justify a particular conclusion. The *justification* process consists of recursively defining and labeling a *dialectical tree* $\mathcal{T}\langle P, c\rangle$ as follows:

1. A single node containing the argument $\langle P, c\rangle$ with no defeaters is by itself a dialectical tree for $\langle P, c\rangle$. This node is also the root of the tree.
2. Suppose that $\langle P_1, c_1\rangle, \ldots, \langle P_n, c_n\rangle$ each defeats $\langle P, c\rangle$. Then the dialectical tree $\mathcal{T}\langle P, c\rangle$ for $\langle P, c\rangle$ is built by placing $\langle P, c\rangle$ at the root of the tree and by making this node the parent node of roots of dialectical trees rooted respectively in $\langle P_1, c_1\rangle, \ldots, \langle P_n, c_n\rangle$.
3. When the tree has been constructed to a satisfactory extent by recursive application of steps 1 and 2 above, label the leaves of the tree *undefeated* (U). For any inner node, label it undefeated if and only if every child of that node is a *defeated* (D) node. An inner node will be a defeated node if and only if it has at least one U node as a child. Do step 4 below after the entire dialectical tree is labeled.
4. $\langle P, c\rangle$ is a *justification* (or, P justifies c) if and only if the node $\langle P, c\rangle$ is labeled U.

Dialectical trees are shown in the UML-TAM traceability templates in Figures 15 and 16, in §10.6; arguments are drawn enclosed in boxes, a dialectical tree relates such boxes with the defeat relationship. The content of arguments is informally expressed, and can be replaced (pending some adjustments) with first-order formulae. However, the informal character thereof does not affect the ability to manually determine relationships between arguments, as they have been presented above, and consequently to proceed to justification. Having formal foundations, as suggested in the present subsection contributes to the precision of the conceptual bases for the argumentation and justification activities.

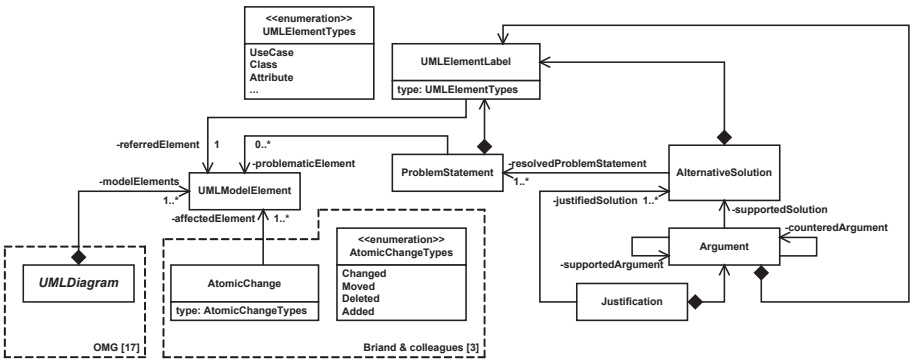


Fig. 14. Metamodel relating UML-TAM to the UML 2.0 metamodel.

10.5.3 UML-TAM Connectors

Connectors in UML-TAM relate information used and produced with the design rationale, and argumentation and justification techniques to the content of the UML diagrams whose rationale traceability is to be ensured. Fig.14 shows the metamodel, written in UML class diagram notation, integrating the relevant concepts of UML-TAM and relating them to the UML 2.0 metamodel [112] through the *UMLDiagram* class. Although the illustration §10.6 discusses the traceability in use case and class diagrams, the metamodel does not limit the potential for bridging UML-TAM and other UML diagrams.

The part of the metamodel proper to UML-TAM integrates the concept of ProblemStatement, AlternativeSolution, Argument, and Justification, each following the definitions given in previous subsections. Note the ProblemStatement can be associated to no UMLModelElement, which occurs when the ProblemStatement results in the addition of new UMLModelElement instances into a UMLDiagram instance. The metamodel is linked to a part of the metamodel underlying the bi-directional link traceability approach from Briand and colleagues [42]: AtomicChange is a modification applicable to the UML diagram, whose execution gives rise to a number of traceability links to ensure that information about what changed and how is captured. The types of atomic changes given in the figure are the basic ones, whereby more extensive taxonomies are suggested by refining each of the four activities, and this depending on the syntax of the underlying UML diagram [42]. An important practical consequence of the above metamodel is that UML-TAM can be thus be combined to automated traceability methods and applied selectively, when stakeholders explicitly identify problems which in turn entail the use of UML-TAM for resolution.

As the content of arguments can be informal or formal, labels are used to highlight the relevant elements of the UML model being mentioned in arguments, alternative solutions, and/or problem statement. The UMLElementLabel concept is thus introduced in the metamodel in Fig.14. In Fig.15, labels are placed within arguments and the alternative solution, whereas the problem statement (the title of the UML-TAM traceability template) does not contain explicit references to elements of the use case diagram, and therefore contains no labels.

The approach to relate the UML artifacts and those produced in UML-TAM is straightforward: as soon as a justified alternative solution is found, and the stakeholders no longer provide arguments to defeat it (i.e., the justification process ends), change is performed in the corresponding UML diagram. A template is filled out—it contains a snapshot showing the original structure of the part of the diagram that is being changed, the problem statement, the alternative solutions, the justification, and all arguments provided for each alternative solution.

10.6 Applying UML-TAM

It has been observed earlier that the initial use case diagram shown in Fig.12(a) is incomplete in several respects. Using UML-TAM, two changes were performed leading to the use case diagram in Fig.15. There, labels are placed on the elements of the diagram to relate them to *traceability templates* used in UML-TAM to summarize information used and produced in moving from the initial version of the diagram to that presented in Fig.15. Each template contains four parts: (i) a label (e.g., T1, T2) for relating the elements of the diagram to the template; (ii) a title, which is the problem statement requiring diagram change; (iii) the dialectical tree for the justified alternative solution; and (iv) the dialectical trees for the rejected alternative solutions. Following the metamodel in Fig.14, information referring to UML diagram elements and appearing in the template is labeled following the kind of UML element the information refers to.

Figures 15 and 16 are self-explanatory and show modified initial use case and class diagrams obtained by applying UML-TAM. Each has been constructed by applying the UML-TAM. Practical experience with UML-TAM that goes beyond the simple, yet illustrative case presented here leads to several observations about the practical use of the proposed method. For instance, it has been empirically observed that nonmonotonic reasoning is hard for humans [94]. Effort involved in finding arguments in UML-TAM is considerable and appears to confirm the cited empirical result. Some techniques derived from theory are particularly hard to apply in practice: for instance, comparing arguments for specificity appeared counterintuitive and was thus seldom used. Prior experience and resources about the debated domain are relevant sources of arguments, so that referring to these is suggested. Although the difficulties are considerable when applying argumentation and justification, a significant benefit is that these techniques lead to the externalization of information usually left implicit in UML modeling. The information made explicit is available to a number of stakeholders who can, through argumentation and justification, question and revise the modeling decisions. Moreover, lessons can be learned from past modeling problems as sources of the problems (such as, e.g., fallacious argumentation) can be identified by going back to the recorded arguments. UML-TAM is therefore of interest for projects in which particularly high degree of rigor is required, as in the case of, e.g., safety-critical systems.

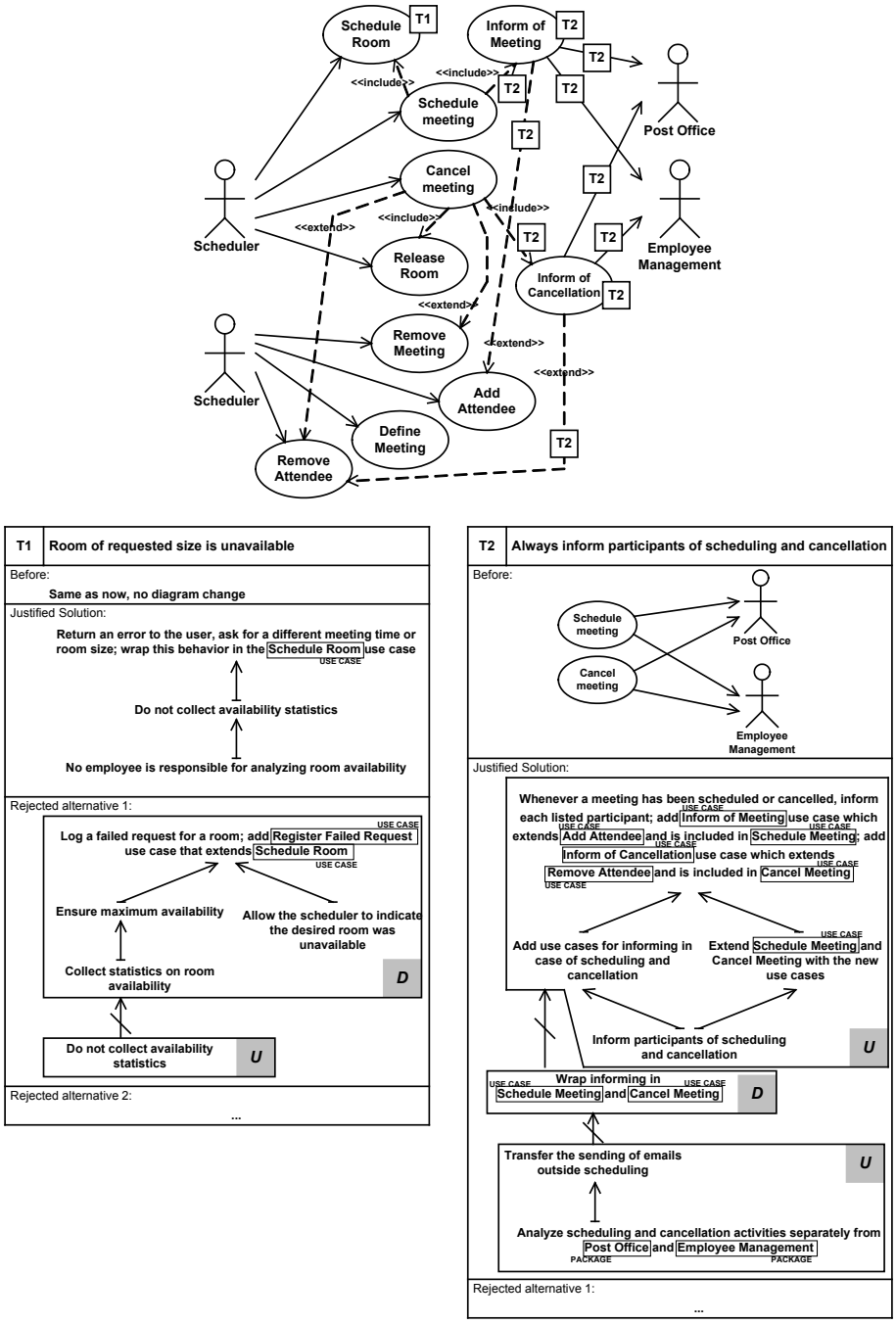


Fig. 15. The modified use case diagram with accompanying rationale traceability information produced with UML-TAM.

10.7 Conclusions and Future Work

The UML Traceability through Argumentation Method presented herein introduces rigorous argumentation and justification when tracing the rationale behind UML modeling decisions. The main contributions are: (1) The information about the design rationale used in modeling is usually lost or, when available, stated in an unstructured manner. UML-TAM provides a simple, yet precise means for representing this information, analyzing it for problematic rationale (by justification), and using it to arrive at justifiably appropriate modeling decisions. (2) Both qualitative and quantitative, informal and formal information can be put into arguments allowing the application of the method to a wide range of settings. (3) When combined with traceability approaches focused on answering how, what, when, and who modified a UML diagram, UML-TAM allows answering and discussing *why* a change was needed. (4) By applying argumentation and justification activities, the modeler can claim that a modeling choice is appropriate or not, while relying on solid and well understood conceptual foundations and rigorous processes for their

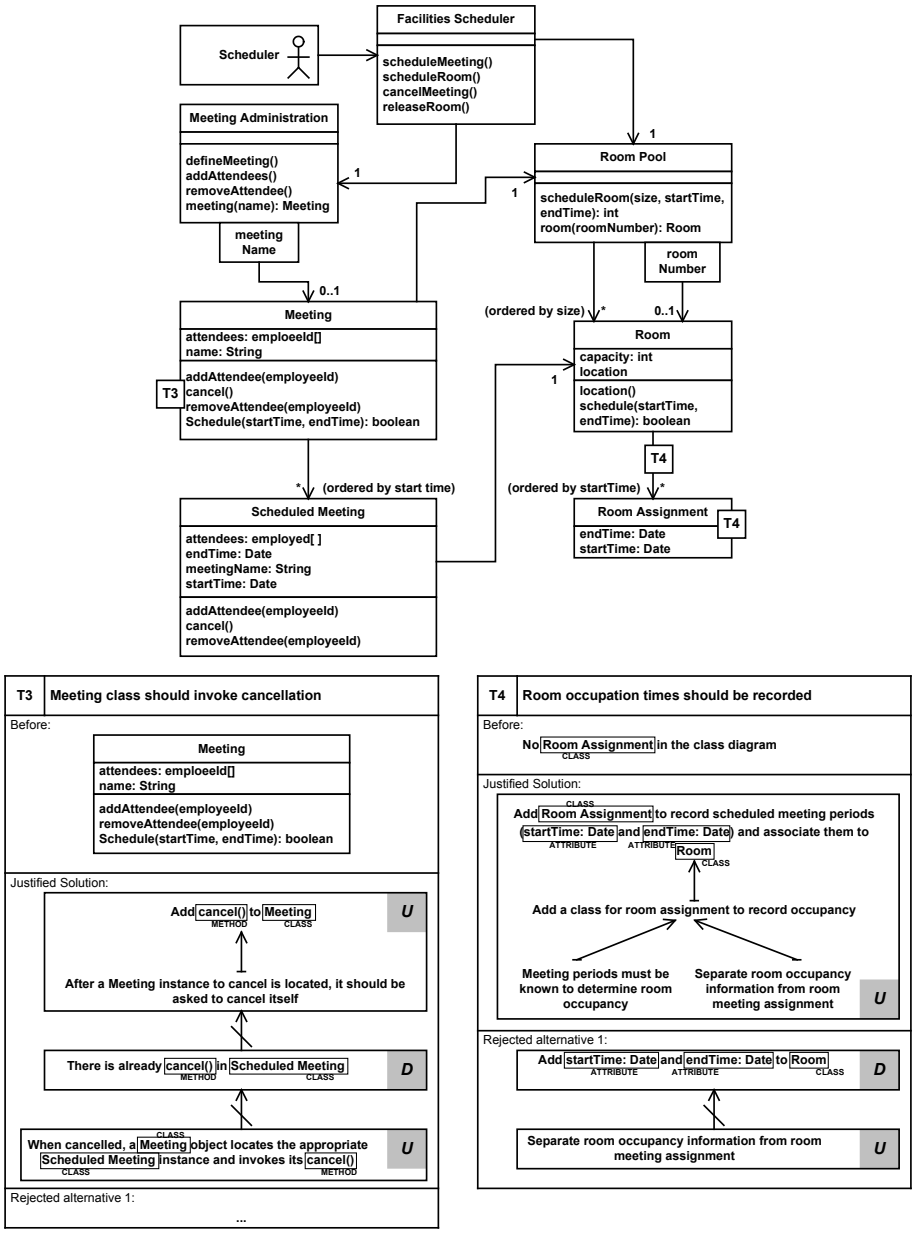


Fig. 16. The modified class diagram with accompanying rationale traceability information produced with UML-TAM.

use. Modeling choices can thus be claimed as justified, or questioned through a step-by-step process. Following the outline of related research efforts §10.2, the proposed method advances the rationale traceability literature, while ensuring compatibility with approaches focusing on traceability of other types of information—this is accomplished by focusing the method on a precise traceability issue, proposing connection points for relating the method to compatible approaches, and avoiding overlap with related techniques.

Current effort includes the exploration of benefits of formalizing arguments in combination with various UML formalizations, to attempt automated analysis of argument and associated UML diagram structures. Experimentation is currently performed to improve usability in industrial settings.

11

Dynamic Requirements Specification for Adaptable and Open Service-Oriented Systems

Abstract. It is not feasible to engineer requirements for adaptable and open service-oriented systems (AOSS) by specifying stakeholders' expectations in detail during system development. Openness and adaptability allow new services to appear at runtime so that ways in, and degrees to which the initial stakeholders' functional and nonfunctional requirements will be satisfied may vary at runtime. To remain relevant after deployment, the initial requirements specification ought to be continually updated to reflect such variation. Depending on the frequency of updates, this paper separates the requirements engineering (RE) of AOSS onto the RE for: individual services (Service RE), service coordination mechanisms (Coordination RE), and quality parameters and constraints guiding service composition (Client RE). To assist existing RE methodologies in dealing with Client RE, the Dynamic Requirements Adaptation Method (DRAM) is proposed. DRAM updates a requirements specification at runtime to reflect change due to adaptability and openness.

11.1 Introduction

To specify requirements, the engineer describes the stimuli that the future system may encounter in its operating environment and defines the system's responses according to the stakeholders' expectations. The more potential stimuli she anticipates and accounts for, the less likely a discrepancy between the expected and observed behavior and quality of the system. Hence a longstanding concern in requirements engineering (RE) on requirements *completeness* (e.g., [333, 315]): i.e., ensuring that the specification accounts for all the relevant stakeholder expectations and environment properties.

Ensuring completeness becomes increasingly difficult as systems continue to gain in complexity and/or operate in changing conditions (e.g., [307, 174, 308]). One relevant response to complexity is to rely on self-contained components [144, 49, 246], as in *service-orientation*, where individual services are "self-describing, open components that support rapid, low-cost composition of distributed applications" [246]. An *open* service-oriented system may involve a large pool of distinct and competing services orignating from various service providers. To be *adaptable*, such a system coordinates service provision by dynamically selecting the participating services according to multiple quality criteria, so that the users continually receive optimal results.

A complete requirements specification for an AOSS would include the description of all relevant properties of the system's operating environment, and of all alternative system and environment behaviors. All services that may participate would thus be entirely known at development time. Following any established RE methodology (e.g., KAOS [71], Tropos [50]), such a specification would be constructed by moving from abstract stakeholder expectations towards a detailed specification of the entire system's behavior. Applying such an approach and arriving at the extensive specification of an AOSS is not feasible because: (i) the set of services that may participate is not known at development time—it is therefore unknown how and to what extent the participating services will satisfy the initial requirements; (ii) an adaptable system is usually needed when the environment is unpredictable—the requirements engineer thus cannot be expected to anticipate all possible operating conditions, and environment and system behaviors; (iii) services and the AOSS are unlikely to be developed by the same people and/or organizations—consequently, all providers cannot be expected to operationalize the same abstract requirements when defining service behaviors.

Following the above observations (i)–(iii), it appears that building a requirements specification to subsequently guide development and ensure stakeholder expectations are satisfied to optimal levels at

runtime carries novel concerns in the case of AOSS compared to non-adaptable and closed systems. As a result of (i), runtime changes in the ways in, and the extent to which stakeholders' expectations will be satisfied ought to be reflected in the requirements specification if the aim remains to ensure that the expectations are not violated. A consequence of (ii) is that it can be inappropriate to attempt to extensively specify initial requirements for the AOSS: placing extensive constraints at the outset limits openness and adaptability. That is, it limits the potential for satisfying stakeholders' requirements at runtime in ways different than those anticipated and allowed in the initial specification: in, e.g., a travel booking system, an airline may allow the user to specify entertainment preferences which is usually perceived as a richer and personalized offering, while its competitors may not do so; if this option has not been anticipated and allowed in the initial specification, strong restrictions on service behavior will block such unanticipated yet relevant service options. Instead, it may be relevant to stop refining and operationalizing requirments at a relatively early stage in the RE for AOSS. Finally, (iii) points to a separation of the RE effort in the case of AOSS: e.g., service providers will perform RE at a different level than those concerned with how services will interact in the AOSS as a whole. In summary, when performing the RE of AOSS: (a) the initial specification ought to be updated at runtime; (b) the initial specification need not be extensive; and, (c) a separation is to be made in the RE for AOSS according to the various concerns of the developers of the services and of the AOSS.

Contributions. Following (a)–(c), this paper introduces concepts and techniques needed to *(1) determine how extensive the initial specification ought to be and what parts thereof are to be updated at runtime to reflect system adaptation*, and *(2) know how to perform such updates*. The specification can then be used to continually survey and validate system behavior. To enable (1), this paper separates the requirements engineering (RE) of AOSS depending on the frequency at which the requirements are to be updated: RE executed for individual services or small sets of services (*Service RE*), RE of mechanisms for coordinating the interaction between services (*Coordination RE*), and RE of parameters guiding the runtime operation of the coordination mechanisms (*Client RE*). Completeness concerns (as they are traditionally known in RE [333, 315]) remain for Service and Coordination RE where requirements vary less frequently than in Client RE. Coordination parameters dealt with at Client RE guide the service selection and composition based on various quality parameters and constraints, influencing therefore the degree to which the functional and nonfunctional concerns of the stakeholders are met by the system. With openness, each new service may have a different set of quality parameters unknown before the system is running, so that requirements completeness at Client RE becomes a continuous aim that extends beyond development time and over to runtime. Returning to the travel booking system, instead of placing extensive constraints, the aim at Client RE is to have mechanisms for updating the initial specification at runtime to reflect adaptation to service characteristics and availability, so that unanticipated parameters may be taken into consideration instead of being blocked. To address (2), this paper focuses on Client RE and introduces a method, called Dynamic Requirements Adaptation Method (DRAM) for performing Client RE for AOSS. DRAM is grounded in a *dynamic* requirements specification which is continually updated to reflect change in the initial requirements specification (one constructed before system is in operation), whereby changes result from the appearance of new services and new/revised stakeholder expectations, both assumed to occur at runtime.

Motivating example. The proposal outlined in the remainder resulted from the difficulties encountered in engineering requirements for an AOSS, call it TravelWeb, which allows users to search for and book flights, trains, hotels, rental cars, or any combination thereof through a web interface. Services which perform search and booking originate from the various service providers that either represent the various airlines and other companies, so that TravelWeb aggregates and provides an interface to the user when moving through the offerings of the various providers. Each provider can decide what options to offer to the user: e.g., in addition to the basics, such as booking a seat on an airplane, some airlines may ask for seating, entertainment, and food preferences, while others may further personalize the offering through additional options.

Organization. The separation of RE for AOSS is introduced first (§11.2). The conceptual foundations and techniques integrated in DRAM are then presented and illustrated (§11.3). After related work is reviewed (§11.4), the paper closes with conclusions and indications for future effort (§11.5).

11.2 Service, Coordination, and Client RE

To engineer the requirements for TravelWeb, a common RE methodology such as Tropos [50] would start with early and late requirements analyses to better understand the organizational setting, where

dependencies between the service providers, TravelWeb, and end users would be identified, along with the goals, resources, and tasks of these various parties. Architectural design would ensue to define the sub-systems and their interconnections in terms of data, control, and other dependencies. Finally, detailed design would result in an extensive behavioral specification of all system components. While other methodologies, such as, e.g., KAOS [71] involve a somewhat different approach, all move from high-level requirements into detailed behavioral specifications. The discussion below, however, concludes that such an approach is not satisfactory, because:
1) TravelWeb is open. Various hotels/airlines/rental companies may wish to offer or retract their services. Characteristics of services that may participate in TravelWeb at runtime is thus unknown at TravelWeb development time. Individual services are likely to be developed outside the TravelWeb development team, before or during the operation of TravelWeb. It is thus impossible to proceed as described for the entire TravelWeb—instead, it is more realistic to apply the RE methodology locally for each individual service, and separately for the entire TravelWeb system, taking individual services as black boxes of functionality (i.e., without knowing of their internal architecture, detailed design, etc.).
2) Resources are distributed and the system adapts. All services may or may not be available at all times. Moreover, individual services are often not sufficient for satisfying user requests—that is, several services from distinct providers may need to interact to provide the user with appropriate feedback. Adaptability in the case of TravelWeb amounts to changing service compositions according to service availability, a set of quality parameters, and constraints on service inputs and outputs. Consequently, RE specific to the coordination of services carries distinct concerns from the engineering of requirements for individual services.
3) Quality parameters vary. Quality parameters for selecting among services are not all known while engineering requirements for TravelWeb. In other words, the vector of quality parameters that guide service coordination is not of a predefined format for all services. It is thus difficult to take as satisfactory a static specification of stakeholders' nonfunctional expectations produced before TravelWeb is in operation: as the sets of quality parameters to account for in composing services change, (a) different sets of stakeholders' nonfunctional expectations will be concerned by various service compositions *and* (b) there may be quality parameters that do not have corresponding expectations in the initial specification. Observation (a) entails that initial desired levels of expectations may not be achieved at all times, making the initial specification idealistic. Deidealizing requirements has been dealt through a probabilistic approach by Letier and van Lamsweerde [198] where requirements are combined with probability of satisfaction estimates. In an adaptable system, the probability values are expected to change favorably over time (see, e.g., our experiments on service composition algorithms for AOSS [148]), so that updating the initial requirements specification to reflect the changes seems appropriate if the specification is to remain relevant after system deployment. Observation (b) relates to the difficulty in translating stakeholders' goals into a specification: as March observed in a noted paper [210], both individual and organizational goals (which translate into requirements herein) tend to suffer from problems of relevance, priority, clarity, coherence, and stability over time, all of which relate to the variability, inconsistency, and imprecision, among other, of stakeholder preferences. Instead of assuming that the initial set of expectations is complete, the specification can be updated at runtime to reflect new system behaviors *and* to enable the stakeholders to modify requirements as they learn about the system's abilities and about their own expectations.

When engineering requirements for non-adaptable and closed systems, such as fully-automated metro lines, mine pumps, ambulance dispathers, and air traffic management systems, where safety is often a primary concern, static requirements specifications produced by established methodologies such as KAOS [71] and Tropos [50] have clear merits (e.g., [315, 316]). However, the above discussion makes it difficult to assume that stakeholders will perceive an AOSS as of high quality if it satisfies only a predefined set of requirements: the system is open and adapts, so that new quality parameters and/or constraints may become relevant *and* stakeholders' goals may change at runtime, along with their preferences both over alternative requirements and alternative ways of satisfying the said requirements.

A requirements specification for an AOSS involves requirements that are of different variability over time. Our experience with AOSS [148] indicates that a particular combination of service-oriented architecture and service coordination algorithm enables adaptability, whereby the architecture and the algorithm act as a cadre in which various requirements can be specified. Since adaptability does not require change in the architecture and algorithm, requirements on these two remain reasonably stable. This observation, along with the localization of service-specific RE to each individual service or small service groups leads to a separation of AOSS RE effort as follows:
Service RE involves the engineering of requirements for an individual service, or a set of strongly related services (e.g., those obtained by modularization of a complex service). Depending on whether

the service itself is adaptable, a classical RE methodology such as Tropos or KAOS can be applied, with or without extensions for engineering adaptable systems (e.g., [44]). Note that the service-level adaptability mentioned here differs from the adaptability of the system as a whole: the service may adapt to perform some tasks in a different sequence or manner, while the system adapts by changing the compositions of services used to fulfil service requests. As the coordination mechanism selects individual services for fulfiling user requests, requirements on an individual service do not change with changes in user requests (inputs and/or outputs and constraints on these and quality parameters change with variation in requests).

Coordination RE takes services as self-contained functionality and focuses on the requirements for the coordination of services. In an AOSS, this typically involves the definition of the architecture to enable openness, service interaction, service selection, and service composition for providing more elaborate, composite services to fulfil user requests. As noted above, these requirements vary less frequently than those elicited as a result of Client RE.

Client RE assumes a coordination mechanism is defined and is guided by constraints to obey, and quality parameters to optimize (e.g., QoS, execution time, service reputation). This is the case after a service-oriented architecture is defined in combination with an algorithm for service coordination (see, e.g., [148]). The aim of the requirements engineer at this level is to define constraints on inputs and/or outputs, quality parameters, and quality parameter values which will direct coordination at runtime *and* define mechanisms used in updating the specification according to change in the system's behavior at runtime. Following the earlier observations, the set of constraints and quality parameters is likely to vary as new services appear and other dissapear. Quality parameter values will vary as well, as the system adapts to the availability of the various services and change in stakeholders' expectations.

A generic AOSS model is given below to facilitate the discussions in the remainder. The model allows a more precise separation of information that is to be produced in Service, Coordination, and Client RE. The model assumes that the AOSS contains a number of *Mediator Services* (MS), each responsible for fulfilling *Service Requests* (SReq) originating from a user. The MS negotiates with and composes *Services* (Serv) into a *Composite Service* (CServ) according to the demands set out in a SReq, while observing past performance of individual Serv, then subsequently using (and updating) this information through a service composition mechanism (i.e., an algorithm—see, e.g., [148]) in the aim of continually optimizing delivery quality. The algorithm guiding the behavior of MS is assumed to be multi-criteria and multi-constraint driven, whereby criteria (i.e., quality parameters) to optimize and constraints to obey are to be (partially or completely) specified by the user in the SReq.

Definition 11.1. *$s = \langle I, O, \sigma, \hat{\mathbf{q}} \rangle$ is called a **Service** (Serv). I and O specify, respectively, the inputs and the outputs. σ is a detailed specification of the service, which in practice includes the behavioral specification of service operation, interfaces, communication protocols, and additional properties necessary to deploy the service. $\hat{\mathbf{q}}$ is a vector of pairs, $\hat{\mathbf{q}} = ((q_1^{\mathsf{Name}}, b^{q_1}), \ldots, (q_n^{\mathsf{Name}}, b^{q_n}))$, where q_i^{Name} is the Name of the quality parameter q_i (see, Def.11.10) and b^{q_i} the constraint on the values of the parameter specified using the following syntax (in BNF notation):*

$$\begin{aligned} a &::= e \mid \{e\} \mid v \mid (\geq v) \mid (\leq v) \mid \{v\} \\ b &::= a \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \end{aligned} \tag{3}$$

Where v is a real, $v \in \mathbb{R}$; e is a member of some finite set E which enumerates possible values associated to a quality parameter with a discrete type.

Requirements on $I, O, \sigma, \hat{\mathbf{q}}$ are defined at Service RE. $\hat{\mathbf{q}}$ gives *advertised* and not the observed values: the MS obtains the latter through actual observation of Serv behavior in various CServ.

Definition 11.2. ***Mediator Service** (MS) s^M is a service capable of coordinating other Serv: $s^M = \langle I^M, O^M, \sigma^M \rangle$ s.t. $\chi \in \sigma^M$, where χ denotes the coordination behavior.*

At the input I^M, $\chi \in \sigma^M$ needs the service request (Def.11.4) and the information on available services in order to determine the service composition satisfying the request. In a separate paper [148], we have suggested a reinforcement learning algorithm to enable efficient behavior of MS. The algorithm optimizes individual quality parameters or parameter aggregates while following one or more hard constraints on quality parameter values. The specification of χ involves detailing the algorithm, along with topics such as, e.g., discovery of Serv and negotiation between the MS and individual Serv. s^M is specified at Coordination RE.

Definition 11.3. *$\langle s^N, s^E, transit, state, s^\iota \rangle$ is a **Composite Service** (CServ) š. (s_j^N, s_j^E) defines a directed acyclic graph. A node is a description of conditions that hold after the Serv (that associated with the entering edge) executes. A Serv is associated to each edge, so that moving along the graph amounts to executing services in a sequence. The two functions label nodes and edges with service information: $transit : s^E \longmapsto \mathbb{S}$ is a partial function returning the Serv for a given edge in the graph ($\mathbb{S}$ is the set of all Serv available to the MS in the system), while $state : s^N \longmapsto \{I\}_{s\in\mathbb{S}} \cup \{O\}_{s\in\mathbb{S}}$ maps each edge to inputs or outputs of Serv. The service on an edge must have the inputs and outputs corresponding to conditions given, respectively, on its origin and its destination node.*

The MS builds (s_j^N, s_j^E) and labels it at runtime according to SReq and $\chi \in \sigma^M$. The above conceptualization allows the service composition problem to be formulated as a deterministic shortest-path problem in a directed weighted hypergraph (see, e.g., [148]). Making a request to the system amounts to providing the specification of constraints on inputs and outputs, along with the definition of a quality parameter vector. In TravelWeb, inputs could be, e.g., the destination city/country and a travel date range; desired output would be specified by stating that the aim is a flight & hotel combination, and quality parameters may include the desired price range, flight class, sports preferences, room size, ocean view, etc. Client RE, at which DRAM is applied (see, §11.3), deals with the specification of requirements on the components of $\ddot{s}$ only.

Definition 11.4. *$\ddot{s} = \langle C, \ddot{\mathbf{q}} \rangle$ is a **Service Request** (SReq), where C is a set of constraints that must hold at the inputs and/or outputs of the CS (or its member Serv) which a MS composes to deliver the SReq, and $\ddot{\mathbf{q}}$ is a vector of pairs: $\ddot{\mathbf{q}} = ((q_1^{\mathsf{Name}}, v^{q_1}), \ldots, (q_n^{\mathsf{Name}}, v^{q_n}))$. In each quality pair, q_i^{Name} is the Name of the quality parameter q_i (see, Def.11.10), v^{q_i} is the constraint on the values associated to the quality parameter according to the syntax in Eq.3 and the associated semantics.*

11.3 Using DRAM at Client RE

DRAM is not a standalone RE methodology—it does not indicate, e.g., how to elicit stakeholder expectations and convert these into precise requirements. Instead, DRAM integrates concepts and techniques for defining mappings between fragments of the requirements specification produced by an existing RE methodology and elements defining service requests $\ddot{s}$. Mapping requirements onto SReqs aims to ensure that the stakeholders' expectations are translated into constraints and quality parameters understood by the AOSS. Mapping in the other direction—from Sreqs onto requirements—allows the initial (also: static) requirements specification to be updated to reflect runtime changes in the system due to adaptability and openness. The specification obtained by applying DRAM on the initial, static requirements specification is referred to as the *dynamic requirements specification.*

Definition 11.5. ***Dynamic requirements specification** $\mathcal{S}$ is $\langle \mathsf{R}, \mathcal{R}, \mathcal{Q}, \mathcal{P}, \mathcal{U}, \mathcal{A} \rangle$, where: R is the **static requirements specification**; $\mathcal{R}$ the set of **service requirements**; $\mathcal{Q}$ the set of **quality parameters**; $\mathcal{P}$ the **preferences specification**; $\mathcal{U}$ the set of **update rules**; and $\mathcal{A}$ the **argument repository**.*

Members of R are specifications of nonfunctional and functional requirements, taking the form of, e.g., goals, softgoals, tasks, resources, agents, dependencies, scenarios, or other, depending on the RE methodology being used. Nonfunctional requirements from R are mapped onto elements of $\mathcal{Q}$ and $\mathcal{P}$, whereas functional requirements from R onto service request constraints grouped in $\mathcal{R}$. As equivalence between fragments of R and $\mathcal{R}, \mathcal{Q}, \mathcal{P}$ can seldom be claimed, a less demanding binary relation is introduced: the *justified correspondence* "$\triangleq$" between two elements in $\mathcal{S}$ indicates that there is a *justification* for believing that the two elements correspond in the given AOSS, at least until a defeating argument is found which breaks the justification. In other words, the justified correspondence establishes a mapping between instances of concepts and relationships in the language in which members of R are written and the language in which members of $\mathcal{R}, \mathcal{Q}, \mathcal{P}$ are written. The preferences specification $\mathcal{P}$ contains information needed to manage conflict and subsequent negotiation over quality parameters that cannot be satisfied simultaneously to desired levels. Update rules serve to continually change the contents of R according to system changes at runtime. Finally, the argument repository $\mathcal{A}$ contains knowledge, arguments, and justifications used to construct justified correspondences and at other places in $\mathcal{S}$, as explained below.

The distinctive property of $\mathcal{S}$ is that it is continually updated to reflect change in the service used to fulfil service requests. Updates are performed with update rules: an update rule will automatically (or with limited human involvement) change the R according to the quality parameters, their values, and the constraints on inputs and outputs characterizing the services composed at runtime to satisfy service

requests. An update rule can thus be understood as a mapping between fragments of R and those of $\mathcal{R}, \mathcal{Q}, \mathcal{P}$. Consequently, it is according to the constraints on inputs/outputs and quality parameter values observed at runtime that fragments of requirements will be added or removed to R. Update rules work both ways, i.e., change in R is mapped onto service requests, and the properties of services participating in compositions are mapped onto fragments of R.

Building fully automatic update rules is difficult for it depends on the precision of the syntax and semantics of languages used at both ends, i.e., the specification language of the RE methodology which produces R and the specification language employed to specify input/output constraints on services and quality parameters. Due to a lack of agreement on key RE concepts and the relatively early stage of works in service-oriented computing, DRAM makes no assumptions about the languages employed for writing R, $\mathcal{R}$, and $\mathcal{Q}$. Hence the assumption that languages at both ends are ill-defined, and the subsequent choice of establishing a justified correspondence between specification fragments. The cost of such an approach is that update rules in many cases cannot be established automatically—the repository of update rules is built at testing and runtime.

$\mathcal{S}$ integrates the necessary means for constructing update rules: to build justified correspondences between elements of R and $\mathcal{R}, \mathcal{Q}, \mathcal{P}$, arguments are built and placed in the argument repository $\mathcal{A}$. Update rules are automatically extracted from justified correspondences. As competing services will offer different sets of and values of quality parameters at service delivery, and as not all will be always available, trade-offs performed by the AOSS need to be appropriately mapped to R. Moreover, stakeholders may need to negotiate among themselves the selection of quality parameters and their values. $\mathcal{P}$ performs the latter two roles.

The remainder of this section introduces the various parts of $\mathcal{S}$ along with the techniques used to manipulate the proposed concepts. Together, they will enable the definition of update rules which are the core of DRAM.

Definition 11.6. *The* ***static requirements specification*** R *is the high-level requirements specification obtained during RE before the system is in operation.*

R is obtained by applying a RE methodology, such as, e.g., KAOS [71] or Tropos [50]. The meaning of "high-level" in Def.11.6 varies accross RE methodologies: if a goal-oriented RE methodology is employed, R must contain the goals of the system down to the operational level, so that detailed behavioral specification in terms of, e.g., state machines, is not needed. If, e.g., KAOS is used, the engineer need not move further than the specification of goals and concerned objects, that is, can stop before operationalizing goals into constraints. If Tropos is used, the engineer stops before architectural design, having performed late requirements analysis and, ideally, formal specification of the functional goals using Formal Tropos [100].

Example 11.7. When a RE methodology with a specification language grounded temporal first-order logic is used[1], the following requirement $\mathsf{r} \in \mathsf{R}$ for TravelWeb states that all options that a service may be offering to the user should be visible to the first time user:

$$\mathsf{1stOpt} = (hasOptions(servID) \wedge firstTimeUser(servID, userID) \Rightarrow \diamond_{1s} showOptions(all, servID, userID))$$

Definition 11.8. *A* ***service requirement*** $r \in \mathcal{R}$ *is a constraint on service inputs or outputs that appears in at least one service request and there is a unique* $\mathsf{r} \in \mathsf{R}$ *such that there is a justified correspondence between it and* r*:* $\mathcal{R} = \{r \mid \exists_{\geq 1} \ddot{s}, partOf(C, \ddot{s}), r \in C \text{ and } \exists_1 \mathsf{r} \in \mathsf{R}, r \triangleq \mathsf{r}\}$.

Example 11.9. Any Serv that performs the visualization of options that other services offer to the user of TravelWeb when booking obeys the following service requirement:

$r =$ (**input:**servID $\notin$ userID.visited $\wedge$ servID.options $\neq \emptyset$;
output:thisService.show = servID.options)

[1] Assuming, for simplicity, a linear discrete time structure, one evaluates the formula for a given history (i.e., sequence of global system states) and at a certain time point. The usual operators are used: for a history H and time points i, j, $(H, i) \models \circ\phi$ iff $(H, next(i)) \models \phi$; $(H, i) \models \diamond\phi$ iff $\exists j > i, (H, j) \models \phi$; $(H, i) \models \Box\phi$ iff $\forall j \geq i, (H, j) \models \phi$. Mirror operators for the past can be added in a straightforward manner. Operators for eventually $\diamond$ and always $\Box$ can be decorated with duration constraints, e.g., $\diamond_{\leq 5s}\phi$ indicates that ϕ is to hold some time in the future but not after 5 seconds. To avoid confusion, note that $\rightarrow$ stands for implication, while $\phi \Rightarrow \psi$ is equivalent to $\Box(\phi \rightarrow \psi)$. For further details, see, e.g., [318]

Definition 11.10. *A **quality parameter** $q \in \mathcal{Q}$ describes stakeholder preferences over alternative system behaviors.* $q = \langle \mathsf{Name}, \mathsf{Type}, \mathsf{Target}, \mathsf{Threshold}, \mathsf{Current}, \mathsf{Stakeholder} \rangle$*, where* Name *is the unique name for the variable; values in* Type *follow usual variable taxonomies (i.e., nominal, ordinal, interval, and ratio) to indicate the type of the variable;* Target *gives a unique or a set of desired values for the variable;* Threshold *carries the worst acceptable values;* Current *contains the current value or average value over some period of system operation; and* Stakeholder *carries names of the stakeholders that agree on the various values given for the variable.*

Example 11.11. The following quality parameters can be defined on the Serv which visualizes the options of other services to the TravelWeb user when booking:

$q_1 = \langle \mathsf{ShowDelay}, \mathsf{Ratio}, \mathsf{500ms}, \mathsf{1s}, \mathsf{780ms}, \mathsf{MaintenanceTeam} \rangle$
$q_2 = \langle \mathsf{OptionsPerScreen}, \mathsf{Ratio}, \{3,4,5\}, 7, (\mathsf{all}), \mathsf{UsabilityTeam} \rangle$
$q_3 = \langle \mathsf{OptionsSafety}, \mathsf{Nominal}, \mathsf{High}, \mathsf{Med}, \mathsf{Low}, \mathsf{MaintenanceTeam} \rangle$ $q_4 = \langle \mathsf{BlockedOptions}, \mathsf{Ratio}, 0, (\geq 1), 0, \mathsf{MaintenanceTeam} \rangle$

As quality parameters usually cannot be satisfied to the ideal extent simultaneously, the preference specification contains information on priority and positive or negative interaction relationships between quality parameters. Prioritization assists when negotiating trade-offs, while interactions indicate trade-off directions between parameters.

Definition 11.12. *The **preferences specification** is the tuple* $\mathcal{P} = \langle \succ, \mathcal{P}^{\succ}, \mathcal{P}^{\pm} \rangle$ *where:*

- *"$\succ$" is a priority relation over quality parameters. The set $\mathcal{P}^{\succ}$ contains partial priority orderings, specified as $(q_i \succ q_j, \mathsf{Stakeholder}) \in \mathcal{P}^{\succ}$ where q_i carries higher priority than q_j, and* Stakeholder *contains the names of the stakeholders agreeing on the given preference relation. Higher priority indicates that a trade-off between the two quality parameters will favor the parameter with higher priority.*
- *The set $\mathcal{P}^{\pm}$ contains interaction pairs. Such a pair indicates that a given variation of the value of a quality parameter results in a variation of the value of another quality parameter. The BNF-like syntax for $p \in \mathcal{P}^{\pm}$ is:*

$$p ::= \left(q_1 \overset{b_1 \Rightarrow b_2}{\longleftrightarrow} q_2\right) \mid \left(q_1 \overset{b_1 \Rightarrow b_2}{\longleftrightarrow} q_2\right) @\phi \mid \left(q_1 \overset{f}{\longleftrightarrow} q_2\right) \mid \left(q_1 \overset{f}{\longleftrightarrow} q_2\right) @\phi \tag{4}$$

The syntax above builds on that defined in Eq.3: b is taken from Eq.3. The semantics for b are as in the cited equation. $q_1 \overset{b_1 \Rightarrow b_2}{\longleftrightarrow} q_2$ indicates that changing the value of the quality parameter q_1 by or to b_1 necessarily leads the value of the parameter q_2 to change for or to b_2. As the interaction may only apply when particular conditions hold, a non-empty condition ϕ can be added to indicate when the interaction applies. The condition is written in the same language as service requirements. When the relationship between the values of two quality parameters can be described with a function, f gives that functional relationship.

Example 11.13. Starting from the quality parameters in Ex.11.11, the following is a fragment of the preferences specification:

$p_1 = \left(\mathsf{OptionsPerScreen} \overset{+1 \Rightarrow +60ms}{\longleftrightarrow} \mathsf{ShowDelay}\right)$
$@(\mathsf{OptionsPerScreen} > 4)$
$p_2 = \left(\mathsf{BlockedOptions} \overset{\geq 1 \wedge \leq Max-1 \Leftrightarrow Med}{\longleftrightarrow} \mathsf{OptionsSafety}\right)$

Above, p_1 indicates that increasing the number of options per screen by 1 increases the delay to show options to the user by 60ms, this only if the number of options to show is above 4. p_2 states that the number of blocked options between 1 and and the maximum of options to show is associated with the Med level of options safety.

Information contained in a preferences specification is usually not explicitly available in established RE methodologies: trade-offs and negotiations are performed during RE so that the resulting specification integrates the outcomes of these activities. In an AOSS, however, trade-offs and negotiations are performed continually so that the preferences specification remains important at runtime.

To convert the content of R into fragments of $\mathcal{R}, \mathcal{Q}, \mathcal{P}$, one either proves that correspondence or justifies it. Proof might be possible if precise correspondence can be established between the syntax and semantics of the specification language used for the former and that of the latter. Since this is rarely possible in realistic settings, we introduce the notion of justified correspondence: intuitively, $\mathsf{r} \in \mathsf{R}$ justifiedly corresponds to $\{r\} \subseteq \mathcal{R} \vee \{q\} \subseteq \mathcal{Q} \vee \{p\} \subseteq \mathcal{P}$ if the engineer and/or the stakeholders can find a justification for that correspondence. Justified correspondence relation strongly relies on the notion of argument and justification process. We have argued for the use of structured argumentation and justification in RE

in a separate paper [150]; we therefore omit herein the detailed definitions of the argument concept, the relationships between arguments, and the argumentation process. Only the necessary notions are mentioned, while the interested reader is referred to [150].

Definition 11.14. *A **justified correspondence** exists between $\phi \in \mathsf{R}$ and $\psi \in \mathcal{R} \cup \mathcal{Q} \cup \mathcal{P}$, i.e., $\phi \triangleq \psi$ iff there is a justification $\langle P, \phi \triangleq \psi \rangle$, i.e., $\phi \triangleq \psi$ iff $\exists \langle P, \phi \triangleq \psi \rangle$.*

Recall from the above that the justified correspondence is a form of mapping in which very few assumptions are made on the precision and formality of the languages being mapped. This entails the usual difficulties (as those encountered in ontology mapping, see, e.g., [160]) regarding conversion automation and the defeasibility of the constructed mappings, making DRAM somewhat elaborate to apply in its current form. Defeasibility does, however, carry the benefit of flexibility in building and revising mappings.

Definition 11.15. *A **justification** $\langle P, c \rangle$ is an argument that remains undefeated after the justification process.*

An argument $\langle P, c \rangle$ is a set of consistent premises P supporting a conclusion c. The language in which the premises and the conclusion are written is enriched with the binary relation $\hookrightarrow$. The relation $\hookrightarrow$ between formulae α and β is understood to express that "reasons to believe in the antecedent α provide reasons to believe in the consequent β". In short, $\alpha \hookrightarrow \beta$ reads "α is reason for β" (see, [289] for details). Formally then, P is an argument for c, denoted $\langle P, c \rangle$, iff[2]: (1) $K \cup P \mathrel{|\!\sim} c$ (K and P derive c); (2) $K \cup P \not\vdash \bot$ (K and P are consistent); and (3) $\nexists P' \subset P, K \cup P' \mathrel{|\!\sim} c$ (P is minimal for K).

Up to this point, the concepts needed in DRAM have been introduced. The remainder of this section describes the techniques in DRAM that use the given concepts in the aim of constructing the dynamic requirements specification.

Technique 17. The *justification process* consists of recursively defining and labeling a *dialectical tree* $\mathcal{T} \langle P, c \rangle$ as follows:

1. A single node containing the argument $\langle P, c \rangle$ with no defeaters is by itself a dialectical tree for $\langle P, c \rangle$. This node is also the root of the tree.
2. Suppose that $\langle P_1, c_1 \rangle, \ldots, \langle P_n, c_n \rangle$ each defeats[3] $\langle P, c \rangle$. Then the dialectical tree $\mathcal{T} \langle P, c \rangle$ for $\langle P, c \rangle$ is built by placing $\langle P, c \rangle$ at the root of the tree and by making this node the parent node of roots of dialectical trees rooted respectively in $\langle P_1, c_1 \rangle, \ldots, \langle P_n, c_n \rangle$.
3. When the tree has been constructed to a satisfactory extent by recursive application of steps 1) and 2) above, label the leaves of the tree *undefeated* (U). For any inner node, label it undefeated if and only if every child of that node is a *defeated* (D) node. An inner node will be a defeated node if and only if it has at least one U node as a child. Do step 4) below after the entire dialectical tree is labeled.
4. $\langle P, c \rangle$ is a *justification* (or, P justifies c) if and only if the node $\langle P, c \rangle$ is labelled U.

Example 11.16. Fig.17 contains the dialectical tree for the justified correspondence $\mathsf{1stOpt} \triangleq r$, where r is from Ex.11.7 and r from Ex.11.9. To simplify the presentation of the example, we have used both formal and natural language in arguing. More importantly, notice that the correspondence $\mathsf{1stOpt} \triangleq r$ is unjustifed, as it is defeated by an undefeated argument containing information on a quality parameter and a fragment of the preferences specification. A justified correspondence such as, e.g., $\mathit{firstTimeUser}(\mathit{servID}, \mathit{userID}) \triangleq \mathit{servID} \notin \mathit{userID.visited}$, becomes an update rule, i.e., $(\mathit{firstTimeUser}(\mathit{servID}, \mathit{userID}) \triangleq \mathit{servID} \notin \mathit{userID.visited}) \in \mathcal{U}$. Having established that justified correspondence, the service requirement is taken to correspond to the given initial requirement until the justified correspondence is defeated. Elements of the argument repository correspond to the argument structure shown in Fig.17.

[2] Some background: Let A a set of agents (e.g., stakeholders) and the first-order language $\mathcal{L}$ defined as usual. Each agent $a \in A$ is associated to a set of first-order formulae K_a which represent knowledge taken at face value about the universe of discourse, and Δ_a which contains defeasible rules to represent knowledge which can be revised. Let $K \equiv \bigcup_{a \in A} K_a$, and $\Delta \equiv \bigcup_{a \in A} \Delta_a$. "$\mathrel{|\!\sim}$" is called the *defeasible consequence* [289] and is defined as follows. Define $\Phi = \{\phi_1, \ldots, \phi_n\}$ such that for any $\phi_i \in \Phi$, $\phi_i \in K \cup \Delta^{\downarrow}$. A formula ϕ is a defeasible consequence of Φ (i.e., $\Phi \mathrel{|\!\sim} \phi$) if and only if there exists a sequence $B_1, \ldots, B_m$ such that $\phi = B_m$, and, for each $B_i \in \{B_1, \ldots, B_m\}$, either B_i is an axiom of $\mathcal{L}$, or B_i is in Φ, or B_i is a direct consequence of the preceding members of the sequence using modus ponens or instantiation of a universally quantified sentence.

[3] Roughly (for a precise definition, see [289]) the argument $\langle P_1, c_1 \rangle$ *defeats at* c an argument $\langle P_2, c_2 \rangle$ if the conclusion of a subargument $\langle P, c \rangle$ of $\langle P_2, c_2 \rangle$ contradicts $\langle P_1, c_1 \rangle$ and $\langle P_1, c_1 \rangle$ is more specific (roughly, contains more information) than the subargument of $\langle P_2, c_2 \rangle$.

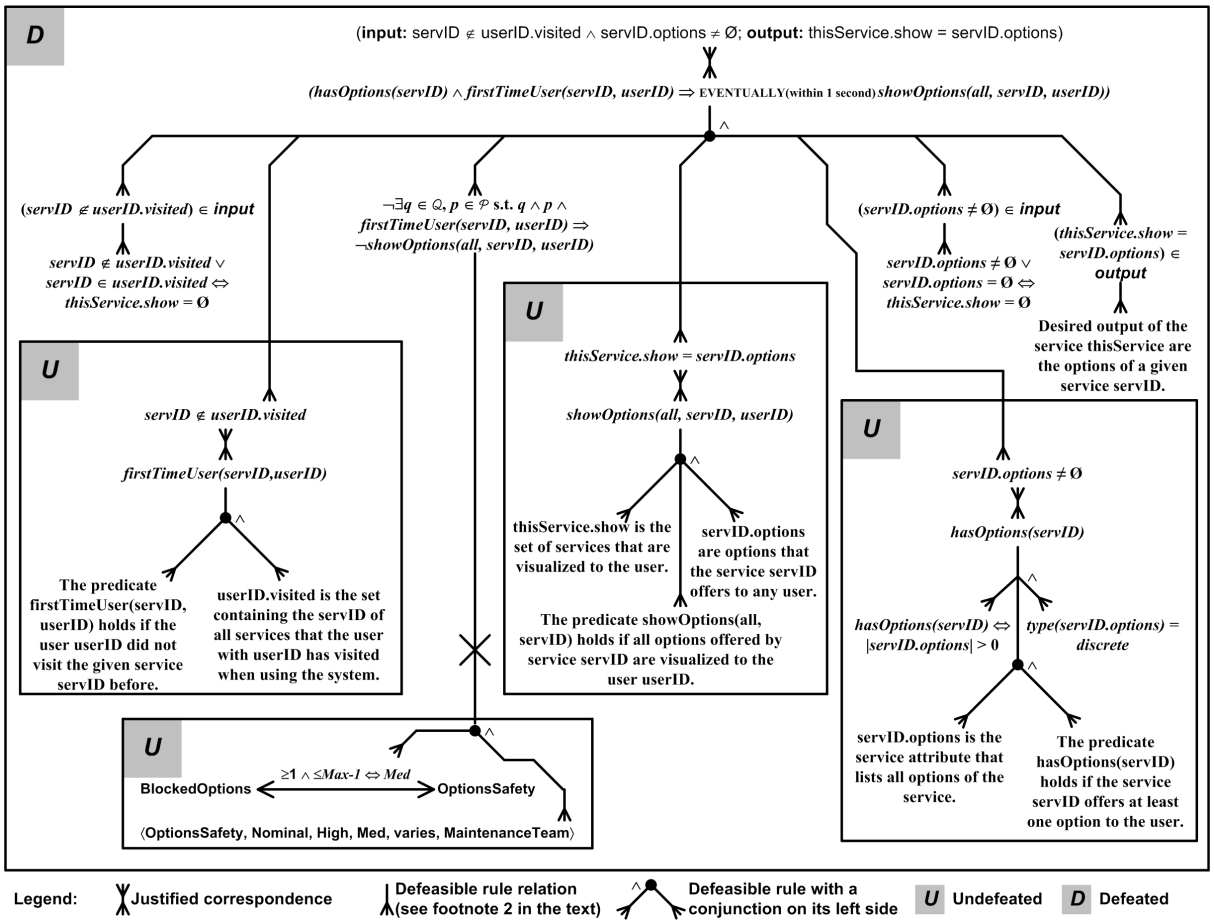


Fig. 17. Example output of the justification process related to Examples 11.7 and 11.9.

Technique 18. DRAM proceeds in the following steps to build the dynamic requirements specification:

1. Starting from the static requirements specification R, select a fragment r ∈ R of that specification that has not been converted into a fragment in $\mathcal{R}$, $\mathcal{Q}$, and/or $\mathcal{P}$.
2. Determine the service requirement and/or quality parameter information that can be extracted from r using Techniques 19 and 20.
3. Write down the obtained $r \in \mathcal{R}$, $q \in \mathcal{Q}$, and/or $p \in \mathcal{P}$ information, along with arguments and justifications used in mapping r into r and/or q. Each justified correspondence obtained by performing the step 2. above is written down as an update rule $u \in \mathcal{U}$.
4. Verify that the new arguments added to $\mathcal{A}$ do not defeat justifications already in $\mathcal{A}$; revise the old justifications if needed.

Technique 19. If r is a functional requirement (i.e., it specifies a behavior to perform), focus is on mapping r onto service requirements. Consider, e.g., the following requirement: Each user of TravelWeb expects a list of available flights for a destination to be shown within 5 seconds after submitting the departure and destination city and travel dates.

$$available(depC, depD, arrC, arrD, flight) \\ \wedge\ correctFormat(depC, depD, arrC, arrD) \\ \Rightarrow \diamond_{5s} shown(searchResults, flight)$$

Starting from the above functional requirement:

1. Identify the various pieces of data that are to be used (in the example: $depC, depD, arrC, arrD, flight$) and those that are to be produced ($searchResults$) according to the requirement.
2. Find services that take the used data as input and give produced data at output (e.g., FlightSearch Serv, w_{fs} s.t. $\{depC, depD, arrC, arrD, flight\} \subseteq I \wedge searchResults \in O$).
3. Determine whether the service requirements available on inputs correspond to the conditions on input data in the requirement, and perform the same for output data (i.e., check if there is a justified correspondence between input/output service requirements and conditions in the relevant requirements in R).

If constraints do not correspond (justified correspondence does not apply), map the conditions from the requirement in R into constraints on inputs and/or outputs, and write them down as service requirements. If there is no single service that satisfies the requirement (i.e., step 2. above fails), refine the requirement (i.e., brake it down into and replace with more detailed requirements)—to refine, apply techniques provided in the RE methodology being used.

4. Use Technique 20 to identify the quality parameters and preferences related to the service requirement obtained by applying the present technique.

Technique 20. If r is a nonfunctional requirement (i.e., describes *how* some behavior is to be performed, e.g., by optimizing a criterion such as delay, security, safety, and so on), the following approach is useful:

1. Find quality parameters that describe the quality at which the inputs and outputs mentioned in a particular service requirement are being used and produced. In the example cited in Technique 19, the delay between the moment input data is available and the moment it is displayed to the user can be associated to a quality parameter which measures the said time period.
2. Following Def.11.10, identify the various descriptive elements for each quality parameter. Use R as a source for the name, target and threshold value, and relevant stakeholders. If, e.g., Tropos is employed to produce R, softgoals provide an indication for the definition of quality parameters.
3. For each quality parameter that has been defined, specify priority and preferences. Initial preferences data used in trade-offs can be obtained by performing test runs.

If deidealization is to be performed through a probabilistic approach, Letier and van Lamsweerde's approach [198] can be employed as the Def.11.10 is compatible with their notion of quality variable.

11.4 Related Work

Engineering requirements and subsequently addressing completeness concerns for AOSS has only recently started to receive attention in RE research. Without providing solutions, Berry and colleagues [24] argue in a note that, while much effort is being placed in enabling adaptive behavior, few have dealt with how to ensure correctness of software before, during, and after adaptation, that is, at the RE level. They do, however, recognize that RE for such systems is not limited to the initial steps of the system development process, but is likely to continue in some form over the entire lifecycle of the system. Zhang and Cheng [342] suggest a model-driven process for adaptive software; they represent programs as state machines and define adaptive behaviors usually encountered in adaptable systems as transitions between distinct state machines, each giving a different behavior to the system. Being situated more closely to the design phase of development than to RE, Zhang and Cheng's process has been related [44] to the KAOS RE methodology by using A-LTL instead of temporal logic employed usually in KAOS. A-LTL, introduced by Zhang and Cheng [341], extends linear temporal logic with a binary operator "$\stackrel{\Omega}{\frown}$" used to indicate, for $\phi \stackrel{\Omega}{\frown} \psi$ that if a state sequence satisfies $\phi \stackrel{\Omega}{\frown} \psi$, then there is a state in that sequence before which ϕ is satisfied and after which ψ is satisfied. In the extended KAOS, a requirement on adaptation behavior amounts to a goal refined into two sequentially ordered goals, whereby the first in the sequence specifies the conditions holding in the state of the system before adaptation while the second goal gives those to hold in the state after adaptation. Lapouchnian and colleagues [190] argue in a note that goal models suggested in, e.g., KAOS and Tropos RE frameworks could be employed in engineering requirements for autonomic systems which exhibit adaptation behavior. They argue that goals provide a useful abstraction for comparing alternative system behaviors and relating the observed behaviors to stakeholder expectations.

The present paper differs from cited efforts in terms of concerns being addressed and the subsequent proposal. The suggested separation onto Service, Coordination, and Client RE for AOSS usefully delimits the concerns and focus of the RE effort when dealing with AOSS. While, e.g., Zhang and Cheng's proposal concerns service RE and to some extent Coordination RE, DRAM addresses Client RE. The notion of dynamic requirements specification, along with the associated concepts and techniques used to update the requirements specification according to changes in the system at runtime is novel with regards to the cited research.

11.5 Conclusions and Future Work

This paper addresses the difficulties in engineering requirements for adaptable and open service-oriented systms. In summary, the contributions are: (1) It is observed that RE of AOSS involves the specification of requirements that may vary at runtime. Consequently, a separation between RE activities based

on the variability of requirements is introduced: namely, the RE for individual services (Service RE) is distinguished from RE of mechanisms for service coordination (Coordination RE), and the RE of parameters and constraints guiding coordination (Client RE). (2) After identifying Client-level requirements as the most variable of the three, the Dynamic Requirements Adaptation Method is suggested to complement a traditional RE methodology in engineering requirements at Client RE. The method assists the requirements engineer in defining the dynamic requirements specification which is grounded in the initial requirements specification and complemented with update rules to ensure that the runtime changes in the AOSS are reflected in the dynamic specification. (3) By basing the update rules on arguments and justifications, DRAM remains flexible since very few assumptions are made as to the specification language of the RE methodology that DRAM complements, and the specification language(s) used at the coordination level of the AOSS.

Automation of the various techniques in DRAM is the focus of current work, namely through patters of justifications for justified correspondences. The use of reinforcement learning algorithms for partial automation of argumentation is also being explored. To facilitate practical application, the construction of sets of mappings specialized for Tropos and KAOS is also considered.

12

Dynamic Web Service Composition within a Service-Oriented Architecture

Abstract. Increasing automation requires open, distributed, service-oriented systems capable of multicriteria-driven, dynamic adaptation for appropriate response to changing operating conditions. We combine a simple architecture with a novel algorithm to enable openness, distribution, and multi-criteria-driven service composition at runtime. The service-oriented architecture involves mediator web services coordinating other web services into compositions necessary to fulfil user requests. By basing mediator services' behavior on a novel multicriteria-driven (including quality of service, deadline, reputation, cost, and user preferences) reinforcement learning algorithm, which integrates the exploitation of acquired knowledge with optimal, undirected, continual exploration, we ensure that the system is responsive to changes in the availability of web services. The reported experiments indicate the algorithm behaves as expected and outperforms two standard approaches.

12.1 Introduction

Managing the complexity of systems is considered a key challenge in computing (e.g., [307, 308]) and is addressed through various approaches aimed at increased automation. *Service-oriented architectures* (SOA) are expected to enable the provision of a large number of distinct and competing web services which the prospective users will be able to choose dynamically in the aim of receiving at all times optimal offerings for their purposes. SOA ought to be *open* to permit many services to participate and avoid biased selection. Openness commits SOA to a *distributed* architecture for entering and leaving resources are bound to be decentralized. To be *adaptable*, service provision should be performed by dynamically selecting and composing the participating services according to multiple quality criteria, so that the users continually receive optimal results. Efficiency and flexibility that such systems are expected to exhibit are valuable given the pressing complexity concerns.

Building systems that exhibit the given characteristics involves many issues already treated to varying degrees in the literature: among them, infrastructure for services (e.g., [49]), description of services (e.g., [60]), matchmaking between descriptions and requests (e.g., [22]), and so on. *This paper focuses on the composition of services under the constraints of openness, resource distribution, and adaptability to changing web service availability w.r.t. multiple criteria and constraints.* To enable such system characteristics, a fit between the system architecture and service composition behavior is needed, that is: (1) To support openness, few assumptions can be made about the behavior of the web services that may participate in compositions. It thus seems reasonable to expect service composition responsibility not to be placed on any web service: the architecture ought to integrate a special set of web services, the madiators, that coordinate service composition. (2) To allow the distribution of web services, no explicit constraints should be placed on the origin of entering services. (3) To enable adaptability, mediator behavior should be specified along with the architecture. (4) Since there is no guarantee that web services will execute tasks at quality levels advertised by the providers, composition should be grounded in empirically observed service quality and such observation be executed by the mediators. (5) The variety of stakeholder expectations requires service composition to be driven by multiple criteria. (6) To ensure continuous adaptability at runtime, composition within the architecture should involve continual observation of service quality, the use of available information to account for service behavior, *and* exploration of new options to avoid excessive reliance on historical information.

Contributions. To respond to the requirements (1)–(6) above, our proposal is to combine a SOA with a novel algorithm for mediator behavior. The architecture organizes web services into groups, called "service centers". Each service center specializes in the provision of a composite service (i.e., the composition

of web services). Within each center, a single "mediator web service" receives service requests. Upon reception, the mediator decides how to compose the available web services into the composite service to be delivered, and this depending on observed prior quality while accounting for anticipated quality of newly available services. As no constraints are placed beyond organizing service delivery through mediators, the architecture places no constraints on openness and distribution. Within such an architecture, each mediator plays a critical role: it organizes work by composing services, negotiates with individual services, and observes their quality in order to adjust compositions in the aim of continually optimizing quality criteria. Mediators' composition behavior is guided by a novel multicriteria-driven (including QoS, deadline, reputation, cost, and user preferences) reinforcement learning (RL) algorithm, called *Randomized Reinforcement Learning (RRL)* (first introduced in [4], and specialized here), which integrates the exploitation of acquired knowledge with optimal, undirected, continual exploration. Instead of relying on experience only, exploration allows the mediator to continually explore the pool of agents available for task allocation, thus ensuring the responsiveness of the system to change. The reported experiments show the RRL outperforming two standard exploration methods, namely, ϵ-*greedy* and *naive Boltzmann*. Because the principal contributions of the present paper are the architecture and the algorithm, no specific commitments are made on, e.g., task allocation or negotiation protocols, to ensure the results are generic. Such choices are left to the designer.

Organization. The remainder of the paper starts with the description and discussion of the architecture (§12.2). Detail of the RRL algorithm is then presented (§12.3), and experimental evaluation and comparison of the algorithm with standard competing solutions are reported (§12.4). Finally, related work is discussed (§12.5) before closing the paper with conclusions and pointers to future work (§12.6).

12.2 Service Center Architecture

The *Service Center Architecture* (SCA) groups services into *Service Centers* (SC). Each SC contains one *Mediator Web Service* (MWS) and all distinct *Web Services* (WS) needed to provide a *Composite Web Service* (CWS) corresponding to the *Service Request* (SReq; e.g., finding the itinerary between two physical addresses as common in map applications, booking a flight, searching for files, and so on) originating from the user (who can be an WS or an MWS). The MWS composes WS by observing past quality of individual WS, then subsequently using (and updating) this information through the RRL (see, §12.3). Combining the SCA and the RRL brings the following benefits: (a) Adaptability to changes in the availability and/or performence levels of WS is ensured, as the algorithm accounts for actual quality observed in the past and explores new compositions as new WS appear. (b) Continuous optimization over various criteria in the algorithm allows different criteria to guide service composition, while exploitation *and* exploitation ensure MWS continually revise composition choices. (c) By localizing composition decisions at each MWS, the architecture remains decentralized and permits distribution of resources. (d) The architecture and algorithm place no restrictions on the openness of, or resource distribution in the system.

Because a number of CWS involve the execution of common services, the presence of generic services (i.e., those whose frequency of execution is above some externally fixed threshold) makes it possible to pool the information about idle WS executing the generic services into the same center, called the *Support Service Center* (SSC). The effects sought in doing so are (i) ensuring the availability of WS which execute frequently needed tasks; (ii) having an MWS dedicated to identifying and scheduling the work of WS which are most appropriate for generic services in various SC; and (iii) avoiding communication between various MWS regarding generic WS, but instead centralizing relevant information on generic WS at one MWS. The remainder of this section revisits the SCA with more precision.

Definition 12.1. *A tuple* $\langle I, O, \tilde{\boldsymbol{s}}^{QoS}, \tilde{s}^{cost}, s\rangle$ *is called a Web Service (WS) w. I and O specify, respectively, the inputs and the outputs of the service. The WS advertises its capability to provide the service to the QoS levels given by the vector* $\tilde{\boldsymbol{s}}^{QoS}$ *and cost* $\tilde{s}^{cost}$*. The structure of* $\tilde{\boldsymbol{s}}^{QoS}$ *is determined by the employed QoS ontology. s is a specification of all additional properties of the service irrelevant for the present discussion, yet necessary when building a system.*

Definition 12.2. *A Mediator Web Service (MWS)* w_c^{MWS} *in a service center* $c \in \mathcal{C}$ *is a WS capable of executing the RRL algorithm (*$\mathcal{A}$*), denoted:* $\mathcal{A} \in w_c^{MWS}$*.*

MWS in SC and in SSC both behave according to the algorithm; the difference being that the MWS in SSC allocates WS to various SC where the local MWS need them.

Definition 12.3. *$\langle s^N, s^E, servTransit, servState, s^\iota\rangle$ is a Composite Web Service (CWS) $\breve{w}$. (s_j^N, s_j^E) defines a directed acyclic graph. In the terminology of the algorithm, a node represents a "state" and an edge a "transition", that is, a node is a description of inputs or outputs of a service, while an edge represents a task[1]. The two functions label nodes and edges with WS information: $servTransit : s^E \longmapsto \mathbb{W}$ is a partial function returning the WS for a given edge in the graph ($\mathbb{W}$ is by convention the set of all WS), while $servState : s^N \longmapsto \{I\}_{w\in\mathbb{W}} \cup \{O\}_{w\in\mathbb{W}}$ maps each edge to inputs or outputs of WS. The WS on an edge must have the inputs and outputs corresponding to conditions given, respectively, on its origin and its destination node.*

A service understood here as a process, composed of a set of tasks (accomplished by WS) ordered over the graph representing the service. The functional specification of the service, i.e., s^ι is not of interest here, but involves in practice, e.g., a specification of interfaces, and other implementation considerations. Requesting a service involves the specification of expected QoS, in addition to a deadline for providing the service, minimal level of reputation for agents that are to participate in service execution, the maximal monetary cost, and explicit user preferences on agents to select (e.g., users may prefer globally the services of some providers over others, regardless of actual quality—this may occur with preferential treatment resulting from environment constraints such as, e.g., legal constracts on cooperation between organizations and/or individuals).

Definition 12.4. *$\hat{s}_j = \langle \breve{w}, \boldsymbol{s}^{QoS}, s^D, \boldsymbol{s}^R, s^{cost}, \boldsymbol{s}^{pref}\rangle$ is called a Service Request (SReq), where:*

- *$\breve{w}$ is the composite service to provide.*
- *$\boldsymbol{s}^{QoS}$ specifies expected qualities and their required level. Its definition follows an QoS ontology, such as, e.g., the FIPA QoS ontology specification [91]. Whatever the specific QoS ontology, expected qualities are likely to be specified as (at least) $\boldsymbol{s}^{QoS} = \langle (p_1, d_1, v_1, u_1), \ldots, (p_r, d_r, v_r, u_r)\rangle$, where:*
  - *p_k is the name of the QoS parameter (e.g., connection delay, standards compliance, and so on).*
  - *d_k gives the type of the parameter (e.g., nominal, ordinal, interval, ratio).*
  - *v_k is the set of desired values of the parameter, or a constraint $<, \leq, =, \geq, >$ on its value.*
  - *u_k is the unit of the property value.*
- *s^D is a deadline, specified as a natural.*
- *$\boldsymbol{s}^R = \langle \hat{R}_k^{a,w_i}, \hat{R}_k^{a,w_{i+1}}, \ldots\rangle$ specifies minimal levels of reputation over quality parameters that any WS must satisfy. It is not necessary to specify reputation for all qualities over all WS, selective reputation expectations are admitted.*
- *s^{cost} is the maximal monetary cost the user requesting the service is ready to pay to obtain the CWS.*
- *$\boldsymbol{s}^{pref}$ is a set of expressions that constrain the pool of potential WS to which the MWS can use in the composition.*

Definition 12.5. *Reputation R_k^{a,w_i} of a WS w_i over the QoS parameter k is:*

$$R_k^{a,w_i} = \frac{1}{n-1}\sum_{i=1}^{n}\left[\left(v_k^{Adv} - \hat{v}_k^i\right)^2 \delta^{-time(\hat{v}_k^i)}\right]$$

where $time()$ returns the time of observation (a natural, 1 for the most recent observed value, $time(\hat{v}_k^i) > 1$ for all other) and δ is the dampening factor for the given quality (can be used with $time()$ to give less weight to older observations). We assume that the advertised quality for w_i is $\tilde{\boldsymbol{s}}_{w_i}^{QoS} = \langle (p_1, d_1, v_1^{Adv}, u_1), \ldots, (p_r, d_r, v_r^{Adv}, u_r)\rangle$, and that n observations $\hat{v}_k^i, 1 \leq i \leq n$ have been made over a quality parameter k.

Reputation and trust receive considerable attention in the literature (e.g., [334, 217]). The ideas underlying Maximilien and Singh's approach [217] are followed, with two caveats: they use "trust" to select services from a pool of competing services and exploit user-generated opinions to calculate reputation, whereas herein WS are selected automatically and reputation is generated by comparing WS behavior observed by the MWS and the advertised behavior of the WS. Reputation is used here instead of trust since no user opinions are accounted for.

Definition 12.6. *$\langle w_c^{MWS}, \breve{w}, W_{\breve{w},c}\rangle$ is called a Service Center (SC) $c \in \mathcal{C}$, where w_c^{SM} is the MWS in the given center, $\breve{w}$ is the CWS that the SC is to provide, and $W_{\breve{w},c}$ is the set of WS involved in the composition chosen by the w_c^{MWS} to deliver $\breve{w}$.*

[1] "Task" refers to the transformation of inputs to outputs that a WS can execute and is necessary in providing a CWS.

Definition 12.7. *A Service Support Center (SSC) c_{SSC} is a service center in which all WS are generic, i.e., $\langle w_c^{MWS}, \breve{w}, W_{\breve{w},c} \rangle$, with the additional constraint that $\forall w_i \in W_{\breve{w},c}, genericTask(w_i, P, x) = true$, where $genericTask : \mathbb{W} \times timePeriod \times \mathbb{N} \longmapsto \{true, false\}$ returns true if a given task has been executed over a given time period for more times than the specified threshold $x \in \mathbb{N}$.*

Definition 12.8. *SCA is a set containing one SSC and $m \geq 1$ SC: $SCA = \{c_1, \ldots, c_m, c_{SSC}\}$.*

12.2.1 Role of the Algorithm

The SCA can be argued open and distributed, for it places no constraints other than centralizing WS composition at MWS. The SCA *cannot* be argued adaptable and optimally responsive to service requests without the algorithm. The RRL presented in §12.3 defines the behavior of MWS by specifying how the mediator-specific service, $t_{\mathcal{A}}$, proceeds to compose WS for CWS delivery by optimizing one or more service request criteria (referred to as r in the remainder), while taking the remaining criteria (vector $\mathbf{s}$ containing all criteria from the service request other than r) as hard constraints. Returning to how a service request is specified in SCA (see, Def.12.4) it is apparent that many criteria can be accounted for when selecting among alternative WS compositions, hence qualifying the algorithm as *multicriteria-driven* within the present paper. As decision making in presence of multiple criteria permits arguing for, and accepting various decision rules (which differ on, e.g., how criteria are aggregated), the algorithm is constructed to leave much freedom to the designer in actual implementation. Moreover, it does not require full specification of all possible criteria for each service—instead, it is up to the users to choose what criteria to specify. The algorithm thus optimizes a single normalized (i.e., taking values in the interval $[0, 1]$) variable, leading to three approaches to specifying this variable and the remaining hard constraints the algorithm takes as input when running:

1. If the user prefers to have one criterion optimized (this being either a ratio-type[2] QoS parameter in $\mathbf{s}^{QoS}$, or s^D, or reputation from $\mathbf{s}^R$, or s_j^{cost}), expected values over the remaining criteria will be treated by the algorithm as hard constraints, whereby task allocations which violate hard constraints will be eliminated by the algorithm.
2. If the user prefers to have several criteria optimized, it is necessary to provide an aggregation function for the relevant criteria, so that the result of the function is what the algorithm will optimize (i.e., r is an aggregate). Guidelines for aggregation functions can be found in, e.g., [325]. Non-aggregated criteria are treated as hard constraints (i.e., $\mathbf{s}$, see §12.3.2 below).
3. A third option is to have the mediator suggest alternative allocations and the user chooses the one to apply. Presence or absence of this approach depends entirely on the choices of the designer, as it does not affect the formulation of the algorithm—it is essentially the first option above, with the nuance that the user asks the mediator to provide a list of optimal allocations for each criteria, and then selects manually.

12.3 Randomized RL Algorithm

Whenever the environment is changing, new WS outperforming the available ones can appear. The exploitation of acquired knowledge about the quality of WS can therefore be usefully combined with the exploration of composition options arising with change in operating conditions. Formally, exploration is the association of a probability distribution to the set of available WS in each state (i.e., choice randomization). Usually, the exploration/exploitation issue is addressed by periodically readjusting the policy for choosing actions (here, such action consists of deciding to use a WS in a composition for delivering a CWS) and re-exploring up-to-now suboptimal paths [224, 303]. Such a strategy is, however, suboptimal because it does not account for exploration. The RRL algorithm introduced in [4] is adapted herein to dynamic service composition while (i) optimizing criteria, (ii) satisfying the hard constraints, (iii) learning about the quality of new WS so as to continually adjust composition, and (iv) exploring new composition options. The exploration rate is quantified with the Shannon entropy associated to the probability distribution of allocating a *task* (i.e., some part of a requested CWS) to a WS. This permits the continual measurement and control of exploration.

Returning to the conceptualization of the SCA, the problem the algorithm resolves is a composition problem, whereby the MWS ought to choose the WS which are to execute tasks in a given CWS. By

[2] Nominal or ordinal QoS parameters that cannot be converted to ratio form give rise to hard constraints.

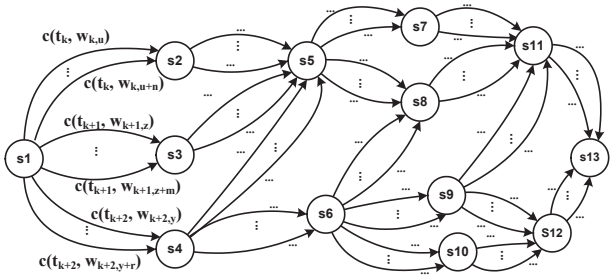


Fig. 18. CWS as a labeled hypergraph.

conceptualizing the CWS as a labeled directed acyclic graph in Def.12.3, the composition problem amounts to a deterministic shortest-path problem in a directed weighted hypergraph. The CWS is thus mapped onto a directed weighted hypergraph G where each node in G is a step in CWS provision and an edge in G corresponds to the allocation of a task t_k to a WS $w_{k,u}$, where u ranges over WS that can execute w_k according to the criteria set in the service request. Each individual allocation of a task to a WS incurs a cost $c(t_k, w_{k,u})$, whereby this "cost" is a function of the criterion (or aggregated criteria, as discussed in §12.2.1) formulated so that the minimization of cost correponds to the optimization of the criterion (i.e., minimization or maximization of criterion value). This criterion is the one the user chooses to optimize, whereas other criteria are treated as hard constraints. For illustration, consider the representation of a generic CWS as a hypergraph in Fig.18 where nodes are labeled with states of the service provision problem, and edges with costs of alternative task-to-WS allocations (for simplicity, only some labels are shown). Nodes are connected by several edges to indicate the presence of alternative allocations of the given task to WS. Any path from the starting node to the destination node is a potential allocation of tasks to WS (i.e., a WS composition). The CWS provision problem is thus a global optimization problem: learn the optimal complete probabilistic allocation that minimizes the expected cumulated cost from the initial node to the destination node while maintaining a fixed degree of exploration, and under a given set of hard constraints (specified in the service request). At the initial node in the graph (in Fig.18, node s1), no tasks are allocated, whereas when reaching the destination node (s13 in the same figure), all tasks are allocated.

12.3.1 RL Formulation of the Problem

At a state k of the CWS provision problem, choosing an allocation of t_k to $w_{k,u}$ (i.e., moving from k to another state) from a set of potential allocations $U(k)$ incurs a cost $c(t_k, w_{k,u})$. Cost is an inverse function of the criterion the user wishes to optimize (see, §12.2.1), say r. The cost can be positive (penalty), negative (reward), and it is assumed that the service graph is acyclic [54]. It is by comparing WS over estimated $\hat{r}$ values and the hard constraints to satisfy (see, §12.3.2) that task allocation proceeds. The allocation $(t_k, w_{k,u})$ is chosen according to a *Task Allocation policy (TA)* Π that maps every state k to the set $U(k)$ of admissible allocations with a certain probability distribution $\pi_k(u)$, i.e., $U(k)$: $\Pi \equiv \{\pi_k(u), k = 1, 2, \ldots, n\}$. It is assumed that: (i) once the action has been chosen, the next state k' is known deterministically, $k' = f_k(u)$ where f is a one-to-one mapping from states and actions to a resulting state; (ii) different actions lead to different states; and (iii) as in [26], there is a special cost-free *destination* state; once the service mediator has reached that state, the service provision process is complete.

Definition 12.9. *The degree of exploration E_k at state k is quantified as:*

$$E_k = -\sum_{u \in U(k)} \pi_k(u) \log \pi_k(u) \tag{5}$$

which is the entropy of the probability distribution of the task allocations in state k [66, 162]. E_k characterizes the uncertainty about the allocation of a task to a WS at k. It is equal to zero when there is no uncertainty at all ($\pi_k(i)$ reduces to a Kronecker delta); it is equal to $\log(n_k)$, where n_k is the number of admissible allocations at node k, in the case of maximum uncertainty, $\pi_k(i) = 1/n_k$ (a uniform distribution).

Definition 12.10. *The exploration rate $E_k^r \in [0, 1]$ is the ratio between the actual value of E_k and its maximum value: $E_k^r = E_k / \log(n_k)$.*

Fixing the entropy at a state sets the exploration level for the state; increasing the entropy increases exploration, up to the maximal value in which case there is no more exploitation—the next action is chosen completely at random (using a uniform distribution) and without taking the costs into account. Exploration levels of MWS can thus be controlled through exploration rates. Service provision then amounts to minimizing *total expected cost* $V_\pi(k_0)$ accumulated over all paths from the initial k_0 to the final state:

$$V_\pi(k_0) = E_\pi \left[\sum_{t=0}^{\infty} c(k_t, u_t) \right] \tag{6}$$

The expectation E_π is taken on the policy Π that is, on all the random choices of action u_i in state k_i.

12.3.2 Satisfying Hard Constraints

Hard constraints are satisfied by adopting a special hypergraph structure and task allocation process, detailed in this section and inspired by critical path analysis (see for instance [21]). As shown in Fig.18, each node of the graph represents the completion of a task and each edge the assignment of a WS to the specific task. Each path from the starting node (e.g., node s1 in Fig.18) to the destination node (node s13 in Fig.18) thus corresponds to a sequence of assigned tasks ensuring the completion of the CWS within the prescribed hard constraints. The model thus assumes that there are alternative ways for completing the CWS. The topology of the graph—i.e., the node structure and the tasks associated to edges between the nodes—is provided by the designer through the service definition, so that the graph is a graphical model of the different ways the service can be performed as a sequence of tasks. Each constraint will be of the form "cannot exceed a given predefined quantity" (upper bounds); for instance, the total duration along any path should not exceed some predefined duration. Extensions to interval constraints could be handled as well, but are not reported in this paper.

To illustrate allocation while maintaining the hard constraints satisfied, let $\mathbf{g}_{k_i}$ be the vector containing the largest values, for each quantity subject to a constraint, along any path connecting the starting node (called k_0) to node k_i, and $\mathbf{h}_{k_i}$ the vector containing the largest values, for each quantity subject to a constraint, along any path connecting node k_i to the destination node (called k_d). Let $\mathbf{s}^{QoS} = (s^1, s^2)$ be the vector containing hard constraints on two QoS criteria (for the sake of simplicity, two-dimensional criteria vectors are considered; extension to n-dimensional vectors is straightforward). It follows that $\mathbf{g}_{k_i}$ represents the worst $\mathbf{s}^{QoS}$ when reaching k_i, while $\mathbf{h}_{k_i}$ is the worst $\mathbf{s}^{QoS}$ for moving from k_i to k_d. Computing the two vectors is straightforward in dynamic programming (e.g., [21]):

$$\begin{cases} \mathbf{g}_{k_0} = \mathbf{0} \\ \mathbf{g}_{k_i} = max_{P(k_i) \to k_i} \{ \mathbf{s}^{QoS}_{P(k_i) \to k_i} + \mathbf{g}_{P(k_i)} \} \end{cases} \tag{7}$$

$$\begin{cases} \mathbf{h}_{k_d} = \mathbf{0} \\ \mathbf{h}_{k_i} = max_{k_i \to S(k_i)} \{ \mathbf{s}^{QoS}_{k_i \to S(k_i)} + \mathbf{h}_{S(k_i)} \} \end{cases} \tag{8}$$

where $P(k_i)$ is the set of predecessor nodes of k_i and $S(k_i)$ the set of successor nodes of k_i. When computing $\mathbf{g}_{k_i}$, the maximum is taken on the set of edges reaching k_i (i.e., $P(k_i) \to k_i$); while when computing $\mathbf{h}_{k_i}$, the maximum is taken on edges leaving k_i (i.e., $k_i \to S(k_i)$). $\mathbf{s}^{QoS}$ is the QoS criteria vector (s^1, s^2) for a WS j associated to an edge. Any vector $\mathbf{g}_{k_i} < \mathbf{s}_{max}$ and $\mathbf{h}_{k_i} < \mathbf{s}_{max}$ is acceptable since it does not violate the constraints (assuming $\mathbf{s}_{max} = (s^{1,max}, s^{2,max})$ contains upper bounds on hard constraints). Suppose then that the SM considers assigning a task on an edge between nodes k_i and k_j to a WS with a vector $\mathbf{s}^{QoS}$ of QoS criteria. It is clear that the WS is eligible for the given task iff $\mathbf{g}_{k_i} + \mathbf{s}^{QoS} + \mathbf{h}_{k_j} < \mathbf{s}_{max}$ (the inequality is taken elementwise). WS is rejected if the inequality is not verified. This rule ensures the constraints are always satisfied along any path, i.e., for any assignment of WS to tasks; it allows to dynamically manage the inclusion of new WS in the service provision.

12.3.3 Computing the Optimal TA Policy

The MWS begins with task allocation from the initial state and chooses from state k the allocation of a WS u to a task t_i with a probability distribution $\pi_k(u)$, which aims to exploration. The associated cost $c(t_i, w_u)$ is incurred and is denoted for simplicity $c(k, i)$ (cost may vary over time in a dynamic environment); the MWS then moves to the new state, k'. This allows the SM to update the estimates of the cost, $\widehat{c}(k, i)$. The RRL for an acyclic graph, where the states are ordered in such a way that there is no arc going backward (i.e. there exists no arc linking a state k' to a state k where $k' > k$), is as follows (for details, see [3]):

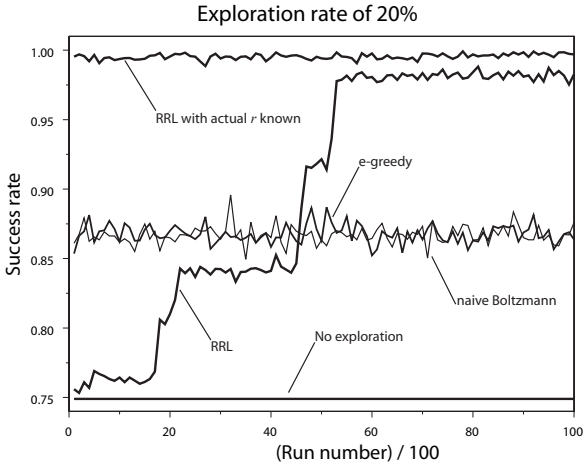


Fig. 19.

1. *Initialization phase:* Set $V(k_d) = 0$, which is the expected cost at the destination state.
2. *Computation of the TA policy and the expected cost under exploration constraints:* For $k_i = (k_d - 1)$ to the initial state k_0, compute:

$$\begin{cases} \pi_{k_i}(u) = \dfrac{\exp\left[-\theta_{k_i}\left(c(k_i,u)+V(k'_{i,u})\right)\right]}{\sum\limits_{u' \in U(k_i)} \exp\left[-\theta_{k_i}\left(c(k_i,u')+V(k'_{i,u'})\right)\right]}, \\ V(k_i) = \sum\limits_{u \in U(k_i)} \pi_{k_i}(u)\left[c(k_i,u) + V(k'_{i,u})\right] \text{ for } k_i \neq k_d \end{cases} \tag{9}$$

where $k'_{i,u} = f_k(u)$ and θ_{k_i} is set in order to respect the prescribed degree of entropy at each state (see Eq.5 which can be solved by a simple bisection search).

One can show that this probability distribution law for task allocation minimizes the expected cost (see Eq.6) from the starting to the destination node for a fixed exploration rate [3, 4].

Various approaches can be used to update the estimated WS criterion $\hat{r}_u$; e.g., exponential smoothing leads to:

$$\widehat{r}_u \leftarrow \alpha \bar{r}_u + (1-\alpha)\widehat{r}_u \tag{10}$$

where $\bar{r}_u$ is the observed value of the criterion for w_u and $\alpha \in]0,1[$ is the smoothing parameter. Alternatively, various stochastic approximation updating rules could also be used. The MWS updates its estimates of the criterion each time a WS performs a task and the associated cost is updated accordingly.

12.4 Experimental Results

Experimental setup. Task allocation for the CWS diplayed in Fig.18 was performed. A total of three distinct WS were made available for each distinct task. Each $w_{k,u}$ is characterized by its actual r_u which is an indicator of the WS's quality over the optimization criterion (see, §12.3.1). In this simulation, it will simply be the probability of successfully performing the task (1 – probability of failure). In total, 57 WS are available to the MWS for task allocation. For all WS u, r_u takes its value $\in [0,1]$; for 70% of the WS, the actual r_u is *hidden* (assuming it is unknown to the MWS) and its initial expected value, $\widehat{r}_u$, is set, by default, to 0.3 (high probability of failure since the behavior of the WS has never been observed up to now), while actual r_u value is available to the MWS for the remaining 30% (assuming these WS are well known to the MWS). Actual r_u is randomly assigned from the interval $[0.5, 1.0]$ following a uniform probability distribution. It has been further assumed that $\widehat{c}(t_i, w_u) = -ln(\widehat{r}_u)$, meaning that it is the product of the r_u along a path that is optimized (this is a standard measure of the reliability of a system). After all tasks are allocated, the selected WS execute their allocated tasks according to their actual r_u value (with failure $1 - r_u$). The estimated WS criterion $\hat{r}_u$ is then updated by exponential smoothing, according to Eq.10. In Eq.10, $\bar{r}_u$ equals 1 if w_u is successful at executing the task it has been allocated, 0 otherwise. Estimated costs are of course updated in terms of the $\widehat{r}_u$ and each time a complete allocation occurs, the probability distributions of choosing a WS are updated according to Eq.9. 10,000 complete allocations were simulated for exploration rates 20%, and 30%.

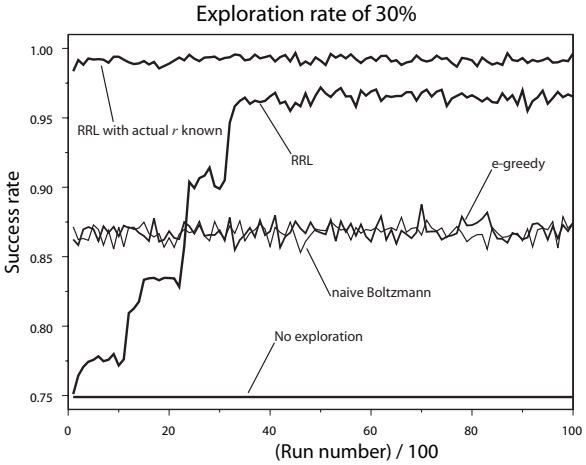


Fig. 20.

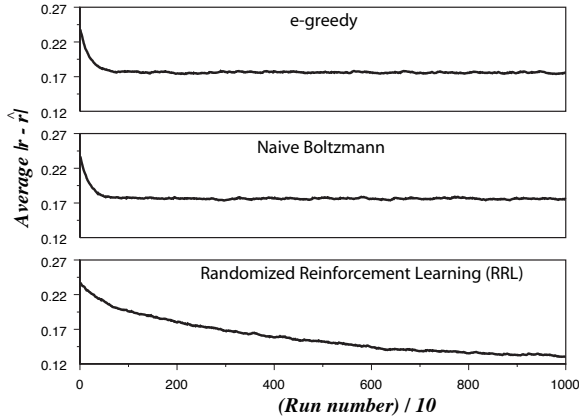


Fig. 21.

Results. The RRL is compared to two other standard exploration methods, ϵ-*greedy* and *naive Boltzmann* (see [3] for details), while tuning their parameters to ensure the same exploration level as for RRL. The *success rate* is defined as the proportion of services that are successfully completed (i.e., all tasks composing the service are allocated and executed successfully) and is displayed in Figures 19 and 20 in terms of the run number (one run corresponding to one complete assignment of tasks, criterion estimation and probability distribution update). Figures 19 and 20 show the RRL behaves as expected. It converges almost to the success rate of the RRL in which all actual r are known from the outset (i.e., need not be estimated)—and indicate that exploration clearly helps by outperforming the allocation system without exploration (which has a constant 75% success rate). Fig.21 compares the three exploration methods by plotting the average absolute difference between actual r_u and estimated $\widehat{r}_u$ criterion values for a 30% exploration rate. Exploration is therefore clearly helpful when the environment changes with the appearance of new agents—i.e., exploration is useful for directing MWS behavior in dynamic, changing, and open architectures, i.e., in the SCA.

12.5 Related Work

Regarding task allocation, closest to the present work is the generalization of the Semi-Markov Decision Process (SMDP) [304] model which provides a representation of the mediator decision problem. Abdallah and Lesser [1] formulate the mediator decision problem by extending the original SMDP formulation to account for randomly available actions and allow concurrent task execution. With regards to prior effort (e.g., [117]), they advance the matter by avoiding only serial task execution, homogenous agents, and deterministic action availability, while the reported experiments indicate their approach outperforms the original SMDP and the Concurrent Action Model [275]. In another paper, Abdallah and Lesser [2] suggest an algorithm for coordinating work between mediators: in a distributed architecture, mediators observe

only part of what other mediators can observe, so that optimal task allocation accross pooled agents can be represented as a game with incomplete information. While coordination across mediators is outside the scope of the present paper, it can be noted that the learning mechanism employed by the cited authors does not involve exploration, only exploitation. The RRL allows the execution of potentially complex processes (but without concurrency, see §12.6) while assuming that the set of available WS is changing. One distinctive characteristic the MWS's behavior suggested in the present paper is that the algorithm accounts for a vector of criteria when allocating tasks, including QoS, service provision deadline, provision cost, explicit user preferences, and agent reputation. Maximilien and Singh [217] propose service selection driven by trust values assigned to service providing agents. Trust is extracted from user-generated reports of past service performance over qualities defined by a system-specific QoS ontology. Level of trust depends on the degree to which reputation and quality levels advertised by the provider match. By basing selection on trust only and generating levels of trust from advertised and user-observed behavior, Maximilien and Singh's approach involves learning driven by exploitation of historical information, without exploration.

Tesauro and colleagues [308] present a decentralized architecture for autonomic computing where tasks are allocated according to the ability of agents to execute them. Allocation is utility-driven, whereby each resource has an associated and continually updated function which provides the value to the application environment of obtaining each possible level of the given resource. Information on the mediation process is very limited, seemingly indicating that no empirical data on actual agent quality is employed—i.e., it seems assumed that advertised behavior is the actual behavior, thus undermining the appropriateness of the given arhitecture for an open system. Shaheen Fatima and Wooldridge [84] suggest an architecture in which permanent agents associated to the system are provided alongside agents that can enter and leave. Their focus is on minimizing the number of tasks that cannot be executed by the system because of overload. No QoS considerations are accounted for in task allocation and it appears that no learning occurs, the former undermining realistic application, while the latter harms adaptability. Tasks are queued based on priority values. Klein and Tichy [177] focus on ensuring reliability and availability through automatic reconfiguration. Agents are self-interested and selection proceeds by reward for accomplishing a task. There are no QoS considerations and no explicit learning based on observed behavior.

12.6 Conclusions and Future Work

In response to the need for architectures for open, distributed, service-oriented systems capable of dynamic adaptation in response to the changing operating conditions, we propose the combination of the SCA and the RRL. The MWS in SCA use a reinforcement learning algorithm—i.e., the RRL—combining exploitation with exploration to ensure both the use of acquired knowledge about the actual quality of web services and the anticipated quality of newly available WS. Composition is dynamic, and is driven by multiple criteria, including QoS, deadline, reputation, cost, and additional explicit user preferences. The reported experiments show the algorithm outperforming standard exploration strategies, ϵ-*greedy* and *naive Boltzmann*. Future work focuses on experimentation of the architecture and the extension of service graph specifications, to allow, e.g., task concurrency, so as to move closer to using the architecture and the algorithm in actual application settings where few limitations on service graphs are acceptable.

13

Requirements-Driven Contracting for Open and Norm-Regulated Multi-Agent Systems

Abstract. Heterogenous agents designed by, and distributed across various providers participate in an open multi-agent system (MAS). Norms (i.e., obligations, prohibitions, and permissions), sanctions, and incentives are enforced within an open MAS to ensure that agents act and interact only in ways that satisfy stakeholders' requirements. We propose a framework, called Requirements-driven Contracting (RdC) for deriving executable norms, sanctions, and incentives from the requirements and associated relevant information. RdC allows the use of requirements engineering concepts and methods to govern the open MAS. RdC ensures that all requirements, along with runtime changes of requirements are reflected in the norms, sanctions, and incentives regulating the behavior of agents in an open MAS.

13.1 Introduction

Engineering and managing the operation of increasingly complex systems is recognized to be a key challenge in computing [144, 307, 174]. It is now widely acknowledged that degrees of automation needed in response cannot be achieved without open, distributed, interoperable, and modular information systems capable of dynamic adaptation to changing operating conditions. Among the various, often overlapping approaches to building such systems, the multi-agent systems paradigm stands out in terms of available research results, possible deployment on the World Wide Web, availability of standards for describing and enabling interaction between agents, attention to interoperability, and interest in industry.

A multi-agent system (MAS) is *open* if it relies on heterogenous agents designed, operated, and maintained by, and distributed across, various providers. The number and kind of agents that may participate in an open MAS is unknown at development time, and varies at runtime. Open MAS enable virtual organizations, i.e., structures that define and regulate interactions among potentially many geographically distributed software and/or human agents that act and interact to achieve individual and global goals [229]. Open virtual organizations are expected to improve the responsiveness of human organizations when satisfying customers' expectations, while reducing the cost at which the expectations are satisfied, namely through individual providers' economies of scale, reduced cost of switching between providers, and the competition among providers (e.g., [235]).

Similarly to human organizations, virtual organizations require that all members behave in a manner consistent with instituted rules. Only then can they fulfil the purpose of the organization by taking individual and joint action. Unregulated behavior is unpredictable, and therefore unacceptable.

Norms (i.e., obligations, prohibitions, and permissions), *sanctions*, and *incentives* are promising abstractions for the specification and operation of open MAS. They govern the observable behavior of self-interested, heterogenous agents designed by various providers who do not necessarily and/or entirely trust each other [65, 74, 322].

Because norms, sanctions, and incentives (NSI) ensure that the open MAS fulfills its purpose, they are derived from requirements placed on the MAS by users and other system stakeholders. For instance, if there is a requirement for a banking MAS stating that a letter of credit cannot be issued if there is no deposit, there must be a norm that ensures that this requirement is indeed not violated at runtime. While it is intuitively clear that there is a relationship between requirements and NSI, the relationships between common requirements engineering concepts and norm-related notions for MAS governance have not been studied. It is, for example, unclear whether/how various kinds of goals used to specify requirements relate to the different kinds of norms. In response, we discuss these relationships and propose a framework, called Requirements-driven Contracting (RdC) to write specifications of requirements and associated notions and subsequently derive executable NSI. RdC is relevant in several respects:

- Concepts and methods known in requirements engineering can be used to specify NSI. Given requirements, domain assumptions, preferences, and priorities, we formalize these using RdC and obtain an environment specification. Once the environment specification is available, we use RdC to derive the specification of executable NSI. We thus bridge two levels of abstraction, the higher requirements level and the lower governance level, arguably simplifying the engineering of open and norm-regulated MAS.
- RdC allows MAS runtime behavior to be governed at runtime through the environment specification. Therefore:
  - Any change in requirements at runtime is reflected in changes of norms and related notions that regulate the open MAS. Requirements elicited and specified at development time need not remain rigid after deployment: we can modify the environment specification at runtime to better adjust, e.g., to changes in stakeholders' preferences and/or to ensure that the system better fits its operating environment.
  - While some norms can be static, others are dynamic, that is, need to change as agents' actions and interactions are observed: obligations can be added, fulfilled, or become obsolete; prohibitions can be lifted or new ones imposed; new permissions and/or sanctions may become relevant, while others less so. While some changes in NSI are requirements-driven (e.g., introduction of new obligations to satisfy new requirements), others arise from the necessity to ensure that current requirements remain satisfied after some variation in the pool of agents participating in the open MAS. Any adjustment, and/or the adding or removing of NSI that is necessary at runtime can be performed by modifying the environment specification.

Relating NSI to stakeholders' requirements involves the manipulation of concepts at three levels of abstraction. The first level, that of *requirements* involves statements of requirements and of other relevant information represented in natural language. This information forms the basis for writing a precise specification at the second, *environment specification* level. The third, *contracts* level involves executable NSI, derived from the environment specification. Below, we first propose and discuss a rich collection of concepts relevant at the requirements level, when representing and reasoning about stakeholders' statements of requirements and associated notions in the context of open MAS (§13.2). We then identify notions that constitute the environment specification and their correspondence to those of the requirements level (§13.3). While the environment specification focuses on relating requirements to NSI, the specification is designed to be extensible, so that various analyses, available in requirements engineering can be applied in conjunction with RdC. The format for NSI in open MAS is overviewed, and it is shown how NSI are derived from the environment specification (§13.4). We then outline the open MAS architecture in which norms govern agent behavior. Related work is then reviewed (§13.6). We close the paper with a discussion of the limitations and directions for future effort (§13.7), along with the conclusions (§13.8).

13.2 Requirements

Requirements engineering (RE) is a structured approach to the assessment of the role that a future system is to have within a relatively well-delimited human and/or automated environment. It involves the identification of goals to be achieved by the system, their operationalization into implementable functionalities and constraints, the identification of resources required to perform those functionalities and the assignment of responsibilities for the resulting requirements to agents, such as humans, devices, and software. At the outset of the RE process, the requirements engineer interacts with the system stakeholders (including, among others, users, buyers, third-party component providers, legacy systems) in order to collect information that subsequently become requirements and/or assumptions about the behavior of the future system and its operating environment. It is necessary to know the kinds of information relevant for the requirements engineer in order to write corresponding NSI. This section identifies the kinds of information collected during the RE of open MAS and that need to be subsequently transformed into NSI.

13.2.1 From Statements to Requirements and Domain Assumptions

In their seminal paper [339], Zave and Jackson outline a terminological framework for RE, which revolves around three foundational concepts. Therein, a *requirement* is an optative (i.e., desired) property of the environment, whereby the environment includes the future system and its relevant surroundings. A *domain assumption* is an indicative property of the environment, describing the environment as it is, i.e.,

in absence of the system or regardless of the system. A *specification* is an optative property intended to be directly implementable and to support the satisfaction of the requirements.

When Zave and Jackson use notions of 'indicative' and 'optative', they are concerned with the grammatical mood of a natural language statement (written, spoken, or communicated otherwise). Broadly speaking, in any given statement the *dictum*, or what is said is distinguished from the *modus*, or how it is said in terms of speaker's attitude about what is said.[1] Modus influences the meaning intended for the given dictum. As Zave and Jackson have observed, this is significant for requirements engineering since knowing the modus allows us to classify some statements as requirements, others as domain assumptions.

While mood has a grammatical flavor, the distinction dictum/modus also corresponds to Searle's separation of, respectively the proposition from the illocutionary (or speech) act [283]. Turning to speech acts instead of grammatical mood gives us a basis for the classification of stakeholders' statements independent of the particular natural language.[2] Searle distinguishes the following kinds of illocutionary acts [284]:

1. *Assertives*, which assert the proposition that the speaker believes is true;
2. *Directives*, which convey the proposition that the speaker wants to see become true;
3. *Commissives* stating what the speaker intends to (do to) make a proposition true;
4. *Expressives* conveying speaker's emotion/attitude;
5. *Declarations* which by the very act of being stated make a proposition true;
6. *Representative declaratives* which recognize the truth of a proposition that has been made true through a declaration.

With the above taxonomy, we can go further than Zave and Jackson in classifying stakeholders' statements. This is relevant when working with open MAS for several reasons:

- Additional detail allows us to have a richer account of concepts relevant for eliciting and specifying requirements, then transforming them into NSI. Namely, Searle's taxonomy will provide a foundation on which to define the notions of preference and priority. Both are particularly important for open systems. Once open, the pool of agents that interact to satisfy requirements will change, so that there will be variation in how and how well requirements are satisfied. We therefore must allow the stakeholders to indicate more and less desirable system behavior: preferences allow stakeholders to establish an order among alternative requirements, which is subsequently accounted for by the system. Hence, the system will at some times satisfy more preferred requirements, at others less preferred ones. There are two problems without such preference orders:
  - Idealistic requirements: the stakeholders define one set of requirements such that the system can only rarely satisfy them, given the changing availability of agents. If they defined a preference order among alternatives, less stringent requirements may be satisfied by the system until new agents become available.
  - Pessimistic requirements: the stakeholders define one set of requirements that the system can satisfy without fully using its resources. In other words, it is possible to satisfy these requirements to a more desirable extent, but the system cannot do so since the stakeholders specified a limit. With a preference order, the stakeholders could have specified also the more desirable alternatives, to which the system can respond when appropriate agents are available.

  Priorities on the other hand are needed once whenever preferences are specified. Given a pair of preferences that cannot be satisfied simultaneously, the stakeholders use priorities to indicate which of the two preferences is more important, which resolves the conflict.
- The agent-based paradigm in computing conceives the agent as an autonomous, internally-motivated entity that is situated within a dynamic and not entirely predictable environment from which it receives inputs and which it changes by performing actions [98]. In widespread deliberative agent architectures [329], agent's decision-making is viewed in terms of beliefs (what the agent knows about its world), desires (what the agent wants), and intentions (what the agent is actually committed to doing). Examples of such Belief-Desire-Intention (BDI) architectures include for instance IRMA [40] and PRS

[1] Arguments in favor of such distinction are discussed in and by, among others: Frege's *Begriffsschrift* (translated in, e.g., [320]), where the content of judgement is distinguished from the act of judgement of the content; Wittengstein's *Tractatus logico-philosophicus* [206], where the content of the proposition corresponds to a possible state, and it distinctively asserted that the proposition describes the real state; Stenius [300] separates a sentence radical which gives the descriptive content of a sentence, from a modal element which signifies mood (of indicative, imperative, or interrogative kind).

[2] This is not to say that Searle's classification of speech acts is not debated—see, e.g., [336] for a recent discussion. It is, however, widely accepted and fits the practical purpose herein.

[103]. Searle's taxonomy is particularly relevant as each illocutionary act is related to a mental state, i.e., assertives and representative declaratives communicate beliefs, directives desires, commissives intentions, expressives attitudes, and declarations become beliefs by the act of communication (i.e., the environment changes by making the declaration). By adopting this taxonomy, we achieve two goals:

- Relating stakeholders' statements to mental states by way of illocutionary acts allows us later on to establish the relationship between these statements and corresponding RE concepts to NSI, as NSI also relate to BDI mental states [245, 178].
- We ensure that the intentional notions span the requirements, the governance, and the architectural levels. RdC thus consistently relies and benefits from a common paradigm at the various levels of abstraction.

Grounding the RE concepts in Searle's taxonomy requires us to propose and discuss herein revised definitions of established RE concepts. Revisions clarify the intended meaning for the relevant concepts, and later on facilitate the specification of requirements and the mapping to NSI. The departure from established notions in RE is uncontroversial, as we have discussed in more detail elsewhere [156, 152]. BDI orientation is implicit in established RE frameworks, in which the "goal" concept plays a key role, and is used interchangeably to model desires or intentions. It is now accepted that the concept is relevant for the elicitation, elaboration, structuring, specification, analysis, negotiation, documentation, and modification of stakeholders' requirements on a system [225, 71, 331, 11, 332, 276, 55, 197, 315, 50, 100, 198, 316]. By reusing the notion of goal, and clarifying the intended meaning appropriate to the present setting, we allow the engineer who uses RdC to combine it with established contributions in RE, in particular the methodologies for the elicitation of requirements [71, 50].

We first discuss below the distinction between the requirements and domain assumptions in light of Searle's illocutionary act taxonomy, then identify kinds of requirements and their corresponding notions in the environment specification.

13.2.2 Domain Assumption and Requirement

Consider the business protocol for the exchange of goods involving a Letter of Credit. There are three agents: supplier (of goods), customer (who buys the goods), and the authority (which supervises the exchange). The customer needs credit to finance the purchase of goods. To obtain credit, the customer must first make a deposit with the authority. When eliciting information about how the Letter of Credit is obtained, we can encounter the following statement:

(Ex.16) *The Banker cannot issue the Letter of Credit cannot be issued if no corresponding deposit has been made with the Banker.*

If the stakeholder who communicates this believes it is already true, the statement is assertive. If the stakeholder intends to institute the above as a rule, the statement is a declaration. If the stakeholder merely reiterates that the above applies, it is representative declarative. Consider the following statements:

(Ex.17) *I hope/wish/desire/expect the letter of credit to be issued immediately after the deposit is recorded.*

(Ex.18) *I will ensure that the letter of credit is issued immediately after the deposit is recorded.*

(Ex.19) *It is preferred that the letter of credit be issued after the deposit has been recorded and the credit capacity of the customer verified.*

The first statement above is directive as it expresses what is desired. The second is commissive for it indicates the stakeholders intention on bringing about something desired. Finally, the third statement is expressive, as it makes explicit stakeholders attitude regarding how the booking confirmation is to be sent to the user. If we follow Zave and Jacksons terminology, we see that the critical distinction between a requirement and a domain assumption lies in that the former expresses something desired, while the latter something that is already the case, or is/will be the case regardless of the system. It is not difficult to see that the three statements above are requirements. Observe that it is difficult to establish modus from written text. What the examples lack is the background over which the statements are made [285]. One way of understanding the background of a statement is by knowing the purpose of the discourse within which the statement appears. Vanderveken [319] suggested that descriptive (in which beliefs about

what is true are exchanged), deliberative (in which the speaker and hearer deliberate on what to commit), declaratory (in which declarations are made), and expressive (in which attitudes of the speaker and the hearer are conveyed) discursive goals be distinguished. If, e.g., we know that some statement is made within a declaratory discourse, we would consider it a declaration and not otherwise. While eliciting the statements, the engineer can proceed by question and answer to determine the kind of discourse, and subsequently the modus of the statement. With the introduction of Searles illocutionary acts, we arrive at finer definitions for the notions of domain assumption and requirement.

Definition 13.1. *Requirement is either a directive, commissive, or expressive illocutionary act about the environment.*

Definition 13.2. *Domain assumption is either an assertive or declarative illocutionary act about the environment.*

A requirement thus communicates what is desired, in accordance with the intuition commonly accepted in RE (e.g., [279, 132, 338, 339, 184, 315]). However, it also communicates attitudes (through expressives), which can be informally seen as pointing out the strength with which something is desired, or comparing it to what is less desired. Requirements therefore encompass preferences. We also take intentions to be requirements since they are not realized when stated; that is, as intentions involve commitment to bring about what is desired, they involve desires, and are therefore considered as requirements. Regarding Zave and Jackson's notion of domain assumption, we consider not only assertives, but also declarations. Declarations are are by definition not requirements, since their very communication brings about what has been desired before communication. Representative declaratives are neither requirements nor domain assumptions, but merely restatements of domain assumptions.

We see from the above definitions that the requirement and domain assumption concepts are too complex to have unique corresponding notions at the environment specification level: below, we discuss taxonomies of requirements and domain assumptions, and identify their corresponding notions at the specification level.

13.2.3 Requirements

A requirement is:

- either *functional* or *nonfunctional*, and
- either *hard* or *soft*, and
- either *free* or *conditional*.

Functional vs. Nonfunctional Requirement

Definition 13.3. *Functional requirement is either a directive or a commissive illocutionary act for which the requirements engineer can at any time verify whether it is satisfied or not in the environment.*

Verifiability is a salient characteristic of functional requirements [315]. Such requirements concern the functionalities or, broadly speaking, services that the system is expected to deliver. In this respect, their satisfaction is clear cut, for the system either does or does not perform the expected functionalities or deliver the services. Above, the functional requirement expresses a desire or an intention. A functional requirement cannot be an expressive illocutionary point because expressives state attitudes and thus fit preferences, as we discuss further below. Each of the following statements is therefore a functional requirement:

(Ex.20) *Issue a Letter of Credit.*

(Ex.21) *Record a deposit in electronic and paper form.*

(Ex.22) *Transfer Letter of Credit amount to supplier.*

Early suggestions on how to distinguish nonfunctional from functional requirements persist. For instance, van Lamsweerde [315] borrows Keller, Kahn, and Panara's [172] distinction which remains implicit or explicit and in unchanged form and interpretation in subsequent literature (e.g., [225, 55, 100, 198]):

> "Functional [requirements] underlie services that the system is expected to deliver whereas nonfunctional [requirements] refer to expected system qualities such as security, safety, performance, usability, flexibility, customizability, interoperability, and so forth."

It appears intuitive that nonfunctional requirements constrain how well functional requirements need to be satisfied. Moreover, there is no nonfunctional requirement without a functional requirement: we do not require "convenience" independently of stating what should be convenient. This perspective is applied in established RE frameworks (e.g., [225, 55, 315]. Consider the following statement:

(Ex.23) *Ensure that the process of obtaining the Letter of Credit is convenient.*

(Ex.24) *Quickly issue the Letter of Credit.*

Convenience and rapidity in the above statements are nonofunctional considerations, used to compare alternative ways of satisfying the associated functional requirements. Nonfunctional requirements give rise to criteria for rating functional requirements. The level of abstraction of these criteria is a key concern for the requirements engineer. It is now accepted that convenience, just as usability, maintainability, security, and so on, have no domain- and project-independent definition [176, 78]. Knowing therefore that the stakeholder expects convenience is not of much use if we do not know how to measure it, and thus evaluate (e.g., through simulation) whether the system-to-be will be convenient enough or not. We are interested in perceivable and measurable characteristics that we can then meaningfully relate to (or aggregate in) abstract notions such as convenience. We return to the measurable characteristics below, when we discuss the nonfunctional specification.

Statements about nonfunctional considerations have two salient characteristics. First, they express attitudes. To state that something is to performed quickly is to implicitly indicate that performing slowly is less interesting. Second, expressions of nonfunctional requirements always involve *gradable adjectives*, such as quick, convenient, secure, or useful, efficient, accurate, and so on. When performing semantic analysis of gradable adjectives, linguists usually make two assumptions (we borrow here from Kennedy [173], emphasis from the original): (i) gradable adjectives map their arguments onto abstract representations of measurement, or *degrees*; and (ii) a set of degrees totally ordered with respect to some *dimension* (height, cost, etc.) constitutes a *scale*. A gradable adjective describes that a characteristic obtains to some degree, whereby the degree is at least as great as some standard of comparison, the standard itself being external to the adjective and determined by the context in which the adjective is used. In Ex.9, whether the Letter of Credit is issued quickly involves a standard of comparison which is often local to each individual stakeholder, so that the same actual time to issue may be called quick by some stakeholder and not quick enough by another.

Definition 13.4. *Nonfunctional requirement is an expressive illocutionary act which communicates an attitude using a gradable adjective without established degrees, and/or scale, and/or dimension.*

Above, by "established" we mean standard, agreed on, or widely shared and used. As we shall se below, establishing one or more relevant standards is needed to convert a nonfunctional requirement into a nonfunctional specification.

Hard vs. Soft Requirements

This distinction is considerably simpler than the functional vs. nonfunctional one. Herein, the aim is to distinguish among requirements whose satisfaction is considered compulsory by the stakeholders, as opposed to optional. Hard requirements and the corresponding hard specification are therefore those that *must* be satisfied if the stakeholders are to consider system behavior as acceptable. In other words, any system design that cannot satisfy all hard requirements is unacceptable. Soft requirements and corresponding soft specification are optional: the more of these requirements the system satisfies, the higher the stakeholders' satisfaction. Behavior that satisfies all hard requirements remains acceptable if it satisfies no soft requirements. Soft requirements are not uncommon in practice: non-critical requirements that cannot be satisfied given the development project resources are soft. A usual example are requirements not satisfied in a given system version and are then are left to be accommodated in a future version.

Free vs. Conditional Requirements

A requirement is conditional if it needs to be satisfied only when some particular conditions hold. The following statement is a constrained requirement:

(Ex.25) *When the credit amount is transferred to the supplier, the authority informs the customer of the transfer.*

The condition can be that another requirement is satisfied (to some specific extent in the case of nonfunctional requirements), or that some domain assumption holds. Free requirements are those for which conditions need not be expressed (e.g., their satisfaction does not depend on whether some other requirement is satisfied), as in the statement below:

(Ex.26) *All transactions involved with a Letter of Credit need to be recorded by the authority.*

In practice, free requirements appear mostly in the earliest stages of the RE process. A requirement such as the one above should be decomposed, so as to identify the various transactions that can take place in relation to the Letter of Credit. By considering each transaction separately, we can then identify the conditions in which the transactions occur, and which then result in the recording by the authority.

13.2.4 Domain Assumptions

We have suggested above that a domain assumption is either an assertive or a declarative illocutionary act about the environment. We classify domain assumptions into either *free* or *conditional* ones. This classification is analogous to that of free or conditional requirements. Free assumptions are those that apply regardless of specific conditions holding, while conditional ones make explicit the conditions that must hold in order for the assumption to apply. While it is evident that we can almost always identify conditions to relate to a domain assumption, there are cases in which adding the conditions does not make the domain assumption more precise or relevant. For instance, a physical law is to be modeled as a free domain assumption: except if the future system is intended to operate outside of the planet, there is no need to, e.g., indicate that gravity has a specific value conditionally to being on Earth. The said taxonomic dimension is therefore not derived from inherent characteristics of the domain assumptions, so that the proposed classification is grounded in the engineer's decision on whether to make explicit the conditions or not.

13.2.5 Preference and Priority

We have observed that nonfunctional requirements express stakeholders' preferences. They do so in an indirect manner: asking for convenience implicitly assumes that more or higher convenience is preferred to less/lower. The same applies, e.g., for security, efficiency, maintainability, usability, and so on. Because we use nonfunctional requirements as criteria for comparison of alternative (sets of) system functionality, we use them to establish an implicit order among functional requirements. By introducing the notion of preference outside the notion of requirement, it becomes possible to make such an ordering explicit and to order both nonfunctional and functional requirements.

A requirement is preferred to another requirement if it more desirable to have the system satisfy the former than the latter.

Definition 13.5. *A preference relation establishes an order of preference between two or more requirements. A statement of preference is an expressive illocutionary point that communicates a preference relation between two or more requirements.*

The following is a statement of preference:

(Ex.27) *Keeping electronic records of transactions for a Letter of Credit is preferred to keeping in addition paper records of these same transactions.*

Not all preferences can be simultaneously satisfied.[3] For instance, quicker issuance of a Letter of Credit may correlate negatively with the depth of credit checks that the authority performs on the customer. When aiming to satisfy one preferred requirement negatively affects the ability to satisfy some other requirement, we say that the involved requirement preferences are conflicting. In such a case, the stakeholders and the engineer should determine the relative importance of the conflicting preferences. That is, they should state which of the preferences it is more important to satisfy.

[3] A preference is more or less satisfied depending on which of the requirements ordered in the preference is satisfied by the system.

Definition 13.6. *A priority relation establishes an order of importance between two or more preference. A statement of priority is an expressive illocutionary point that communicates a priority relation between two or more preferences.*

If a stakeholder prefers that Letter of Credit transactions be recorded and kept for as long as possible in paper format, and thus expresses preference for paper records kept for, e.g., five years instead of three years, satisfying this stakeholder's preference conflicts with satisfying the preference given in Ex.12. We therefore state that satisfying one is more important thann satisfying another, thus introducing a priority order between them and resolving the conflict.

13.3 Environment Specification

We have introduced above a number of concepts to cover the various kinds of statements that are encountered during RE, and for which corresponding NSI need to be identified. Given requirements, domain assumptions, preferences, and priorities, the engineer's task amounts to define an environment specification (ES). If the MAS behaves according to the ES, it satisfies the various sets of requirements ordered by preference, while ensuring that the domain assumptions and the priorities are not violated. ES involves four parts:

- *Functional ES* specifies functional requirements and domain assumptions. It describes the functional goals that need to be achieved, the constraints on plans used to achieve the goals, and the domain constraints that must not be violated during system operation. Preference orders are also given over alternative functional goals and/or alternative plan constraints.
- *Nonfunctional ES* specifies nonfunctional requirements. This specification defines measures of system behaviors, and indicates the desired values for the measures. Preference orders are indicated over alternative values of the measures.
- *Priorities ES* specifies priorities over conflicting preferences. Based on experience and observed runtime behavior of the system, conflicting preferences are identified and orders of importance defined to avoid conflict at runtime.
- *Terminological ES* specifies the domain ontology relevant for the MAS. Terms appearing in the functional, nonfunctional, and priority ES refer to concepts in the domain in which the MAS operates. While the Terminological ES is optional, defining the ontology and using it consistently accross the ES and governance levels makes it possible to (i) delimit more precisely the meaning intended for words used at requirements, ES, and governance levels; and (ii) improve the overall quality of matching agents' caabilities to the plans that agents need to execute.

Note that the term "environment specification" differs from the (requirements) "specification", as it is commonly used in RE [339]: a (requirements) specification excludes the domain assumptions, preferences, and priorities. The ES covers all of these notions mainly for two reasons: (a) because various information collected at the requirements level can give rise to norms and/or sanctions, the ES acts as a unique source of information from which to derive NSI; (b) we should have a unique specification on which to perform inference.

Graphical and Formal Representation of ES.

Two approaches are usually applied when specifying requirements and associated relevant information during RE. As statements communicating this information are usually given in natural language (be they spoken, written, or otherwise), it is necessary to proceed first to a somewhat imprecise structuring and recording of the given information, then write it in a precise form using a mathematical notation. The initial representation is useful as it can be submitted to analyses intended to ensure, among others, that stakeholders understand and agree on the intended meaning of the given information (e.g., [32, 153]), and that stakeholders understand the overall changes that the system is likely to bring about in the environment [332, 50]. The formal representation is given in a mathematical notation and supports inferences. Various grahical and formal notations have been suggested, including the i^* goal modeling notation [331, 332] used in the Tropos RE methodology [50, 100], the KAOS goal trees [71], the Vienna Development Method (VDM) [189], the Z notation [299], the GAIA agent-oriented software engineering methodology [337], the ISLANDER framework for agent institutions [82], and the OMNI framework for modeling agent organizations [322]. If we reuse herein one of the specification languages outlined in the cited works, we encounter several difficulties:

- The KAOS and Tropos frameworks use a variant of first-order linear temporal logic to write the specification. In addition to being attractive in terms of expressivity, model checking of the requirements specification remains tractable in practice for reasonably small systems (e.g., [99]). Model checking remains feasible for small systems specified in Z [295] or VDM notations. Because it is difficult to maintain the small size assumption for open MAS, we limit the specification to Horn clauses. Future effort will be focused on the use of more expressive formalisms.
- Reading and writing specifications in temporal FOL, or in Z and VDM requires sophisticated training. These characteristics warrant sparse and focused use in practice [36]. Reading and writing ES should instead be accessible.
- GAIA, ISLANDER, and OMNI feature less expressive specification notations than RE frameworks, Z, and VDM. Neither of these approaches starts from the level of requirements, domain assumptions, preferences, and priorities, hence lacking the correponding concepts at the specification level. While OMNI is focused on engineering governance mechanisms in open MAS, it does not focus significantly on the conceptualization of requirements; it does not integrate the notions of preference and priority.

As in all of the said approaches, the specification in RdC is written in the form of templates. Each template carries attributes relevant for the kind of information (i.e., requirement, domain assumption, or other). Logical constraints that appear in some attributes are written as Horn clauses. A general form of Horn clause is $a_0 \vee (\neg a_1) \vee \ldots \vee (\neg a_n)$, where each $a_i, i = 1, \ldots, n$ is an atom. This is equivalent to $a_0 \vee \neg(a_1 \wedge \ldots \wedge a_n)$, which in turn is equivalent to $(a_1 \wedge \ldots \wedge a_n) \Rightarrow a_0$. We adopt here the the standard notation for a Horn clause, i.e., $a_0 \leftarrow a_1, \ldots, a_n$; the clause is read a_0 is true if $a_1, \ldots, a_n$ are true. Since the problem of testing a set of Horn clauses for satisfiability is known to have linear time solution algorithms (e.g., [140]), the ES supports rather efficient inferences compared, e.g., to RE specification formalisms. We use the common i^* goal modeling notation for the visual representation of Functional ES, without providing a graphical representation for the remaining parts of the ES.

Governance and Non-governance Attributes.

In this paper, we are primarily interested in using the ES as a source of information for the definition of executable NSI that govern open MAS. A specification such as the ES serves other purposes in practice, namely to guide the acquisition of relevant information and to assist the documenting effort. Attributes guide the engineer when identifying the information to acquire. However, not all of the attributes that may appear in the ES templates are directly relevant for the definition of NSI. We therefore distinguish governance attributes (of immediate interest when defining norms) and non-governance attributes. Non-governance attributes appear for instance in the definition of measures used in the Nonfunctional ES, where we would normally document, among others how the measure is collected and how it is transformed (e.g., moving average). We do not discuss non-governance attributes in this paper, as they depend on the methods used jointly with RdC during the RE of open MAS.

13.3.1 Functional ES

Figure 22 shows one way of representing the functional requirements specification for the exchange of goods involving a Letter of Credit. We use the established i^* framework to draw the model [332, 50], whereby we employ only primitives relevant to functional considerations. The goal model shows the customer, the supplier, and the banker roles. Dependencies between the roles are shown: e.g., the banker can verify the credit capacity of the customer only if the customer provides the credit request, so that, to satisfy the goal receive credit request, the banker depends on the customer to act accordingly. Dependencies are particularly relevant with regards to norm-regulated MAS, as they highlight situations in which there is vulnerability of one of the parties involved in the dependency. NSI in such situations prove invaluable to ensure that the vulnerability is not exploited to malicious aims. Lozenge shapes represent plans that agents occupying the roles roles will perform in order to achieve goals local to each role (shown as rounded rectangles). Individual goals (i.e., goals local to a role) are shown within the internal rationale of each role (i.e., the dotted line). Plans are related among themselves with plan decomposition relationships. Such a relationship indicates that one plan is part of the other, composite plan. A dependency incoming to a plan from a goal (label "D" turned towards the plan) means that the goal is achieved by performing the plan. A dependency outgoing from a plan to a goal (label "D" turned away from the plan) means that the goal must be achieved before the plan can be executed. Finally, a plan can be related to a goal with a means-ends relationship to indicate that successfully performing the plan satisfies the goal.

The visual representation of the functional ES is incomplete as the functional ES involves functional goals and preferences thereon, dependencies, plans, and domain constraints.

Functional Goals

Stakeholders' intentions and desires about the system are represented in RE using the concept of *goal*. While precise definitions vary [152], there is consensus in RE that goals restrict possible states of the system to those that satisfy requirements (see, e.g., [225, 11, 332, 55, 318, 50, 271]; for an overview, see [315]). Taxonomic dimensions we introduced earlier, i.e., functional/nonfunctional, hard/soft, and free/conditional can be immediately accommodated within the goal-oriented conceptualization.[4]

Definition 13.7. *Given a functional requirement, the corresponding functional goal is a logical constraint that restricts the possible states of the system only to those that satisfy the given requirement.*

As mentioned above, logical constraints are written as Horn clauses in this paper. Below is one way to formalize the the hard conditional functional requirement "Record a deposit in electronic and paper form if the deposit is received and approved."

UID:	Record deposit in el. & paper form
Type:	Goal (Functional/Compulsory/Conditional)
Preferences:	(Record deposit in electronic form only >> Record deposit)
Supergoals:	
BelongsTo:	Role (Banker)
Source:	Record a deposit in electronic and paper form if the deposit is received and approved.
Source type:	Requirement (Functional/Hard/Conditional)
Parameters:	d: Deposit
Satisfy:	$eL : elLog$, $pL : paLog$ $logs(eL, pL, d) \leftarrow isPaperLog(pL, d), isElectronicLog(eL, d)$
Conditions:	$received(d), approved(d)$
Concepts:	Deposit = (`and` Amount (> 0) (`exists` has-purpose Credit) (`all` in-currency `aset`(USD,EUR))); paLog = (`and` Log (`all` has-purpose TransactionRecord) (`all` recorded-on Database)); elLog = (`and` Log (`all` has-purpose TransactionRecord) (`all` recorded-on Paper))

The functional goal is specified using a template, in which slots have the following meaning:

- UDI: The value of the unique identifier (UID) relates the given fragment of the requirements specification with the fragment in a visual representation of the requirements specification, as in Figure 22.
- Type: The type of the goal classifies it along the taxonomic dimensions that correspond to those of requirements. To avoid terminological confusion with softgoals (which we discuss in relation to the nonfunctional requirements specification), hard requirements correspond to *compulsory* goals, soft requirements to *optional* goals.
- Preferences: indicates the preference order in which the given goal appears. Element on the left hand side of >> is preferred to the one on the right. As usual [39], the >> relation is modeled as a strict partial order, i.e., >> is anti-reflexive, anti-symmetric, and transitive.
- Supergoals: goal that has been refined to obtain the current goal. A functional requirement can have more than one corresponding functional goal. This occurs if the functional goal that satisfies the requirement is refined. To refine a functional goal, one replaces that goal with several other functional goals. The subgoals are such that, if all verify, then the refined goal (i.e., supergoal) also verifies [72].
- BelongsTo: identifies the role or dependency to which the goal is associated.
- Source: The source is the requirement which is satisfied if the functional goal verifies.
- Source type: The classification of the source.
- Parameters: The requirements engineer is interested in *generic* functional goals, such as "record deposit" or "issue letter of credit", which are instantiated at runtime for each particular deposit to record or letter of credit to issue. We thus highlight the parameters in the template.

[4] We have discussed elsewhere the similarities and differences among the various conceptualizations of goal in RE [151, 152]. While we do mention differences from related RE work herein, detailed discussions are given in the cited publications. The conceptualization proposed herein, and grounded in the cited work, is appropriate for the problem at hand, namely the transformation of requirements into NSI.

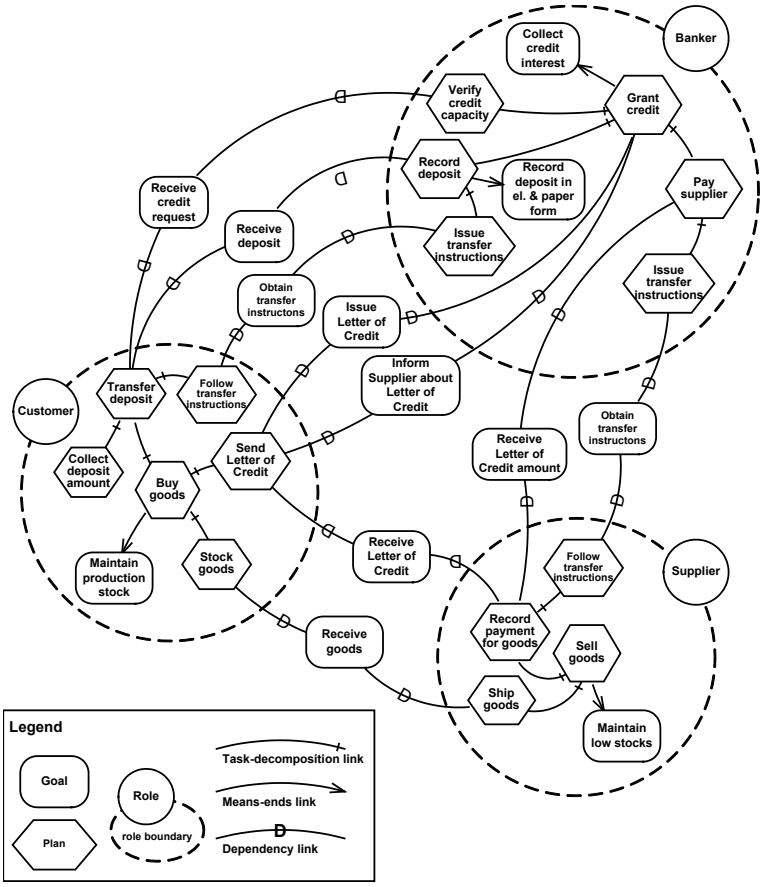


Fig. 22. A representation of functional requirements using a goal model.

- Satisfy: Logical constraints on parameters. If the conditions verify, the goal verifies.
- Conditions: Any conditions that must hold to make the goal relevant.
- Concepts: Delimits the meaning intended for terms used in the "satisfy" and "conditions" slots. Each term in these slots is associated with a concept from a local domain ontology. That is, the term is a reference to the concept. The concept definition is included in the "concepts" slot. The local domain ontology is written in an ontology definition language.

Plans

A functional goal is satisfied through the execution of appropriate plans. In open MAS, each agent advertises its capabilities. Plans can be seen as requests for capabilities, and thereby as restrictions on capabilities that can be potentially needed to perform the plan and consequently satisfy functional goals. In this respect, a plan mirrors capability advertisements. A capability is described in terms of its input and output signature, and constraints on inputs and outputs. In the Language for Advertisement and Request for Knowledge Sharing (LARKS), advertisements also carry the descriptions of the terms used in the capability, so that results of automated matchmaking between requests and capabilities can be of higher overall quality [306]. We have the following template for the "Transfer deposit" plan:

UID:	Transfer deposit
Type:	Plan
TargetGoals:	Maintain production stock
Preferences:	Transfer low deposit >> Transfer deposit
Superplans:	Buy goods
BelongsTo:	Role (Customer)
Input:	sourceAcctNum;

	destAcctNum; minDepAmt;
Output:	destAccountBalance;
InConstraints:	higherBalance(minDepAmt, sourceAcctNum);
OutConstraints:	higherBalance(minDepAmt, destAcctNum);

Concepts:	sourceAcctNum = (`and` Account (`all` held-at Bank) (`all` in-currency `aset`(USD,EUR)) (`all` source Account));
	destAcctNum = (`and` Account (`all` held-at Bank) (`exists` in-currency `aset`(USD,EUR)) (`exists` destination Account));
	minDepAmt = (`and` Amount (`ge` 50000) (`exists` in-currency `aset`(USD)) (`all` serves-for Deposit));
	destAccountBalance = (`and` Amount (`exists` in-currency `aset`(USD));

TxtDescription:	Transfer deposit amount from the source to destination account number.

Apart from the UID, Type, and Concept attributes, which have the meaning given earlier, we have plan-specific attributes:

- TargetGoals: the plan is executed in order to bring about the given goal. Executing the given plan is necessary but may not be sufficient for the target goal (i.e., the above plan may need to be executed in a sequence with other plans). The means-ends relationship in the visual representation, directed from the plan to a goal, or from a composite plan to a goal is noted in the template using the "Target goal" attribute.
- Preferences: indicate the preference order in which the given plan participates.
- Superplans: of which the given plan is part. The value can be obtained from the visual representation, as it corresponds to the task-decomposition link which is directed from the given task. We require that any decomposition be complete, that is: executing all subplans gives same results as those obtained by execuring the superplan.
- Input and Output: input/output variable declarations.
- InConstraints and OutConstraints: logical constraints on input/output variables that appear in the input/output declaration.
- TxtDescription: a natural language description of the plan.

A plan need not be specified with the detail described in the above example. If it is, the engineer places significant constraints on the pool of potential agents that can satisfy the plan. If fewer constraints are given, more agents are likely to carry capabilities that match the plan. In this respect, Input, Output, InConstraints, OutConstraints, and Concepts are optional attributes.

Domain Constraints

A domain constraint is associated to a particular domain assumption to indicate the logical constraints that must hold in order not to violate the assumption.

Definition 13.8. *Given a domain assumption, the corresponding domain constraint is a logical constraint that restricts the possible states of the system only to those that do not violate the given domain assumption.*

Below is the template for the domain constraint that corresponds to the domain assumption "Letter of Credit cannot be issued if there is no deposit."

UID:	Maintain credit deposit
Type:	Domain constraint (Free)
Source:	Letter of Credit cannot be issued if there is no deposit.
Source type:	Domain assumption (Free)
Uphold:	empty(letterOfCredit) $\leftarrow$ accountNumberBalance $\leq$ minDepAmt
Conditions:	creditAccount(accountNumber)
Concepts:	letterOfCredit = (`and` Contract (`exists` has-purpose Credit))

Dependencies

In a dependency relationship, a role (called "depender") depends on another role ("dependee") in order for a functional goal to be achieved. A dependency highlights that the agent occupying the depender role is vulnerable, in the sense that it depends on the behavior of the agent occupying the dependee role. While a dependency intuitively exists between two roles, it is specified as a relationship between plans performed by different roles. Namely, dependency exists if the depender cannot execute its plan if the agent occupying the dependee role has not executed some plan. We therefore have the following template for the dependency involving the "Receive deposit" functional goal:

UID:	Receive deposit
Type:	Dependency
Functional goal:	Receive deposit
Depender:	Banker
Depender plan:	Record deposit
Dependee:	Customer
Dependee plan:	Transfer Deposit

13.3.2 Nonfunctional ES

Given nonfunctional requirements, the aim of the nonfunctional ES is to delimit system states only to those that satisfy to the most preferred and feasible extent the nonfunctional requirements. As the nonfunctional ES must be verifiable, we define measures and preferences over alternative values for measures.

A measure quantifies an observable and measurable characteristic of system behavior, such as the time to respond to a query, the proportion of runs that satisfy the nonfunctional requirements to some particular level, the average number of runs over a period that fail to satisfy minimal levels of nonfunctional requirements, the proportion of time some agent is available, the the number of clicks it takes the user to obtain some information, and so on. We have observed earlier that a nonfunctional requirement communicates attitude using a gradable adjective without established degrees, and/or scale, and/or dimension. Given a gradable adjective such as, e.g., "quick" associated to "Issue Letter of Credit" to point out that the letter of credit is to be issued in short time, measures serve as indicators for stakeholders when evaluating whether the letter of credit is issued quickly enough or not. It is through measures that we establish degrees, scales, and dimensions intended to act as proxies for the gradable adjective in the nonfunctional requirement.

A measure is used in a nonfunctional goal when defining preferred values (in the "Preferences" attribute) and threshold values that are to be maintained (the "Satisfy" attribute). The measure itself is defined in the domain ontology (and given in the nonfunctional goal's "Concepts" attribute), which allows the engineer to freely choose a measure definition method (e.g., GQM/MEDEA [43]) and associate to the metric the attributes relevant in the chosen method. Below is the nonfunctional goal template for the nonfunctional requirement "Quickly issue Letter of Credit", which uses the measure LoCIssuingTime to define a threshold waiting time for a Letter of Credit.

UID:	Quick LoC issuing time
Type:	Goal (Nonfunctional/Compulsory/Free)
Preferences:	minimize(monthAverage(LoCIssuingTime))
Supergoals:	Conveniently obtain Letter of Credit
FunctionalSource:	Issue Letter of Credit
Source:	Quickly issue Letter of Credit
Source type:	Requirement (Nonfunctional/Hard/Free)

Satisfy:	monthAvgLoCIssTime < 4 BusinessDay;
Conditions:	
Concepts:	LoCIssuingTime = (`and` Duration (`exists` in-unit `aset`(BusinessDay));

The "FunctionalSource" of the nonfunctional goal is either a functional goal or a plan. Measures can be aggregated or otherwise combined in a meaningful manner to facilitate observation of system behavior and control thereof. A nonfunctional goal can therefore be refined onto several other goals, provided it places constraints on several measures and/or a measure obtained by aggregation/combination of two or more other measures. In the above example, the given nonfunctinal goal is a subgoal of the "Conveniently obtain Letter of Credit". The latter places constraints on the time needed to issue the Letter of Credit, while also restricting other measures, such as the time needed for the Banker to respond to the Letter of Credit request, the minimum deposit amount, the interest rate, and the quantity of information exchanged between the Customer and the Banker.

13.3.3 Priorities ES

Templates defined for functional and nonfunctional ES describe preferences within the "Preferences" attribute. Given that preferences are distributed accross the ES, the templates in Priorities ES refer to the UIDs of templates carrying conflicting preferences.

Consider the nonfunctional goal "Extensive credit capacity verification" for the "Verify credit capacity" plan, according to which it is preferred that as many as available checks be performed to minimize risk involved in granting credit. Performing extensive checks increases the time needed to issue a Letter of Credit, so that the preference defined for the said nonfunctional goal conflicts with preferences defined for the "Quick LoC issuing time" nonfunctional goal. If stakeholders deside that lowering the risk is more important than quickly granting credit, we have the following priority template:

UID:	Credit checks more important than LoC issuing time
Priority:	(Extensive credit capacity verification >>> Quick LoC issuing time)

The priority relation >>> is modeled as a strict partial order. Given a preference order X and another preference order Y, X >>> Y indicates that satisfying the preference order X is more important than satisfying Y.

13.3.4 Terminological ES

The Terminological ES defines the domain ontology applicable to the open MAS. The ontology delimits the intended meaning of terms used throughout the ES. The ontology ensures that the terms used in the ES carry the same meaning when placed within NSI. By enforcing the same ontology at the ES and governance levels, we aim to avoid terminological mismatch between these two levels. The ontology also helps improve the overall enforcement of NSI onto agents. Enforcement involves checking agents' capabilities against NSI so as to identify whether agents can act and/or under what constraints. If the ontology is available, semantics are associated to norms, sanctions, and agent capability advertisements. Checking then not only relies on keyword retrieval, but also on the semantics of NSI, and of agent's capability advertisements.

In an ES template, the "Concepts" attribute lists the definitions of concepts whose instances are referred to in the template. All of the concepts are defined within the ontology. We use the ITL (Information Terminological Language) [306, 305]. Therein, conceptual knowledge about a given application domain is defined by a set of concepts and roles these concepts play in relationships, in which they take part. Each term intended to define a concept C is a conjunction of logical constraints, which are necessary for any object to be an instance of C. The set of related ITL definitions gives one form of ontology: a terminology. Any definition of concepts in a terminology relies on:

- a set of concepts and roles already defined in the terminology and/or
- a given basic vocabulary of words (primitives) which are not defined in the terminology, that is, their semantics are assumed to be known and consistently used.

The following partial terminology is written in ITL and covers most of the concepts used in the earlier examples in the banking domain:

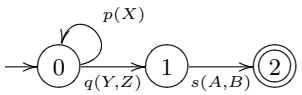


Fig. 23. Sample VO as a finite state machine

Bank	=	`aset`(UBS,Citi,BankOfAmerica,HSBC,CreditAgricole,RBS)
Amount	=	(`and` Real (`all` in-currency `aset`(USD,EUR)))
Credit	=	(`and` Amount (`all` pays Interest))
Interest	=	(`and` Amount (`exists` in-duration `aset`(Day,Week,Month)))
Deposit	=	(`and` Amount (> 0) (`exists` has-purpose Credit) (`all` in-currency `aset`(USD,EUR)))
Account	=	(`and` BankProduct (`and` has-owner Person) (`and` has-amount Amount) (`exists` in-currency `aset`(USD,EUR)))
sourceAcctNum	=	(`and` Account (`all` held-at Bank) (`all` in-currency `aset`(USD,EUR)) (`all` source Account))
destAcctNum	=	(`and` Account (`all` held-at Bank) (`exists` in-currency `aset`(USD,EUR)) (`exists` destination Account))
minDepAmt	=	(`and` Amount (`ge` 50000) (`exists` in-currency `aset`(USD)) (`all` serves-for Deposit))
LoCIssuingTime	=	(`and` Duration (`exists` in-unit `aset`(BusinessDay))
paLog	=	(`and` Log (`all` has-purpose TransactionRecord) (`all` recorded-on Database))
elLog	=	(`and` Log (`all` has-purpose TransactionRecord) (`all` recorded-on Paper))

13.4 Contracts

Requirements defined for an open MAS, that is, a virtual organization (VO) are enforced through VO governance. Governance measures take the form of contracts to which the agents engage when entering the VO. The purpose of contracts is therefore to ensure that the heterogenous agents act only in ways that satisfy requirements, are in accord with domain assumptions and priorities, and satisfy preferences to the maximal feasible extent. Proper definition of contracts is therefore a critical activity when developing and managing the operation of VO. Unexpected and undesired behavior is bound to be observed in the VO if contracts are incoherent with requirements.

In order to establish the relationships between the various templates in the environment specification and the information in contracts, we first define VO in more detail. This allows us to define the notions of VO roles, contracts, and governance measures that go into contracts (i.e., obligations, permissions, prohibitions, and sanctions). We subsequently show how the environment specification relates to the various governance measures, and how the latter are derived from the former.

13.4.1 Virtual Organizations

VOs [235] enable various heterogenous agents to coordinate resource sharing and problem solving activities. The VO paradigm has found strong applications in Web service orchestration [194], e-Science [229], and the Grid [95]. In their most generic formulation, VOs coordinate many software and human agents when these engage in sophisticated forms of interaction.

We formally represent VOs as finite-state machines, in which the actions of individual agents label the edges between discrete states. Although a VO can be represented in a much more sophisticated way (using, e.g., AUML [249] and electronic institutions [274]), we choose the said representation to make the discussion generic. We reasonably assume that any higher-level formalism can be mapped onto a finite state machine (possibly with some loss in expressiveness). Figure 23 shows a simple VO represented as a finite state machine.

Definition 13.9. *A virtual organization $\mathcal{I}$ is the triple $\langle S, s_0, E, T\rangle$ where $S = \{s_1, \ldots, s_n\}$ is a ginite and non-empty set of states, $s_0 \in S$ is the initial state, E is a finite set of edges $(s, s', \phi), s, s' \in S$ connecting s to s' with a first-order atomic formula ϕ as a label, and $T \subseteq S$ is the set of terminal state.*

Edges are directed, so $(s, t, \phi) \neq (t, s, \phi)$. The sample VO in Figure 23 is formally represented as $\mathcal{I} = \langle\{0, 1, 2\}, 0, \{(0, 0, p(X)), (0, 1, q(Y, Z)), (1, 2, s(A, B))\}, \{2\}\rangle$. We assume an implicit existential quantification on any variable in ϕ, so that, for instance, $s(A, B)$ stands for $\exists A, B\ s(A, B)$.

An agent can only enter VO if it engages in a contractual relationship. The NSI given in the contract determine the *role* that the agent occupies within the VO. The VO can therefore be seen as involving a collection of agents, each playing a role within the AO. Once the agent starts to occupy a role, it can act within the VO. There are two desirable sources of uncertainty in VO, appearing in two kinds of decision situations for agents. In one, the agent can choose one of the edges leaving a state; the other arises when the agent instantiates variables in formulas that label edges. Such freedom in acting is desirable for it enables flexible coordination within VO. We do, however need to record the actions that agents take. Although the actions comprising a VO are carried out distributedly, we use an explicit global account of all events. This is achieved in practice by requiring individual agents to declare whatever actions they have carried out. Agents are therefore either assumed truthful, or we introduce *governor agents* [102] for surveillance purposes. In order to record the authorship of the action, we annotate the formulas with the agents' unique identification. Our explicit global account of all events is a set of ground atomic formulae φ, that is, we only allow constants to appear as terms of formulae. Each formula is a truthful record of an action specified in the VO. Notice, however, that in the VO specification we do not restrict the syntax of the formulas: variables may appear in them, and when an agent performs an actual action then any variables of the specified action must be assigned values. We thus define:

Definition 13.10. *A global execution state of a VO, denoted as Ξ, is a finite, possibly empty, set of tuples $\langle a : r, \bar{\varphi}, t\rangle$ where $a \in$ Agents is an agent identifier, $r \in$ Roles is a role label, $\bar{\varphi}$ is a ground first-order atomic formula, and $t \in \mathbb{N}$ is a time stamp.*

For instance, $\langle ag_1 : buyer, p(a, 34), 20\rangle$ states that agent ag_1 adopting role *buyer* performed action $p(a, 34)$ at instant 20. Given a VO $\mathcal{I} = \langle S, s_0, E, T\rangle$, an execution state Ξ and a state $s \in S$, we can define a function which obtains a possible next execution state, *viz.*, $h(\mathcal{I}, \Xi, s) = \Xi \cup \{\langle a : r, \bar{\varphi}, t\rangle\}$, for one $(s, s', \varphi) \in E$. Such function h must address the two kinds of non-determinism above, as well as the choice on the potential agents that can carry out the action and their adopted roles. We also define a function to compute the set of all possible execution states, $h^*(\mathcal{I}, \Xi, s) = \{\Xi \cup \{\langle a : r, \bar{\varphi}, t\rangle\} | (s, s', \varphi) \in E\}$.

13.4.2 VO Governance through Contracts

VO governance consists of ensuring that agents entering the VO perform as expected, that is, act and interact in accordance to the ES. Governance amounts to defining roles within VO. A role is defined through a collection of contracts, whereby each contract is a collection of relevantly related contractual atoms. We say that a set of contractual atoms is relevantly related if all given contractual atoms need to be obeyed in order to satisfy a functional goal or some other delimited fragment of the ES.

Given a deliberative agent architecture, in which the agent's actions and interactions depend on its beliefs, desires, and intentions, a role must define constraints on beliefs, desires, and intentions so as to limit agent's behavior to that expected in the VO. A role is itself not an agent, for it lacks capability to reason and act—it is instead an organizational construct [245]. This relationship between the concept of role and the mental states is critical herein, for it brings two benefits. First, it allows us to define contractual atoms in relation to mental states that they restrict. Second, mental states permeate the two higher levels of abstraction we have discussed above: we have seen that statements at the requirements level relate to mental states through illocutionary acts, and subsequently shown how these statements correspond to concepts that intervene in the ES. We therefore ensure conceptual consistency by using mental states concepts as a basic ontology that underlies and allows us to relate concepts at all three levels of abstraction in RdC. Contractual atoms are *obligations*, *prohibitions*, *permissions*, and *sanctions*.[5]

While contracts define rights and responsibilities between contracting parties, ensuring that a contract is honored requires a third party acting as a supervising authority. The special, *authority* role is a necessary role in VO that empowers the agent occupying it to monitor and ensure that a contract is honored and apply sanctions in case contractual provisions are violated. Intervention of the authority in contract execution, that is *supervised interaction* [179, 180, 181] is a pattern of interaction involving three roles. One is the authority, acting as an impartial witness and sanction enforcer to the contract established between two other roles. The other two roles engage in the contractual relationship to transfer value, information, goods, or to perform services. To enforce contracts, the authority relies on sanctions.

[5] We referred to the first three as kinds of *norms* throughout the text.

Definition 13.11. *A sanction defines (i) logical conditions that hold if an obligation or prohibition is violated, and (ii) the action that the agent occupying the authority role must execute once the violation verifies.*

Sanctions are coercive. We have seen earlier that there are optional goals, corresponding to soft requirements. Being optional, their satisfaction cannot involove coercion. Instead, it involves incentives, that is measures introduced to motivate without coercion the agents participating in VO to act in order to satisfy soft requirements in addition to hard requirements.

Definition 13.12. *An incentive defines (i) logical conditions that hold if the incentive is to be provided, and (ii) the action of providing the incentive, executed by the agent occupying the authority role.*

Below, let ψ be a logical constraint, written as an atomic first-order sentence.

Definition 13.13. *An obligation of ψ applies to agent a (through the role that the agent occupies in the VO) if all of the following conditions are true:*

- *the authority agent desires that ψ be brought about;*
- *not bringing about ψ by a is considered as a violation by the authority;*
- *the authority applies a sanction if there is a violation of ψ.*

Obligations can therefore be seen as motivating agents to satisfy logical conditions by executing actions. Given an obligation, a norm-governed agent, that is, one occupying a role in the norm-governed VO will select and execute appropriate actions among those available to it. The aim of prohibitions is to make sure that agents do not bring about states in which some specific logical conditions hold. Permissions in contrast explicitly allow agents when acting to bring about some conditions.

Definition 13.14. *A prohibition of ψ applies to agent a (through the role that the agent occupies in the VO) if all of the following conditions are true:*

- *the authority agent desires that ψ is not brought about;*
- *bringing about ψ by a is considered as a violation by the authority;*
- *the authority applies a sanction if there is a violation of ψ.*

Definition 13.15. *A permission of ψ applies to agent a (through the role that the agent occupies in the VO) if bringing about ψ by a is not considered as a violation by the authority.*

Sanctions are by definition conditional, as they are applied only if an obligation or prohibition is violated. We also allow obligations, prohibitions, and permissions to be conditional to constraints. Given the earlier formal model of VO, we define NSI therein as follows.

Definition 13.16. *A constraint, represented as κ is a first-order atomic sentence (i.e., predicate over terms). Some constraints are of the form the form: $\tau \lhd \tau'$, where $\lhd \in \{=, \neq, >, \geq, <, \leq\}$. τ and τ' are first-order terms, that is, constants, variables, or functions (themselves applied to terms).*

Definition 13.17. *A norm ω is a tuple $\langle \nu, t_d, t_a, t_e \rangle$ where ν is any of the following three:*

- *An obligation, $\mathsf{O}_{\tau_1:\tau_2}\phi \wedge \bigwedge_{i=0}^{n} \kappa_i$;*
- *A permission, $\mathsf{P}_{\tau_1:\tau_2}\phi \wedge \bigwedge_{i=0}^{n} \kappa_i$;*
- *A prohibition, $\mathsf{F}_{\tau_1:\tau_2}\phi \wedge \bigwedge_{i=0}^{n} \kappa_i$;*

where τ_1 and τ_2 are first-order terms, ϕ is a first-order atomic formula and $\kappa_i,\ 0 \leq i \leq n$ are constraints. The elements $t_d, t_a, t_e \in \mathbb{N}$ are, respectively, the time when ν was declared (introduced), when ν becomes active, and when ν expires, $t_d \leq t_a \leq t_e$.

Term τ_1 identifies the agent(s) to which the norm applies and τ_2 is the role that each of these agents occupies. $\mathsf{O}_{\tau_1:\tau_2}\phi \wedge \bigwedge_{i=0}^{n} \kappa_i$ thus represents an obligation on agent τ_1 occupying role τ_2 to bring about ϕ, if and when all constraints $\kappa_i,\ 0 \leq i \leq n$ hold. The constraints place limits on variables occuring in ϕ. In line with the use of Horn rules in the ES, only conjunctions of constraints. If disjunctions are required, a norm must be established for each disjunct. For example, if we introduce the permission $\mathsf{P}_{A:R}move(X) \wedge (X < 10 \vee X = 15)$, we must break it into two permissions, $\mathsf{P}_{A:R}move(X) \wedge (X < 10)$ and $\mathsf{P}_{A:R}move(X) \wedge (X = 15)$. This applies beacuse of implicit universal quantification over variables in ν. For instance, $\mathsf{P}_{A:R}p(X, b, c)$ stands for $\forall A \in Agents, \forall R \in Roles, \forall X\ \mathsf{P}_{A:R}(X, b, c)$.

Our earlier definition of the sanction and incentive concepts lead to the following corresponding conceptualizations in the formal model of norm-governed VO.

Definition 13.18. *A sanction* ς^- *is a tuple* $\langle \omega, \bigwedge_{j=0}^{m} \kappa_j, \mathcal{A}_{A:R} \rangle$*, where:*

- ω *is a norm in which* ν *is either an obligation or a prohibition;*
- $\bigwedge_{j=0}^{m} \kappa_j$ *is the conjunction of constraints—if* $\bigwedge_{j=0}^{m} \kappa_j$ *holds, then* $\mathcal{N}$ *is violated;*
- $\mathcal{A}$ *is the action that the agent* A *occupying the authority role* R *must execute once* $\mathcal{N}$ *is violated.*

Definition 13.19. *An incentive* ς^+ *is a tuple* $\langle \omega, \bigwedge_{j=0}^{m} \kappa_j, \mathcal{A}_{A:R} \rangle$*, where:*

- ω *is a norm in which* ν *is a permission;*
- $\bigwedge_{j=0}^{m} \kappa_j$ *is the conjunction of constraints—if* $\bigwedge_{j=0}^{m} \kappa_j$ *holds, the incentive is to be provided;*
- $\mathcal{A}$ *is the action that the agent* A *occupying the authority role* R *must execute once the incentive is to be provided.*

A contract is defined as a collection of tuples, one per role involved in the contract. Each tuple carries the role identifier and the norms, sanctions, and incentives associated to the role in the given contractual relationship.

Definition 13.20. *A contract* $\boldsymbol{C}$ *among* m *roles is a tuple:*

$$\langle \langle r_1, norms, sanctions, incentives \rangle_{r_1}, \ldots, \langle r_m, norms, sanctions, incentives \rangle_{r_m} \rangle$$

where each r_i *is a role identifier and norms, sanctions, and incentives are possibly empty collections of, respectively, norms, sanctions, and incentives.*

We formally represent the obligations, permissions, and prohibitions of all agents in a VO from a global perspective.

Definition 13.21. *A global normative state* Ω *is a finite and possibly empty set of norms* ω*, sanctions* ς^- *and incentives* ς^+*.*

Global normative states complement the execution states of a VO with information on the normative positions of individual agents. We can relate them via a function to obtain a norm-regulated next execution state of a VO, that is, $g(\mathcal{I}, \Xi, s, \Omega, t) = \Xi'$, with t being the time of the update. For instance, we might want all prohibited actions to be excluded from the next execution state, that is, $g(\mathcal{I}, \Xi, s, \Omega, t) = \Xi \cup \{\langle a : r, \bar{\phi}, t \rangle\}$, $(s, s', \phi) \in E$ and $\langle \mathsf{F}_{a:r}\phi, t_d, t_a, t_e \rangle \notin \Omega, t_a \leq t \leq t_e$. We might also be interested that only permitted actions be chosen for the next execution state. We do not recommend any particular way to regulate VOs, offering instead the basics that allow arbitrary policies to be put in place. In the same way that a normative state is useful to obtain the next execution state of a VO, we can use an execution state to update a normative state. For instance, we might want to remove any obligation specific to an agent and role, which has been carried out by that specific agent and role, that is $f(\Xi, \Omega) = \Omega - Obls$, $Obls = \{\langle \mathsf{O}_{a:r}\phi, t_d, t_a, t_e \rangle \in \Omega | \langle a : r, \bar{\phi}, t \rangle \in \Xi\}$.

13.4.3 From ES to Contracts

As the purpose of contracts is to ensure that a VO behaves according to stakeholders' requirements and obeys domain assumptions, we use the following automated techniques (i.e., algorithms) for the writing of contracts from the ES. We refer below to all contracts as the *contract specification* (CS). To simplify the writing of the transformation algorithms, we introduce some notational conventions first.

Let G be some goal in ES, and G.UID be the value of the UID attribute of G. We refer to the values of the other attributes in the obvious way (e.g., G.Satisfy for the value of the Satisfy attribute in the template of G). We also let NG be some nonfunctional goal, P some plan, DC some domain constraint, DD some dependency, and PP some priority in ES. Following earlier discussions, we assume that the value of the Satisfy, InConstraints, OutConstraints, and Uphold attributes is of the form $a_0 \leftarrow a_1, \ldots, a_n$ whereby $n \geq 1$ and a_0 can be empty, and this for all elements having any of these attributes in ES. We also take that the value of Conditions attribute wherever it appears in ES is of the form $b_0 \leftarrow b_1, \ldots, b_m$ whereby $m \geq 1$ and b_0 can be empty. Recall that any $a_i, 0 \leq i \leq n$ and any $b_j, 0 \leq j \leq m$ is an atomic first-order formula. Also, we write $(\cdot)$ for content of no interest in the particular discussion or procedure—e.g., if we write the norm $\langle (\cdot), \bigwedge_{j=1}^{m} \kappa_j, (\cdot), (\cdot), (\cdot) \rangle$, we are interested only in $\bigwedge_{j=1}^{m} \kappa_j$ while the content of the other elements can be any allowed by the definition for the norm concept.

Contracts from Functional and Nonfunctional Goals

Given the ES, we first select a goal that has not yet been subjected to the algorithm. Given that a goal can be refined, we only take lowest-level goals, that is, those that are not supergoals. In lines 2–12, we check if the goal is compulsory. If it is, we generate as many obligations as needed to cover the Horn clause(s) in the goal's Satisfy attribute, and we create corresponding sanctions activated if the obligations are violated. In case the goal is conditional in addition to being compulsory, we create constraints that correspond to the conditions. If the goal is instead optional (lines 13–23), we do not generate obligations but permissions, and then define incentives in place of sanctions. The treatement of functional vs. nonfunctional goals differs in the way that the contracts are created. If the goal is functional, we create a new contract (lines 24–28). The roles referred in the new contract then vary if the functional goal belongs to a role or a dependency. In case of G being a nonfunctional goal, there are no new contracts to write (lines 29–34). Instead, contracts obtained by transforming the functinal goals in G.FunctionalSource are updated so that new obligations (or permissions, depending on the type of G) and sanctions (or incentives) and are added to the relevant roles in all of these contracts.

Contract from Goal (RdC-CG)	
`Input:`	One goal G such that: (i) G has not been transformed previously; (ii) there are no goals in ES for which G is the supergoal.
`Output:`	If G is a functional goal, then one contract including norms, sanctions, and/or incentives for ensuring that logical conditions of G must or can be brought about. If G is a nonfunctional goal, then already existing contracts on all functional goals listed in G.FunctionalSource are updated to include norms, sanctions, and/or incentives for ensuring that logical conditions of the nonfunctional goal G must or can be brought about.
`begin`	
1	**If** G.Type is Compulsory **then**
2	**For each** $a_i, 1 \leq i \leq n$ in G.Satisfy **do**
3	Create obligation $\omega_i = \langle \mathsf{O}_{a:r}\phi \wedge \bigwedge_{j=1}^{m} \kappa_j, t_d, t_a, t_e \rangle$ where $\phi = a_i$ and t_d, t_a, t_e are recorded automatically.
4	**If** G.Type is Conditional **then**
5	**For each** $\kappa_j, 1 \leq j \leq m$, $\kappa_j = b_j$ **do**
6	**End If.**
7	**If** G.Type is Free **then**
8	**For each** $\kappa_j, 1 \leq j \leq m$, **do** κ_j is empty and b_j is empty.
9	**End If.**
10	Create sanction $\varsigma_i^- = \langle \omega_i, \kappa_i, \mathcal{A}_{a:r} \rangle$ where $\kappa_i = \neg a_i$ and the engineer is asked to define $\mathcal{A}_{a:r}$.
11	**End If.**
12	**If** G.Type is Optional **then**
13	**For each** $a_i, 1 \leq i \leq n$ in G.Satisfy **do**
14	Create permission $\omega_i = \langle \mathsf{P}_{a:r}\phi \wedge \bigwedge_{j=1}^{m} \kappa_j, t_d, t_a, t_e \rangle$ where $\phi = a_i$ and t_d, t_a, t_e are recorded automatically.
15	**If** G.Type is Conditional **then**
16	**For each** $\kappa_j, 1 \leq j \leq m$, $\kappa_j = b_j$ **do**
17	**End If.**
18	**If** G.Type is Free **then**
19	**For each** $\kappa_j, 1 \leq j \leq m$, **do** κ_j is empty and b_j is empty.
20	**End If.**
21	Create incentive $\varsigma_i^+ = \langle \omega_i, \kappa_i, \mathcal{A}_{a:r} \rangle$ where $\kappa_i = a_i$ and the engineer is asked to define $\mathcal{A}_{a:r}$.
22	**End If.**
23	**If** G.Type is Functional **then**
24	Create contract $\mathbf{C} = \langle \langle r_1, \{\omega_i | 1 \leq i \leq n\}, (\cdot), (\cdot) \rangle_{r_1}, \langle r_2, (\cdot), \{\varsigma_i^- | 1 \leq i \leq n\}, \{\varsigma_i^+ | 1 \leq i \leq n\} \rangle_{r_2} \rangle$ where:

25 **If** G.BelongsTo is Role **then** r_1 is the name of that role and r_2 is an authority role. **End If.**
26 **If** G.BelongsTo is Dependency **then** r_1 is the name of the depender role and r_2 is an authority role.
27 **End If.**
28 **If** G.Type is Nonfunctional **then**
29 **For each** functional goal G_k in G.FunctionalSource **do**
30 **For each** contract obtained by transforming G_k **do**
31 Add obligations (resp. permissions) $\{\omega_i | 1 \leq i \leq n\}$ to the role in the contract that carries the obligations (resp. permissions). That role is the supervised role.
32 Add sanctions $\{\varsigma_i^- | 1 \leq i \leq n\}$ (or incentives $\{\varsigma_i^+ | 1 \leq i \leq n\}$) to the authority (i.e., supervisor) role in the contract.
33 **End If.**
end

The algorithm is correct: it returns for each goal, a set of norms, sanctions and/or incentives according to the type of the goal. Obligations are generated if the goal is compulsory, while permissions are returned if the goal is instead optional. Sanctions are defined for cases in which obligations are not fulfilled, whereas incentives are defined to apply when permitted states are achieved. Contracts are created and norms, sanctions, and incentives distriubuted between the role being supervised and the authority (i.e., supervisor) role. The supervisor role receives responsibility for sanctions and/or incentives, while the supervised role receives obligations and permissions. In case of a nonfunctional goal, the obligations and/or permissions are handed over to supervised roles in all contracts on functional goals listed in the nonfunctional goal's FunctionalSource attribute, the sanctions and/or incentives are handed over to supervisor roles in these same respective contracts. It is apparent that the algorithm terminates, as each of the **for each** loops goes through finite sets and considers one element at a time.

We obtain the following contract on the functional goal "Record deposit in el. & paper form":

$$\langle\langle Banker, \{\omega_1, \omega_2\}, \emptyset, \emptyset\rangle, \langle BankerSupervisor, \emptyset, \{\varsigma_1^-, \varsigma_2^-\}, \emptyset\rangle\rangle$$

where:

$$\omega_1 = \langle \mathsf{O}_{a:Banker} isPaperLog(pL, d), received(d) \wedge approved(d), t_d, t_a, t_e\rangle$$
$$\omega_2 = \langle \mathsf{O}_{a:Banker} isElectronicLog(pL, d), received(d) \wedge approved(d), t_d, t_a, t_e\rangle$$
$$\varsigma_1^- = \langle \omega_1, \neg isPaperLog(pL, d), \mathcal{A}^{\varsigma_1^-}_{a:BankerSupervisor}\rangle$$
$$\varsigma_2^- = \langle \omega_2, \neg isElectronicLog(pL, d), \mathcal{A}^{\varsigma_2^-}_{a:BankerSupervisor}\rangle$$

Above, the *BankerSupervisor* is the role that acts as the authority for the *Banker* role. The functional goal gives us two obligations for the *Banker* role and the corresponding sanctions enforced by the *BankerSupervisor* role. $\mathcal{A}^{\varsigma_1^-}_{a:BankerSupervisor}$ and $\mathcal{A}^{\varsigma_2^-}_{a:BankerSupervisor}$ are actions defined by the engineer or selected from a repository; these actions determine how the sanction is applied in case the relevant obligations are violated.

The transformation of nonfunctional goals is also straightforward. If we take the nonfunctional goal "Quick LoC issuing time" discussed earlier, we can immediately notice that it will give rise to two obligations, one per logical constraint in its Satisfy attribute, and two corresponding sanctions that apply depending on whether the logical constraints verify.

Contracts from Plans

Given an ES plan, the kinds of norms we generate, along with the accompanying sanctions and incentives, depend on the kinds of goals that appear in the plan's TargetGoals attribute. Given that a plan can be decomposed, and that it is always fully decomposed (see, §13.3.1) we only transform lowest-level plans. For notational simplicity below, assume that the plan template gives logical constraints according to the following rules:

- Each element in the plan's Input attribute gives a predicate $input((\cdot))$ where $(\cdot)$ is replaced with the name of the input variable.

- Each element in the plan's Output attribute gives a predicate $output((\cdot))$ where $(\cdot)$ is replaced with the name of the output variable.
- Each logical constraint in the plan's InConstraints remains in the same form if it is alone or in the body of a Horn clause (i.e., for a clause $a_0 \leftarrow a_1, a_2$ we keep only a_1 and a_2 and count them as separate since we need atomic first-order formulas; we do not keep a_0).
- We apply the same approach as for InConstraints to the logical constraints in the plan's OutConstraints attribute.

We assume that there are in total n atomic first-order formulas, denoted $p_i, 1 \leq i \leq n$ that we obtain by applying the above four rules to a plan template. Notice that, because a plan is applied to satisfy a goal, the conditions of a goal are the conditions to apply a plan. We therefore must ensure that these conditions are carried over to the norms obtain by transforming a plan. To further simplify the notation, we assume that each goal gives m atomic first-order formulas, denoted (same as above) $b_j, 0 \leq j \leq m$ that need to be carried over as conditions to norms. Notice furthermore that we need to ensure that the NSI derived from the plan obey all constraints obtained from nonfunctional goals associated to the functional goals mentioned in the plan's TargetGoals attribute. We comment the procedure below.

Contract from Plan (RdC-CP)

`Input:` One P from ES such that: (i) P has not been transformed previously; (ii) there are no plans in ES for which P is the superplan.

`Output:` As many contracts as there are functional goals in P.TargetGoals, whereby each contract includes norms, sanctions, and/or incentives to ensure that the logical conditions given in the plan must or can be brought about.

`begin`

`1` **For each** goal G_k in P.TargetGoals **do**

`2` **If** G_k.Type is Compulsory **then**

`3` **For each** $p_i, 1 \leq i \leq n$ from P **do**

`4` Create obligation $\omega_{k,i} = \langle \mathsf{O}_{a:r}\phi \wedge \bigwedge_{j=1}^{m} \kappa_j, t_d, t_a, t_e \rangle$ where $\phi = p_{k,i}$ and t_d, t_a, t_e are recorded automatically.

`5` **If** G_k.Type is Conditional **then**

`6` **For each** $\kappa_j, 1 \leq j \leq m$ **do** $\kappa_j = b_{k,j}$.

`7` **End If.**

`8` **If** G_k.Type is Free **then**

`9` **For each** $\kappa_j, 1 \leq j \leq m$ **do** κ_j is empty.

`10` **End If.**

`11` Create sanction $\varsigma^{-}_{k,i} = \langle \omega_{k,i}, \kappa_i, \mathcal{A}_{a:r} \rangle$ where $\kappa_i = \neg p_{k,i}$ and the engineer is asked to define $\mathcal{A}_{a:r}$.

`12` **End If.**

`13` **If** G.Type is Optional **then**

`14` **For each** $p_i, 1 \leq i \leq n$ from P **do**

`15` Create permission $\omega_{k,i} = \langle \mathsf{P}_{a:r}\phi \wedge \bigwedge_{j=1}^{m} \kappa_j, t_d, t_a, t_e \rangle$ where $\phi = p_{k,i}$ and t_d, t_a, t_e are recorded automatically.

`16` **If** G_k.Type is Conditional **then**

`17` **For each** $\kappa_j, 1 \leq j \leq m$ **do** $\kappa_j = b_{k,j}$.

`18` **End If.**

`19` **If** G_k.Type is Free **then**

`20` **For each** $\kappa_j, 1 \leq j \leq m$ **do** κ_j is empty.

`21` **End If.**

`22` Create incentive $\varsigma^{+}_{k,i} = \langle \omega_{k,i}, \kappa_i, \mathcal{A}_{a:r} \rangle$ where $\kappa_i = p_{k,i}$ and the engineer is asked to define $\mathcal{A}_{a:r}$.

`23` **End If.**

`24` **For each** k **do** create contract $\mathbf{C}_k = \langle \langle r_{k,1}, \{\omega_{k,i} | 1 \leq i \leq n\}, (\cdot), (\cdot) \rangle_{r_{k,1}}, \langle r_{k,2}, (\cdot), \{\varsigma^{-}_{k,i} | 1 \leq i \leq n\}, \{\varsigma^{+}_{k,i} | 1 \leq i \leq n\} \rangle_{r_{k,2}} \rangle$ where:

`25` **If** G_k.BelongsTo is Role **then** $r_{k,1}$ is the name of that role and $r_{k,2}$ is an authority role.

26 **If** G_k.BelongsTo is Dependency **then** $r_{k,1}$ is the name of the depender role and $r_{k,2}$ is an authority role.
end

The first part of the algorithm produces several sets of NSI from the logical constraints $p_i, 1 \leq i \leq n$. A set of n obligations and corresponding sanctions is produced if the k-th goal is compulsory (lines 3–13), whereby constraints for the obligations are obtained from the conditions associated to the k-th goal. Deriving constraints in this way is due to the fact that a plan is executed only if a goal is to be achieved, and the goal is to be achieved when its conditions hold. Hence, conditions must be carried over to the corresponding plan(s). If the k-th goal is free, there are no constraints in the obligations. Given that we generate obligations, we also create sanctions that are appropriate with regards to the obligations. If the k-th goal is optional (lines 14–24), we generate, as we did in the earlier algorithm (§13.4.3), permissions and incentives from the $p_i, 1 \leq i \leq n$ logical constraints. The last part of the algorithm (lines 25–27) defines contracts, one per goal to achieve by executing the plan. Contracts are assigned to roles depending on whether the relevant goal belogs to a role or a dependency.

The algorithm is correct. It returns as many contracts as there are goals listed in the plan's TargetGoals attribute. Depending on whether each considered goal is compulsory or optional, the algorithm generates, respectively obligations (and corresponding sanctions) or permissions (and corresponding incentives) to bring about states in which the logical conditions of the plan hold. One contract is created per goal in the plan's TargetGoals attribute, whereby obligations/permissions are allocated to the supervised role and sanctions/incentives to the supervisor role. We therefore ensure that the constraints on the system imposed by the specification of a plan in the ES correspond to a collection of contracts, one contract per goal associated to the plan. It is straightforward to see that the algorithm terminates as all **for each** loops moves through finite sets and considers one element at the time.

We obtain the following contract on the plan "Transfer deposit", assuming that the goal "Maintain production stock" is compulsory and free:

$$\langle\langle Customer, \{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6\}, \emptyset, \emptyset\rangle,$$
$$\langle CustomerSupervisor, \emptyset, \{\varsigma_1^-, \varsigma_2^-, \varsigma_3^-, \varsigma_4^-, \varsigma_5^-, \varsigma_6^-\}, \emptyset\rangle\rangle$$

where:

$$\omega_1 = \langle \mathsf{O}_{a:Customer} input(sourceAcctNum), none, t_d, t_a, t_e\rangle$$
$$\omega_2 = \langle \mathsf{O}_{a:Customer} input(destAcctNum), none, t_d, t_a, t_e\rangle$$
$$\omega_3 = \langle \mathsf{O}_{a:Customer} input(minDepAmt), none, t_d, t_a, t_e\rangle$$
$$\omega_4 = \langle \mathsf{O}_{a:Customer} output(destAccountBalance), none, t_d, t_a, t_e\rangle$$
$$\omega_5 = \langle \mathsf{O}_{a:Customer} higherBalance(minDepAmt, sourceAcctNum), inOutConstraint(in), t_d, t_a, t_e\rangle$$
$$\omega_6 = \langle \mathsf{O}_{a:Customer} higherBalance(minDepAmt, destAcctNum), inOutConstraint(out), t_d, t_a, t_e\rangle$$

$$\varsigma_1^- = \langle \omega_1, \neg input(sourceAcctNum), \mathcal{A}^{\varsigma_1^-}_{a:CustomerSupervisor}\rangle$$
$$\varsigma_2^- = \langle \omega_2, \neg input(destAcctNum), \mathcal{A}^{\varsigma_2^-}_{a:CustomerSupervisor}\rangle$$
$$\varsigma_3^- = \langle \omega_3, \neg input(minDepAmt), \mathcal{A}^{\varsigma_3^-}_{a:CustomerSupervisor}\rangle$$
$$\varsigma_4^- = \langle \omega_4, \neg output(destAccountBalance), \mathcal{A}^{\varsigma_4^-}_{a:CustomerSupervisor}\rangle$$
$$\varsigma_5^- = \langle \omega_5, \neg higherBalance(minDepAmt, sourceAcctNum), \mathcal{A}^{\varsigma_5^-}_{a:CustomerSupervisor}\rangle$$
$$\varsigma_6^- = \langle \omega_6, \neg higherBalance(minDepAmt, destAcctNum), \mathcal{A}^{\varsigma_6^-}_{a:CustomerSupervisor}\rangle$$

Above, the *CustomerSupervisor* is the role that acts as the authority for the *Customer* role. The plan gives us six obligations for the *Customer* role and the corresponding sanctions enforced by the *CustomerSupervisor* role. $\mathcal{A}^{\varsigma_1^-}_{a:CustomerSupervisor}$ to $\mathcal{A}^{\varsigma_6^-}_{a:CustomerSupervisor}$ are sanctioning actions defined by the engineer or selected from a repository.

Contracts from Domain Constraints

Domain constraints extracted from domain assumptions define conditions in the environment that cannot be violated. Any domain constraint is therefore transformed into prohibitions. The transformation is straightforward. A domain constraint DC that has not yet been transformed is taken from the ES. In the first part of the algorithm (lines 1–10), we generate prohibitions and sanctions for all atomic first-order formulas $a_i, 1 \leq i \leq n$ that we extract from the Horn clauses in the DC.Uphold attribute (i.e., for a clause $a_0 \leftarrow a_1, a_2$ we keep only a_1 and a_2 and count them as separate since we need atomic first-order formulas). In case DC is conditional, we also define constraints on the prohibitions. In the second part of the algorithm (lines 11–13), we add the prohibitions to all (including authority) roles in all contracts. We then add all sanctions to all authority roles in all contracts. This is necessary in order enforce that the domain constraints are not violated throughout the MAS.

Contract from Domain Constraint (RdC-CDC)	
Input:	One domain constraint DC from the ES, such that DC has not been transformed previously.
Output:	Prohibitions and corresponding sanctions added to all contracts in the MAS to ensure that the logical conditions given in DC.Uhold are not violated.
begin	
1	**For each** $a_i, 1 \leq i \leq n$ in DC.Uphold **do**
2	Create prohibition $\omega_i = \langle \mathsf{F}_{a:r}\phi \wedge \bigwedge_{j=1}^{m} \kappa_j, t_d, t_a, t_e \rangle$ where $\phi = a_i$ and t_d, t_a, t_e are recorded automatically.
3	**If** DC.Type is Conditional **then**
4	**For each** $\kappa_j, 1 \leq j \leq m$, $\kappa_j = b_j$
5	**End If.**
6	**If** DC.Type is Free **then**
7	**For each** $\kappa_j, 1 \leq j \leq m$, **do** κ_j is empty and b_j is empty.
8	**End If.**
9	Create sanction $\varsigma_i^- = \langle \omega_i, \kappa_i, \mathcal{A}_{a:r} \rangle$ where $\kappa_i = \neg a_i$ and the engineer is asked to define $\mathcal{A}_{a:r}$.
10	Add $\{\omega_i | 1 \leq i \leq n\}$ to all roles in all contracts.
11	Add $\{\varsigma_i | 1 \leq i \leq n\}$ to all authority roles in all contracts.
end	

We create a prohibition (and a corresponding sanction) for each atomic first-order formula that appears in the Horn clauses in the domain constraint's Uphold attribute. We thereby have prohibitions and sanctions for all logical conditions given in the domain constraint. Since we expect the domain constraints to be respected by all roles involved in the MAS, we distribute the prohibitions to all supervised roles in the MAS, and the corresponding sanctions to all supervisor roles in the MAS. The algorithm is thus correct. The algorithm always terminates as every **for each** loop goes through a finite set and considers one element at a time.

If we apply the above algorithm to the domain constraint "Maintain credit deposit", we obtain the following prohibition and sanction:

$$\begin{aligned} \omega = \langle \mathsf{F}_{a:r}(accountNumberBalance \leq minDepAmt), \\ creditAccount(accountNumber), t_d, t_a, t_e \rangle \\ \varsigma^- = \langle \omega, (accountNumberBalance \leq minDepAmt), \\ \mathcal{A}_{a:r}^{\varsigma^-} \rangle \end{aligned}$$

Contracts and Preferences

Preferences establish an order over requirements, and therefore over goals in the ES. Transformation of a functional goal returns a contract, whereas the transformation of a nonfunctional goal modifies already defined contracts. In both cases, the preference order defined over goals corresponds to a preference order over contracts. We do not, however, carry over the very notion of preference order onto the governance level. We instead use an approach that does not involve extending the conceptualizations at the governance level. Overall, we know that a preference for a functional goal A over B means that honoring the contract on A is more desirable than honoring the contract on B. In absence of preference orders to establish

relative desirability at the governance level, we must rely on constraints in norms. Namely, we can place constraints on norms of contract B so that those norms are activated (i.e., agents go about honoring the contract on B) only if the contract on A cannot be honored. Consequently, we can constrain the activation of norms in B to cases when norms in A cannot be honored. There is, however, no absolute criterion for knowing when a contract cannot be honored. We therefore leave it to the engineer to choose a set of *critical* conditions that, once met, mean that the contract on A cannot be honored, and that the contract on B is to be activated. Following intuition, these conditions are the execution of sanctions and/or the failure to provide incentives given in the contract on A. Indeed, sanctions are executed only if violations occur, while incentives are not provided if permissions are not exploited—if this occurs, we can reasonably expect that the contract on A will not be honored. Note that if we set conditions based on *all* sanctions/incentives in contract A, we will let the MAS violate some of them, then continue trying to honor A. As this may, e.g., take too much time, we can limit the relevant conditions only to those related to part of the sanctions/incentives in contract A. In summary then, if we have a preference order $A >> B >> C$, then if the contract obtained on A is not honored (i.e., sanctions of that contract are applied or incentive not provided), we activate the contract on B, and if the contract on B is not honored, we activate the contract on C. To activate the contract on B, we introduce additional constraints to those already defined within the norms in B: if the additional constraints hold, the contract on A is not honored.

The procedures for preferences on functional and for those on nonfunctional goals differ somewhat, but both rely on the same principle just described above. We first consider preferences on functional, then those on nonfunctional goals.

Preferences over Contracts on Functional Goals (RdC-CPFG)

`Input:` One preference order order among functional goals, from ES, that has not been transformed previously. Assume for simplicity that the order involves w goals, and is of the form $\mathsf{G}_1.\mathsf{UID} >> \mathsf{G}_2.\mathsf{UID} >> \mathsf{G}_3.\mathsf{UID} >> \ldots >> \mathsf{G}_w.\mathsf{UID}$, and that a "preference pair" from that order is $\mathsf{G}_k.\mathsf{UID} >> \mathsf{G}_{k+1}.\mathsf{UID}$, for $1 \leq k \leq w-1$.

`Output:` Contract on each less preferred goal in a preference pair, updated with constraints ensuring that the norms in the contract activate only if the contract on the more preferred goal cannot be honored.

```
begin
1   For each preference pair G_k.UID >> G_{k+1}.UID, 1 ≤ k ≤ w − 1 do
2       Choose a set of critical sanctions/incentives in the contract on G_k;
3       For each critical sanction/incentive ς_i^((·)) = ⟨(·), φ_i, (·)⟩ do
4           For each norm ⟨(·), ⋀_{j=1}^m κ_j, (·), (·), (·)⟩ in the contract on G_{k+1} do
5               If ς_i^((·)) = ⟨(·), φ_i, (·)⟩ is a sanction then
6                   Replace (⋀_{j=1}^m κ_j) by (κ_{m+i} ∧ ⋀_{j=1}^m κ_j) where κ_{m+i} = φ_i.
7               End If.
8               If ς_i^((·)) = ⟨(·), φ_i, (·)⟩ is an incentive then
9                   Replace (⋀_{j=1}^m κ_j) by (κ_{m+i} ∧ ⋀_{j=1}^m κ_j) where κ_{m+i} = ¬φ_i
10              End If.
end
```

The algorithm considers each preference pair in a preference order. Given a pair of functional goals, we take the more preferred functional goal, and consider individually each of the critical sanctions or incentives appearing in the contract on that goal. The engineer is asked to select, among all sanctions/incentives in the contract on the more preferred goal those that are *critical*. A sanction/incentive is critical if its activation is (for the engineer) a sufficient condition to abandon the contract on the more preferred goal, and move to realize the contract on the less preferred goal in the preference pair. For each critical sanction/incentive, we add the sanction/incentive's activation condition ϕ to each of the norms appearing in the contract on the less preferred goal. By doing so, we ensure that the contract on the less preferred goal will only be considered if all critical sanctions are activated and/or no critical incentives are provided. Notice that if we consider all sanctions/incentives as critical, we will only fall back to the contract on the less preferred goal if all sanctions are violated and none of the incentives is provided. This is a limiting case, for it may lead the MAS rarely to fall back to less preferred goals. Indeed, the MAS is in that case sensitive only to full violations, and so is persistent in satisfying to the most desirable extent the preferences, which may not be the most desirable option. We therefore have the engineer choose critical sanctions/incentives.

The approach is slightly different for nonfunctional goals. Let $AN >> BN >> CN$ be a preference order between three nonfunctional goals. We know that nonfunctional goals transform into norms,

sanctions, and incentives added to already available contracts, themselves obtained from functional goals referenced in each nonfunctional goal's FunctionalSource attribute.

Preferences over Contracts on Nonfunctional Goals (RdC-CPNFG)

`Input:` One preference order (from ES) among nonfunctional goals that has not been transformed previously. Assume for simplicity that the order involves w nonfunctional goals, and is of the form $\mathsf{NG}_1.\mathsf{UID} >> \mathsf{NG}_2.\mathsf{UID} >> \mathsf{NG}_3.\mathsf{UID} >> \ldots >> \mathsf{NG}_w.\mathsf{UID}$, and that a "preference pair" from that order is $\mathsf{NG}_k.\mathsf{UID} >> \mathsf{NG}_{k+1}.\mathsf{UID}$, for $1 \leq k \leq w$.

`Output:` Updated contract on each functional goal listed in the less preferred nonfunctional goal's FunctionalSource attribute. The update consists of constraints ensuring that the norms in the contract activate only if the contracts on the functional goals listed in the more preferred nonfunctional goal's FunctionalSource cannot be honored.

`begin`

`1` **For each** preference pair $\mathsf{NG}_k.\mathsf{UID} >> \mathsf{NG}_{k+1}.\mathsf{UID}$, $1 \leq k \leq w-1$ **do**

`2` Choose a set of *critical* sanctions/incentives among sanctions/incentives in contracts of all functional goals listed in NG_k.FunctionalSource. Sanctions/incentives chosen as critical must be those obtained by transformation (through the RdC-CPFG algorithm described earlier §13.4.3) of NG_k.

`3` **For each** critical sanction/incentive $\varsigma_i^{((\cdot))} = \langle (\cdot), \phi_i, (\cdot) \rangle$ **do**

`4` **For each** functional goal G listed in NG_{k+1}.FunctionalSource **do**

`5` **For each** norm $\langle (\cdot), \bigwedge_{j=1}^{m} \kappa_j, (\cdot), (\cdot), (\cdot) \rangle$ in the contract on a functional goal G **do**

`6` **If** $\varsigma_i^{((\cdot))} = \langle (\cdot), \phi_i, (\cdot) \rangle$ is a sanction **then**

`7` Replace $(\bigwedge_{j=1}^{m} \kappa_j)$ by $(\kappa_{m+i} \wedge \bigwedge_{j=1}^{m} \kappa_j)$ where $\kappa_{m+i} = \phi_i$.

`8` **End If.**

`9` **If** $\varsigma_i^{((\cdot))} = \langle (\cdot), \phi_i, (\cdot) \rangle$ is an incentive **then**

`10` Replace $(\bigwedge_{j=1}^{m} \kappa_j)$ by $(\kappa_{m+i} \wedge \bigwedge_{j=1}^{m} \kappa_j)$ where $\kappa_{m+i} = \neg\phi_i$

`11` **End If.**

`end`

The algorithm proceeds similarly to RdC-CPFG; the differences result from the fact that the transformation of a nonfunctional goal does not give a standalone contract, but adds norms, sanctions, and/or incentives to contracts defined for all functional goals listed in the nonfunctional goal's FunctionalSource attribute. RdC-CPNFG considers each preference pair in the preference order. Given a pair of nonfunctional goals, we take the preferred nonfunctional goal, and let the engineer choose critical sanctions/incentives in contracts of all functional goals listed in the preferred nonfunctional goal's FunctionalSource attribute. Criticality herein is defined in the same way as in RdC-CPFG. We add each sanction/incentive's activation condition ϕ to each of the norms appearing in each of the contracts defined for functional goals listed in the less preferred nonfunctional goal's FunctionalSource attribute. We do so to ensure that the contracts of all of the functional goals listed in the less preferred nonfunctional goal's FunctionalSource attribute, are activated if the critical sanctions are applied and none of the critical incentives is provided. In other words, we ensure that, if the preferred nonfunctional goal is not satisfied, we activate the contracts intended to satisfy (through functional goals) the less preferred nonfunctional goal.

Both RdC-CPFG and RdC-CPNFG are correct: they ensure that the contracts on less preferred options in the given preference orders are activated only when contracts on more preferred options cannot be honored. Both algorithms always terminate as all of their **for each** loops move through finite sets, and always process one element at a time.

Contracts and Priorities

Recall that a priority order establishes relative importance between two or more preferences. The preferences are of the same kind, that is, we cannot have a priority order between a preference over functional goals and a preference over nonfunctional goals. Let $Pf_A >>> Pf_B$ be a priority between Pf_A and Pf_B, where $Pf_A = A_1 >> A_2 >> A_3 >> A_4 >> A_5$ and $Pf_B = B_1 >> B_2 >> B_3$. For simplicity, let all $A_i, 1 \leq i \leq 5$ and $B_j, 1 \leq j \leq 3$ be functional goals. As we do not intend to carry over the notion of priority to the governance level, we need to determine the conditions which activate the honoring of contracts on B_j, and this depending on whether the contracts on A_i are honored. Because $Pf_A >>> Pf_B$,

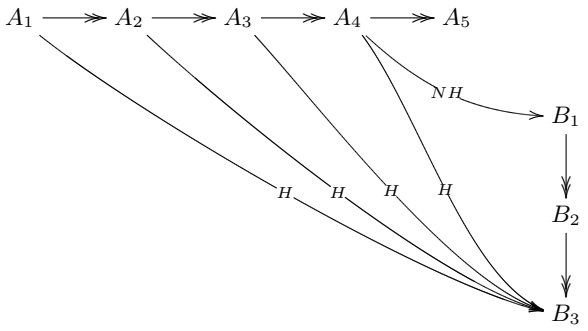


Fig. 24. Prioritizing choices.

we know that it is more important to honor contracts on A_i and this starting from the most preferred A_1, than to honor preferred contracts on B_j.

To understand our approach to translating priorities to the governance level, consider the diagram in Figure 24, where we relate the mentioned functional goals with arrows of two kinds. Arrows of the form $A_i \longrightarrow\!\!\!\!\!\rightarrow A_{i+1}$ denote preference of A_i over A_{i+1}. An arrow labeled H from A_i to B_j means that if the contract on A_i is honored, the contract on B_j should be honored. An arrow labeled NH from A_i to B_j means that if the contract on A_i is *not* honored, the contract on B_j should be honored. Because satisfying the preference Pf_A means honoring the contract on its most preferred, yet feasible functional goal, honoring the contract on A_1 means that we cannot honor the contract on the most preferred feasible functional goal in Pf_B—this is due to the priority order, which indicates that we cannot achieve most desirable goals in *both* Pf_A and Pf_B. Consequently, if we honor the contract on A_1, the safest choice is to attempt to honor the contract on the least preferred goal in Pf_B. If the MAS fails to honor the contract on A_1, but instead honors the contract on A_2, we still safely choose to attempt to honor the contract on B_3. This rationale applies until A_4. If the MAS fails to honor A_4, we know that the preference Pf_A is not satisfied (i.e., we ended up with the least preferred choice), so that we can instead try to honor the most preferred choice in the less important preference order—that is, we try to honor B_1. Note that there is no arrow coming out of A_5 because it is at present irrelevant whether the contract on A_5 is honored or not: as soon as we know that the contract on A_4 is not honored, we know that the most preferred choice on the less important priority scale should be activated (i.e., B_1). This is a simple and safe approach; another may be to relate some of the inner A_is with B_2, but it would require that we can quantify the difference in perceived utility among As and Bs, and therefore compare them. We do not consider this option here; it is, in any case not applicable to all priority orders.

The described approach ensures that we force the MAS to try satisfying the more important preference order, and if it fails, we still expect the MAS to try to satisfy as much as possible the less important preference order. Once we know the order in which to try to honor the contracts, we can condition the norms in the way that the given order is respected (we do so in a similar way as we did for preferences, in RdC-CPFG and RdC-CPNFG).

To enforce, in the MAS, the behavior described in Figure 24, we need to introduce additional constraints only in the norms in the contract on B_1. These additional constraints will activate the norms in the contract on B_2 only if the critical[6] conditions, as chosen by the engineer, hold (and therefore indicate that the contract on A_4 cannot be honored.

Consider a more general case. Let $Pf_1 >>> Pf_2 >>> \ldots >>> Pf_h$ be a priority order between h preference orders. Let each $P_x, 1 \leq x \leq h$ be of the form $P_x = G_{x,1} >> \ldots >> G_{x,w_x}$ with $w_x \geq 2$ ordered functional goals. To obtain the desired behavior, we must introduce constraints in all norms in the contract on $G_{x+1,1}$, where $1 \leq x \leq h-1$. The constraints we introduce are obtained from critical sanctions/incentives of the goal G_{x,w_x-1}: once these critical sanctions are applied or incentives not provided, we abandon trying to honor G_{x,w_x-1} and move to try to honor the most preferred goal in the next less important preference order (i.e., $G_{x+1,1}$). If we deal with priorities over preferences, themselves over *nonfunctional* goals, we must introduce constraints in all norms in the contracts on all functional goals mentioned in the nonfunctional goal $NG_{x+1,1}$'s attribute FunctionalSource. Constraints are obtained from the sanctions/incentives of the nonfunctional goal NG_{x,w_x-1}.

[6] Critical in the same sense as in "critical conditions" introduced earlier (§13.4.3).

The algorithm behaves in the manner we described in Figure 24. For a chosen preference order (line 2), we check if its elements are functional or nonfunctional goals. If functional (lines 3–13), we need to choose a set of sanctions and/or incentives whose, respectively, execution and/or non-provision is sufficient to indicate that the contract on the functional goal R_{x,w_x-1} (i.e., A_4 in Figure 24) cannot be honored. We then take each critical sanction/incentive (lines 5–12) and add its activation condition ϕ_i to constraints on all norms in the contract on $R_{x+1,1}$ (i.e., B_1 in Figure 24); by doing so, we ensure norms in the contract on $R_{x+1,1}$ activate only if the contract on R_{x,w_x-1} is not honored. There are two differences if $R_{x+1,1}$ is a nonfunctional goal (lines 14–25): (i) we choose critical sanctions/incentives from all functional goals listed in R_{x,w_x-1}.FunctionalSource, and not in a single functional goal (line 15); (ii) we add the critical sanctions/incentives' activation conditions ϕ_i to constraints on all norms in all the contracts on functional goals listed in in $R_{x+1,1}$.FunctionalSource (line 17). The algorithm is correct, in that it adjusts contracts in the MAS to the behavior we described in Figure 24: norms in the contract on each most preferred goal in next less important preference order (i.e., $R_{x+1,1}$) are modified so as that they activate only if we fail to honor the contract on R_{x,w_x-1}. The algorithm always terminates: the lenght of the priority order, the lenghts w_x of the w preference orders Pf_x, and the numbers of norms, sanctions, and incentives are finite and usually small; if Rs are nonfunctional goals, we also need to account for the numbers of functional goals listed in the FunctionalSource attributes of the considered nonfunctional goals—again, these are finite and usually small.

	Priorities over Preferences over Contracts on Goals (RdC-CPP)
`Input:`	One priority order (from ES) that has not been transformed previously. Assume for simplicity that the chosen priority order is of the form $Pf_1 >>> Pf_2 >>> \ldots >>> Pf_h$, between h preference orders. Let each $Pf_x, 1 \leq x \leq h$ be of the form $Pf_x = R_{x,1} >> \ldots >> R_{x,w_x}$ with $w_x \geq 2$, where all R are either functional, or nonfunctional goals.
`Output:`	Updated contract on $R_{x+1,1}$. If $R_{x+1,1}$ is a functional goal, the update consists of constraints ensuring that the norms in the contract activate only if the contract on R_{x,w_x-1} cannot be honored. If $R_{x+1,1}$ is a nonfunctional goal, the update consists of constraints ensuring that the norms of the contracts on functional goals listed in $R_{x+1,1}$.FunctionalSource activate only if the contracts on nonfunctional goals listed in R_{x,w_x-1}.FunctionalSource cannot be honored.
`begin`	
1	**For each** $Pf_{x+1}, 1 \leq x \leq h-1$ **do**
2	**If** $R_{x+1,1}$ is a functional goal **then**
3	Choose a set of *critical* sanctions/incentives in the contract on R_{x,w_x-1};
4	**For each** critical sanction/incentive $\varsigma_i^{((\cdot))} = \langle(\cdot), \phi_i, (\cdot)\rangle$ **do**
5	**For each** norm $\langle(\cdot), \bigwedge_{j=1}^{m} \kappa_j, (\cdot), (\cdot), (\cdot)\rangle$ in the contract on $R_{x+1,1}$ **do**
6	**If** $\varsigma_i^{((\cdot))} = \langle(\cdot), \phi_i, (\cdot)\rangle$ is a sanction **then**
7	Replace $(\bigwedge_{j=1}^{m} \kappa_j)$ by $(\kappa_{m+i} \wedge \bigwedge_{j=1}^{m} \kappa_j)$ where $\kappa_{m+i} = \phi_i$.
8	**End If.**
9	**If** $\varsigma_i^{((\cdot))} = \langle(\cdot), \phi_i, (\cdot)\rangle$ is an incentive **then**
10	Replace $(\bigwedge_{j=1}^{m} \kappa_j)$ by $(\kappa_{m+i} \wedge \bigwedge_{j=1}^{m} \kappa_j)$ where $\kappa_{m+i} = \neg\phi_i$
11	**End If.**
12	**End If.**
13	**If** $R_{x+1,1}$ is a nonfunctional goal **then**
14	Choose a set of *critical* sanctions/incentives among sanctions/incentives in contracts of all functional goals listed in R_{x,w_x-1}.FunctionalSource. Sanctions/incentives chosen as critical must be those obtained by transformation (through the RdC-CPFG algorithm described earlier §13.4.3) of R_{x,w_x-1}.
15	**For each** critical sanction/incentive $\varsigma_i^{((\cdot))} = \langle(\cdot), \phi_i, (\cdot)\rangle$ **do**
16	**For each** functional goal listed in $R_{x+1,1}$.FunctionalSource **do**
17	**For each** norm $\langle(\cdot), \bigwedge_{j=1}^{m} \kappa_j, (\cdot), (\cdot), (\cdot)\rangle$ in the contract for the functional goal **do**

```
18                          If ς_i^((·)) = ⟨(·), φ_i, (·)⟩ is a sanction then
19                              Replace (⋀_{j=1}^{m} κ_j) by (κ_{m+i} ∧ ⋀_{j=1}^{m} κ_j) where κ_{m+i} = φ_i.
20                          End If.
21                          If ς_i^((·)) = ⟨(·), φ_i, (·)⟩ is an incentive then
22                              Replace (⋀_{j=1}^{m} κ_j) by (κ_{m+i} ∧ ⋀_{j=1}^{m} κ_j) where κ_{m+i} = ¬φ_i
23                          End If.
24              End If.
end
```

13.5 Enabling VO through Open and Norm-Governed MAS

We now describe how our norm-regulated VOs give rise to norm-aware agent societies. We address open and heterogeneous MAS: we accommodate external agents by providing each of them with a corresponding *governor agent* [102]. This is a kind of "chaperon" that interacts with an external agent, and observes and reports on its behaviour. The architecture is shown in Figure 25.

A number of external agents interact (denoted by the "$\rightleftarrows$") with their corresponding governor agents. The governor agents have access to the VO description $\mathcal{I}$, the current state s of the VO enactment, the global execution state Ξ and the global normative state Ω. Governor agents are able to write to and read from (denoted by the "$\Longleftrightarrow$") a shared memory space (*e.g.*, a blackboard-like solution implemented as a tuple space), updating the global configuration (denoted by the "$\rightsquigarrow$") to reflect the dynamics of the VO enactment. As we have noted earlier, governor agents are necessary because we cannot anticipate or legislate over the design or behaviour of external agents. We depict below how the pairs of governor/external agents work together: any non-deterministic choices on the VO are decided by the external agent; any normative aspects are considered by the governor agent.

The governor agent represents the external agent within the VO. As such, it has the unique identifier of the external agent. The governor agent keeps an account of all roles the external agent is currently playing: in our VOs, it is possible for agents to take up more than one role simultaneously. We define in Figure 26 how governor agents work—we use a logic program for this purpose. The first clause (lines 1–3) depicts the termination condition: `get_tuple`/1 (line 2) retrieves $\langle \mathcal{I}, s, \Xi, \Omega \rangle$ from the shared tuple space and `terminate`/4 checks if the current VO enactment (recorded in Ξ) has come to an end. The team of governor agents synchronise their access to the tuple space [102], thus ensuring each has a chance to function.

The second clause (lines 4-9) depicts a generic loop when the termination condition of the first clause does not hold. In this case, the tuple is again retrieved (line 5) and the governor agent proceeds (line 6) to analyse the current global normative state Ω with a view to obtaining the subset $\Omega_{Id} \subseteq \Omega$ of norms referring to agent *Id* under roles *Roles*. Predicate `filter_norms`/4 collects the norms which apply to agent *Id* (the governor agent's external agent). In line 7 the governor agent, while in possession of the applicable norms as well as other relevant information, interacts with the external agent to decide on a set of *Actions* which are norm-compliant—these actions will be used to update (line 8) the global execution state Ξ. In the process of updating the state of execution, a new set of roles must be assigned to the external agent, represented as *NewRoles*. The governor agent keeps looping (line 9) using the identifier for the external agent and its new set of roles.

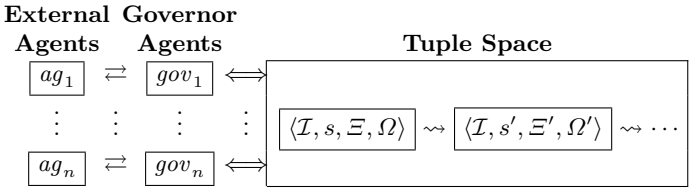


Fig. 25. Architecture for Norm-Aware Agent Societies

```
1 main(Id, Roles) ←
2    get_tuple(⟨I, s, Ξ, Ω⟩)∧
3    terminate(Id, Roles, I, Ξ, Ω)
4 main(Id, Roles) ←
5    get_tuple(⟨I, s, Ξ, Ω⟩)∧
6    filter_norms(Id, Roles, Ω, Ω_Id)∧
7    discuss_norms(Id, Roles, I, s, Ξ, Ω_Id, Actions)∧
8    update_tuple(Roles, Actions, NewRoles)∧
9    main(Id, NewRoles)
```

Fig. 26. Governor Agent as a Logic Program

13.6 Related Work

The primary purpose of RdC is to ensure that norms, sanctions, and incentives govern the actions and interactions within an open and norm-regulated MAS to the aim of satisfying stakeholders' requirements, and this to the most desirable feasible extent. RdC therefore integrates:

- concepts that guide the collection and organization of requirements and associated information expressed in natural language;
- concepts used to specify in a structured and formal way the requirements and associated considerations information within an environment specification;
- concepts used to write norms, sanctions, and incentives needed to govern an open and norm-governed MAS, that is a virtual organization;
- techniques, in the form of algorithms, used to write norms, sanctions, and incentives automatically from the environment specification, thus allowing the definition of VO governance using RE concepts.

Related efforts include frameworks for modeling and developing MAS, for engineering VO, and for defining norms in MAS. We discuss prominent results in each of these research directions below.

13.6.1 Frameworks for Modeling and Developing MAS

OMNI.

The Organizational Model for Normative Institutions (OMNI) modeling framework for MAS [322] involves three dimensions: the normative dimension, specifying norms and rules to which members are expected to adhere; the organizational dimension, describing what roles are involved in the MAS, and how they interact; and the ontological dimension, defining relevant concepts of the application domain so as to facilitate the exchange of information among agents occupying the roles. Requirements in OMNI take the form of "statutes" of the organization, that is, objectives, values, and the context (i.e., application domain) in which the organization operates. Objectives are associated to roles, along with rights, norms, and rules. OMNI provides three general ways for organizing the interaction between roles, namely the market (coordination by price), network (by trust), and hierarchy (by authority). Norms in OMNI cover obligations, permissions, and prohibitions, and are converted into rules, whose satisfaction can be verified. Violations lead to sanctions.
Discussion. The requirements level in OMNI is limited compared to that in RdC: (i) clear definitions of objectives and values are not available; (ii) preferences and priorities cannot be ported onto the governance level, and it is unclear how they are treated during MAS design; (iii) objectives are functional, so that it remains unknown how objectives relate to nonfunctional requirements. The logical formalism used for specifications is that of temporal, relativized, and conditional deontic logic, whereby the specification is translated into a propositional dynamic logic at the stage when rules are written from norms and sanctions. RdC does not propose how roles are to be organized, focusing instead on binding pairs or small groups of roles through contracts. Roles are therefore meaningfully grouped according to the goals, and therefore requirements. The structure is therefore driven by requirements, and not by an *a priori* structural form. There is, however, no difficulty in introducing, at the requirements level, the requirements *about* the structural form, which by definition of RdC algorithms give rise to norms, sanctions, and incentives, through which the structural choice will be enforced. OMNI descends to the inter-agent communication level which is not considered in RdC.

Gaia.

The Gaia methodology [337] for the analysis and design of MAS starts with the identification of goals that the MAS is to satisfy. A model is then constructed of the environment in which the future MAS will operate. The environmental model describes resources available to the MAS and the actions that can be performed and that use the resources. The engineer then constructs a preliminary roles model, in which each role is defined as a collections of protocoles and activities that the agent occupying the role can perform, permissions assigned through the role to the agent, and liveness and safety responsibilities. A model of interactions between roles is then built. The approach stops when the rules governing the overall operation of the MAS organization are defined. These indicate for instance, that some protocols can be executed only a given number of times, the sequence of protocols, and so on. Gaia does not commit to a particular formalism, leaving it to the engineer to choose a formalism relevant given the application domain at hand.
Discussion. Gaia does not cover the requirements phase, and since it is rather open as to the form of governance, its combined use with RdC is a topic for future work. This combination may be interesting as Gaia covers for instance the interaction considerations which are not considered in RdC. Moreover, Gaia suggests a linear approach, in which it is unclear how revisions of requirements cascade through the various lower-level models. RdC addresses such issues through automated propagation of changes at the requirements level and environment specification level to the governance level. One of the mismatches between RdC and Gaia, however, is Gaia's writing of norms through rules. Governance-level notions in RdC are more elaborate, covering various kinds of norms, along with sanctions and incentives.

SODA.

The Societies in Open and Distributed Agent spaces [238] is a MAS design methodology, which involves analysis and design phases. In the former, role, resource, and interaction models are produced. The role model defines tasks that need to be performed, and tasks are allocated to roles, while roles are placed into groups. The resource model describes the MAS environment in terms of services (which exploit resources) that the environment can provide to the agents. The interaction model shows roles, groups, and resources interacting through protocols involving exchanges of information under interaction rules. For instance, a role is defined in terms of tasks, permissions to use resources, and an interaction protocol in which it takes part. At the design level, roles are mapped to agent classes, groups to societies of agents (defined by permissions, roles, and interaction rules), and resources to infrastructure classes (characterized by access modes, permissions, and allowed interaction protocols).
Discussion. SODA does not address the requirements level. It produces rigid agent societies for it maps roles to agent classes at design time. Normative notions other than access permissions are missing. Role of a domain ontology in SODA is unclear.

ISLANDER.

The ISLANDER approach [82] considers any agent-based organization as a setting in which message exchanges play a critical role, in the sense that they enable interaction. Several agents involved in interaction will participate in a scene, whereby a scene follows a precisely defined protocol. Primitive scenes are considered to be modules, and are composed into more elaborate scenes to guide complex interactions. In such a context, a role defines the scences to which an agent can take part, and the protocols it should follow. ISLANDER requires a domain ontology, and involves ambient calculus for specification.
Discussion. ISLANDER does not cover the requirements level and lacks normative notions. In this respect, it differs strongly from RdC and frameworks discussed at present. It does, however, provide a modular approach to the definition of interactions, and is in this respect relevant when considering the design of intra-role interactions.

Tropos.

The Tropos methodology [50, 100] spans the requirements, design, and implementation phases of MAS development. It features rich requirements models, grounded in the i^* modeling framework [331, 332]. Functional and nonfunctional requirements and domain assumptions are covered, along with roles and patterns of organized interaction [182]. Dependency models are used at the early requirements phase for preliminary structuring and documentation of natural language requirements. These models allow the engineer to study the impact that the MAS is likely to have within the environment in which it will be deployed. Early requirements are formalized using a linear temporal first-order logic and model

checking can be performed on specifications of restricted size [100]. Late requirements phase invovles the definition of a MAS architecture in terms of roles and their interdependencies, whereby the architecture is chosen so as to best satisfy nonfunctional requirements. Detailed design involves the definition of data schemas, tasks, and interactions, using, e.g., the Unified Modeling Language. Implementation involves the definition of intelligent agents needed to populate the MAS.
Discussion. We have discussed at some lenght elsewhere [151, 152, 156] that requirements models, such as those of Tropos, need to be enriched with notions of preference and priority, along with a finer taxonomy of requirements, and corresponding goals. We have argued above (§13.2.1) for the relevance of preferences in open MAS, in particular for avoiding idealistic and/or pessimistic requirements. Priorities become necessary as soon as preferences are used, to resolve conflicts between preferences. We do use informal models derived from the *i** framework as well, although precise specification in RdC relies on a less expressive though computationally attractive formalism. The role of a domain ontology is implicit in Tropos, while it is explicit in RdC, again due to the focus on open MAS. Tropos does not consider that MAS are governed, but instead assumes that appropriate agents are designed or otherwise found to properly occupy roles. Governance of open MAS is therefore not a concern in Tropos. RdC automates some activities that are manual in Tropos: given already designed agents, RdC governs their behavior through the conversion of requirements to governance notions, so that RdC skips detailed by exploiting the openness of norm-governed MAS.

13.6.2 Frameworks for Engineering VO

CONOISE-G.

CONOISE-G [250, 229] is a framework incorporating agent-based models and techniques needed for the automated formation and maintenance of virtual organizations. The framework covers the entire lifecycle of VO, including formation, operation, perturbation, and dissolution phases. At formation, a requirements agent represents the user, acquires user's requirements, then initiates VO formation by distributing a call for bids for the specific requirements. Once the deadline for the proposals passes, the requirements agent selects a combination of agents best capable to fulfil the requirements. After formation, the agents act and interact in order to fulfil the requirements, wherbey their actions are monitored to ensure expected quality of service levels. Perturbation occurs when better agents become available or because the current members do not perform as expected and agreed. Agents are replaced accordingly. The VO dissolves once the requirements have been fulfilled, and agents are no longer needed.
Discussion. CONOISE-G does not focus on the requirements level, and is more concerned about the finding of optimal groups of agents to fulfil functional user requests. Nonfunctional considerations are catered through QoS constraints enforced by the monitoring agents. It is unclear whether the VO is governed or managed otherwise. While functional and nonfunctional constraints can be defined, the treatment of preferences and priorities seems to be missing. Transformation of requirements onto lower-level notions is also not discussed, seemingly indicating that requirements are already specified and that their format is directly approproiate for managing agents selected by the requirements agent.

ADEPT.

The Advanced Decision Environment for Process Tasks [145] views a business process as a collection of autonomous agents that interact in order to reach mutually acceptable agreements that coordinate their interdependent subactivities. Agents manage services, that is, conceptual units of problem-solving activity in the business process. ADEPT simplifies the design of business processes by automating the allocation, scheduling, and execution decisions that are specified at design time in non-agent-based systems. Each service is described with a unique ID, and the specification of inputs, outputs, guard, and body. Service level agreements specify contractual terms regulating interactions between agents.
Discussion. ADEPT starts off from already defined contracts—the relationships between the requirements that resulted in the contracts and the contract provisions are not studied. Governance notions are limited, being restricted to obligations and sanctions.

13.6.3 Frameworks for Defining Norms in MAS

OCA.

The role-based model for Organized Collective Agency [244] involves the use of a deontic and action modal logic to capture the concept of acting in a role and relating it to obligations, permissions, and prohibitions.

An organization in OCA consists of roles, deontic characterizations of roles (i.e., obligations, permissions, and prohibitions intrinsic to the role), transmissions of obligations (describing how roles are assigned to agents—this amounts to stating that whenever an agent has some specific obligation, it necessarily holds the role to which the obligation is assigned), and relationships between roles (sub-role, implication to ensure that if some role is held then another one is held as well, and incompatibility of roles so that they cannot be assigned simultaneously to the same agent). Interactions between agents are governed by contracts. A contract identifies the agents involved, the roles they occupy, and the obliations, permissions, and/or prohibitions of the roles.
Discussion. The focus of OCA is on the use of a multi-sorted modal deontic and action first-order logic for the reasoning about roles and norms. The implications of its computational complexity are not considered. The focus is not on the requirements or specification level; the framework applies to norms only. Working without the higher levels (namely, requirements and environment specification) leads unavoidably to a restricted set of concepts that are considered relevant. In this sense, sanctions and incentives are lacking, and the goal dependencies that we would expect to see as a kind of relationship between roles is missing.

LGI.

"Law Governed Interaction is a mode of interaction that allows a heterogenous group of distributed agents to interact with each other, with confidence that an explicitly specified set of rules of engagement—called the law of the group—is complied with." [223] The law of the group is an explicit and enforced set of rules that are enforced among the agents participating in the group. The law defines roles of agents, and their associated obligations. Sanctions apply when obligations are not held. Laws are written in Prolog, and the the coordination mechanism on which laws can be defined is implemented in the Moses toolkit [222].
Discussion. Again, as for much of mentioned related efforts, the higher levels of abstraction, namely those of requirements and the corresponding specification, are not considered. Governance notions are restricted.

13.7 Discussion

RdC is limited in several respects that may not be directly apparent from the text. We discuss these limitations, and consequently provide indications on directions for future effort.

13.7.1 Managing Inconsistency

Work on managing inconsistencies in a requirements specification is available for RE frameworks relying on linear temporal first-order logic [317, 318]. We have not shown how inconsistencies are managed in the RdC's environment specification, so that it is unclear at present how precisely the engineer is to proceed, e.g., to detect and resolve conflicts between goals. We have argued that preferences can be conflicting and that priorities are then used to avoid conflicts; we have not, however presented an automated method for identifying conflicting preferences. Two remarks are relevant in this respect. First, it is possible to automate preference elicitation, and approaches to automated identification of conflicting preferences are available [39]. Second, automated resolution of conflicts between norms at our governance level is available [321]. First-order term unification [90] is employed to find out if and how norms overlap in their influence (*i.e.*, the agents and values of parameters in agents' actions that norms may affect). This allows for a fine-grained solution whereby the influence of conflicting or inconsistent norms is *curtailed* for particular sets of values. For instance, norms "agent x is permitted to $send\_bid(ag_1, 20)$" and "agent ag_2 is prohibited from doing $send\_bid(y, z)$" (where x, y, z are variables and $ag_1, ag_2, 20$ are constants) are in conflict because their agents, actions and terms (within the actions) unify. We solve the conflict by annotating norms with sets of values their variables *cannot* have, thus curtailing their influence. In our example, the conflict is avoided if we require that variable y cannot be ag_1 and that z cannot be 20. RdC can consequently address inconsistencies in an automated manner at the lower, governance level at present. More work is necessary for combining inconsistency detection and resolution at both the ES and governance levels.

13.7.2 Expressivity and Coverage

We have already noted that our choice of Horn clauses limits the kinds of information we can represent and reason about at the ES level (§13.3). Regarding the expressivity of the governance level, several improvements can be considered. We rely on supervised interaction to ensure that behavior of heterogenous agents is supervised, and sanctions applied and/or incentives provided. More elaborate structuring of roles may be needed than the decentralized model we have chosen. More rigid structures can be obtained by introducing additional role relationships, such as, e.g., OCA's implication and incompatibility. Additional normative concepts, such as that of power and loyalty, may be of interest. We can draw from economics (e.g., [292]) in conceptualizing loyalty as some degree of correspondence between agent's individual goals and those that it is brought to achieve within the organization. Calculating loyalty would facilitate the allocation of resources for supervision, so that, e.g., higher initial trust may be assigned to agents with higher loyalty values. We have not discussed the precise mechanisms needed to determine 'appropriate' levels of incentives or sanctions. Turning to how much of the MAS lifecycle RdC covers, we have noted earlier (§13.6) that RdC focuses only on generating norms, sanctions, and incentives from requirements. Norms generated in RdC are executable, relying on a toolset whose theoretical aspects are outlined elsewhere [178], and which is currently in its final development stages. RdC is not entirely standalone, as it does not deal with how the requirements are elicited. However, as RdC relies and extends common RE conceptualizations, it can reuse, in its current form, established processes for requirements elicitation (e.g., [50]). The scope of RdC stops at the governance level, so that, e.g., practical reasoning of agents when obeying to norms and the agent's actual integration within the VO are not discussed.

13.8 Conclusions

Open and norm-governed multi-agent systems appear to be one relevant approach to the engineering and management of virtual organizations. Norms, sanctions, and incentives are appropriate abstractions when establishing governance of the actions and interactions of heterogenous agents, designed, operated, and maintained by, and distributed across various providers. Any norm, sanction, and incentive is relevant in an open MAS only if it governs agent behavior in such ways as to ensure that requirements on the MAS are satisfied. It is therefore clear that the norms, sanctions, and incentives to define when engineering open and norm-governed MAS are determined by the requirements that the stakeholders place on the MAS. While this relationship between governance notions and requirements is intuitively clear, we have shown that limited effort has been invested in establishing the precise relationships between requirements engineering concepts and MAS governance ones. We have also argued that investing effort in this endeavor is relevant: engineering and governance of open MAS can be facilitated by defining norms, sanctions, and incentives using intuitive requirements concepts such as, e.g., goals and preferences; any change in the requirements can be automatically reflected on corresponding norms, enabling therefore MAS governance through requirements specifications.

We have introduced Requirements-driven Contracting (RdC), a framework for defining norms, sanctions, and incentives from a rich specification of requirements. To enable this, we first identified kinds of relevant requirements-related information usually expressed in natural language, then established a conceptualization in which these can be represented and reasoned about, enabling us subsequently to define an environment specification language. The language is simple enough to enable transformation of requirements-related information into norms, sanctions, and incentives, while being rich enough to cover various kinds of functional and nonfunctional requirements, domain assumptions, and preferences and priorities. The language uses extensible templates to allow combined use of RdC with methodologies for, e.g., measure definition (e.g., [43]). We have then defined governance concepts (i.e., obligations, permissions, prohibitions, sanctions, incentives, roles, and contracts) and related these to the conceptualizations at the requirements and environment specification levels. This allowed us to define procedures for transforming the environment specification (and therefore requirements) into collections of executable norms, sanctions, incentives, and contracts. By using RdC, the engineer ensures that all requirements and related information is represented at the governance level, or, in other words, that agents' behaviors are regulated in accordance to requirements.

Part IV

Conclusions

14

Outline of Part IV

Part IV closes the thesis. Because the contributions in Parts I and II rely on specific choices, these choices – for instance, the choice of one particular formal model of argumentation – are revisited in Chapter 15 in light of available alternatives. In particular, reasons are given (i) for choosing DOLCE over alternative foundational ontologies in Part I, (ii) for adopting Simari and Loui's model of argumentation over comparable ones in Part II, and (iii) for using a simple definition of preference throughout this thesis over some more elaborate model. It was noted earlier in the text that validation is herein limited to acceptance by the relevant research communities and case studies. Some of the contributions can be put to further validation. Chapter 16 therefore discusses how conceptual contributions in Part I and the methodological contributions in Part II can be further tested to better understand their respective merits and limitations. In addition to specific directions for future work given in each of the chapters in Parts I, II, and III, the final chapter of the thesis presents one important direction for future research that arises from the combined contributions of the thesis. Namely, a problem is identified that cannot be resolved by requirements engineering frameworks available at the time of writing. Based on the contributions of the thesis and related efforts, guidelines are suggested for readers interested in advancing the state of the art towards a solution of the problem in question. Chapter 17 also carries a summary and concluding remarks on the contributions of the thesis.

15

Remarks on Choices

Contributions in Parts I and II rely on specific choices for which the rationale has already been mentioned in these respective parts. The aim of this chapter is to provide more details on the reasons behind these choices in light of alternative choices that could have been made. Section 15.1 discusses why DOLCE is chosen over alternative foundational ontologies in Part I. Section 15.2 indicates reasons for adopting Simari and Loui's model of argumentation over comparable ones in Part II. Finally, Section 15.3 discusses why a simple definition of the concept of preference is used throughout this thesis over some more elaborate model.

15.1 DOLCE and Alternative Foundational Ontologies

A foundational ontology is a theory about the abstract domain-independent categories in the real world. Its main purposes are to act as a starting point for building new ontologies, as a reference point for easy and rigorous comparisons among ontological approaches, and as a foundational framework for analyzing, harmonizing, and integrating existing ontologies. Among the various available foundational ontologies (e.g., [294, 282, 68, 134]; for a comparison, see [187, 57]), we chose – in Chapter 4 – DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) [187], which rests on intuitively attractive ontological choices for the present discussion. DOLCE makes the following ontological choices:

- DOLCE is descriptive in that it aims to capture the ontological categories underlying natural language and human common sense. Categories in the ontology are therefore conceived as cognitive artifacts ultimately depending on human perception, cultural imprints, and social conventions. Descriptive contrasts to revisionary; choosing the latter commits to the aim of capturing the intrinsic nature of the world, that is, answering what exists, as opposed to what is perceived to exist.
- DOLCE is multiplicative, allowing different entities to be co-localized in the same space-time. Broadly speaking, in a multiplicative ontology, we can say "the paper ticket is constituted of an amount of paper", whereas in a reductionist ontology, we would say "the paper ticket is an amount of paper". Consequently, if we speak of the departure date, we say in the former case "the paper ticket has a departure date" and not "the amount of paper has a departure date" as it would be more appropriate in the latter case.
- DOLCE takes a possibilist view: entities are allowed independently of their actual existence. At first sight, possibilism seems well adapted to RE for we are concerned with optative statements, which speak of what is presently not but is desired to be. DOLCE seems to adopt possibilism as a consequence of adopting modal logic for its formal characterization (see, [215]). The actualism/possibilism discussion is considerably more elaborate, so that we may still properly deal with optative statements in (some variants of) actualism (for an introduction to the debate, see [220]).
- It distinguishes between and allows both enduring and perduring entities assuming thus that entities have both temporal and spatial parts. The ontological choice at play here is that between endurantism and perdurantism, which has consequences on the questions of how identity and change are understood. DOLCE's cognitive bias leads to allowing both endurants and perdurants, whereby they differ in that something is an endurant iff (i) it exists at more than one moment and (ii) statements about what parts it has must be made relative to some time or other. This is interesting in the present discussion mostly in that the cognitive bias is accounted for, so that statements about perdurants and endurants are equally accepted.

- DOLCE is an ontology of particulars – DOLCE takes the domain of discourse not to contain, e.g., the instantiation relation, so that no entity in the domain has instances. The consequence is that universals are not subject of being organized and characterized within, but are left outside DOLCE:

  > "We take the ontology of universals as formally separated from that of particulars. Of course, universals do appear in an ontology of particulars, insofar they are used to organize and characterize them: simply, since they are not in the domain of discourse, they are not themselves subject to being organized and characterized."

The taxonomy of basic ontological categories in DOLCE is shown in Figure 27. It is visible from the considerations above why DOLCE is an attractive choice when discussing a core ontology for requirements engineering that relates requirements to speech acts. Being a descriptive ontology, categories in DOLCE are defined to reflect notions that depend on human perception, cultural imprints, and social conventions. Since RE is oriented towards the resolution of practical problems, in which decisions are informed by stakeholders with perceptual bias, individual cultural and other background, and specific social conventions, an ontology that does not recognize such bias could hardly be an appropriate choice. Such an ontology would be removed from the reality of the problem that we intend to resolve. As its authors point out [215], DOLCE has a clear cognitive bias, for its ontological categories are intended to capture ontological categories that underlie natural language and human commonsense. This fits well with the approach adopted in defining our core ontology for RE, as we define the concepts of our ontology by drawing from types of speech acts, which themselves are intended to reflect the communication of content expressed in natural language. Note that the RE effort involves the representation of information that is communicated by the stakeholders, and is therefore a process in which already formed conceptualizations are made explicit. Categories in DOLCE are defined to be descriptive notions that assist in rendering explicit already formed conceptualizations. In this respect, by being descriptive, DOLCE appears to be relevant to the RE process which itself is descriptive, that is, aims to capture what is perceived and communicated.

It is DOLCE's notion of *quality* that we use to distinguish between functional and nonfunctional requirements– that is, goals and quality constraints – in our core ontology for RE. It is the way that qualities are conceptualized in DOLCE that made them particularly suitable for the purposes in this thesis. In DOLCE, qualities are basic entities that one perceives and measures. A quality differs from the concept of "property" in that the former is a particular, while the latter is a universal. Every entity comes with some qualities, and other qualities inhere in a quality. Qualities belong to a finite set of quality types, such as, e.g., size, color, etc. and are characteristic for – or inhere in – specific individuals. A quality (e.g., color) is distinguished from its "value" (e.g., some shade of red); the latter is called a "quale" and describes the position of an individual in a certain conceptual space, that is "quality space". Two particulars having exactly the same color thus carry color qualities which have the same position in the color space, or, in other words, have the same color quale. This conceptualization is attractive here, since it appears that natural language in certain constructs appears to make a similar distinction, and as we discussed earlier, this seems to be a relevant way to separate the content of desires that are functional – thus not directly referring to qualities, from nonfunctional, which directly refer to qualities and constrain values in the relevant quality spaces, that is, delimit admissible quales. As we argued above, this is further relevant herein because we can ground the difference between softgoals and quality constraints in the characteristics of quality spaces. Consider the following observation of DOLCE's authors:

> "Each quality type has an associated quality space with a specific structure. For example, lengths are usually associated to a metric linear space, and colors to a topological 2D space. The structure of these spaces reflects our perceptual and cognitive bias: this is another important reason for taking the notion of 'quale', as used in philosophy of mind, to designate quality regions, which roughly correspond to qualitative sensorial experiences of humans."

Referring to ill-defined and/or subjective quality spaces when distinguishing softgoals from quality constraints allows that separation to be grounded in how qualities and associated notions are assumed to be experienced and understood.

The choice of DOLCE is thus grounded in essentially three arguments: (i) DOLCE is descriptive, and the modeling effort in RE appears to be descriptive as well; (ii) DOLCE takes a possibilistic view, which seems closer than the actualistic perspective on how reasoning is performed during RE; and (iii) the notion of quality, quality spaces, and quality values which seem to fit the intuition well.

When choosing a foundational ontology, alternative foundational ontologies are compared over so-called ontology meta-criteria [216]. These meta-criteria reflect the possible important ontological choices, that we mentioned above. Among the dozen or so available foundational ontologies (for an overview,

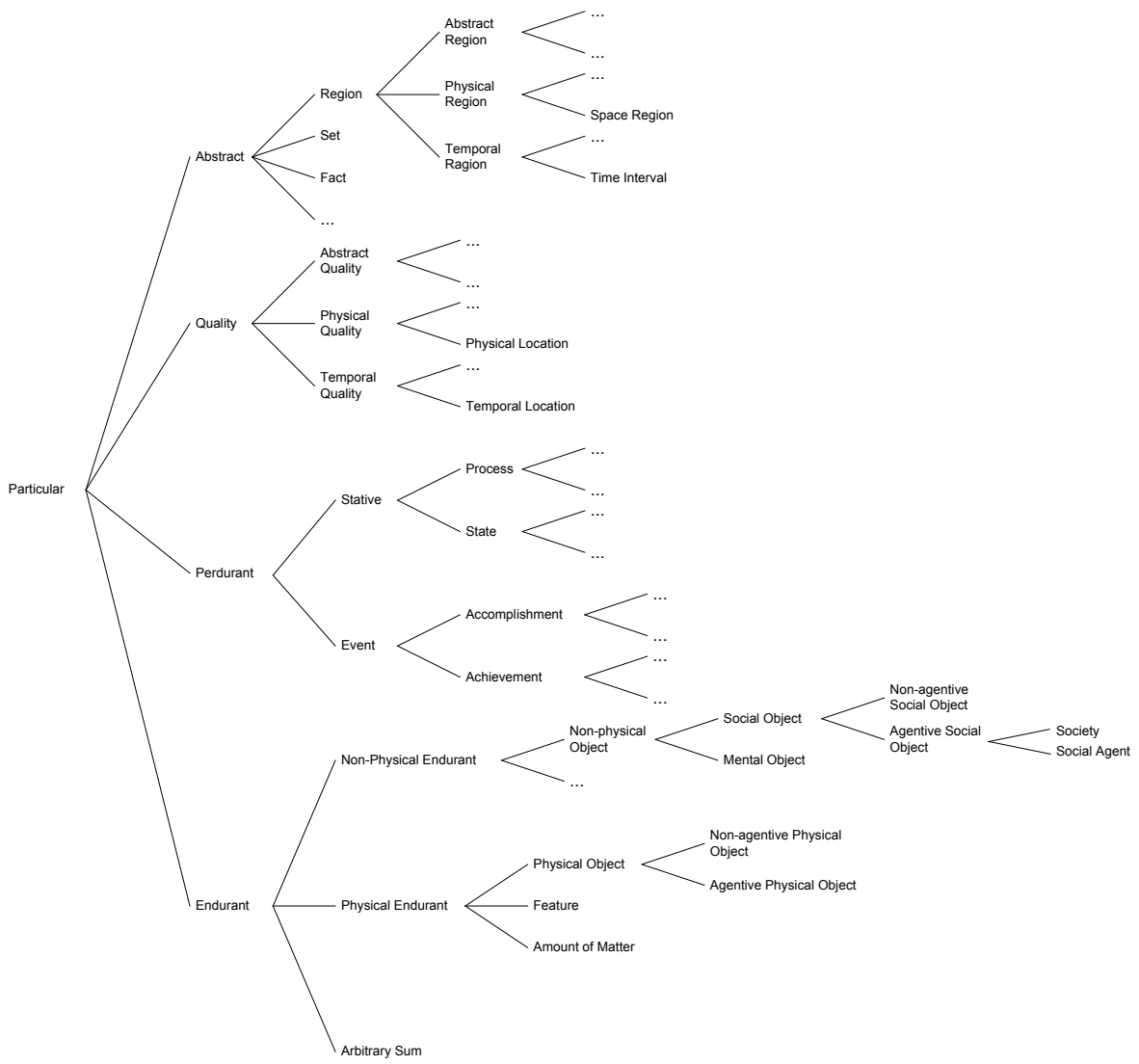


Fig. 27. Taxonomy of basic categories in DOLCE, borrowed from Masolo and colleagues [215].

Table 3. Comparison of main foundational ontologies and their ontological choices (from [57]).

	BFO	DOLCE	OCHRE	OpenCyc	SUMO
Descriptive	no	yes	no	yes	yes
Multiplicative	no	yes	unclear	unclear	yes
Possibilism	no	yes	yes	unclear	unclear
Perdurantism	yes	yes	no	unclear	yes

see [232]), four are considered in general as the most relevant ones, namely, BFO [294], DOLCE [215], OCHRE [282], SUMO [134], and OpenCyc [68] (see, e.g., [57] for a more detailed discussion). Requirements engineering is model artifacts of human commonsense, so that the foundational ontology of choice should be descriptive. It should also be multiplicative, since reductionism seems further away from human commonsense [233]. RE involves to a considerable extent the reasoning about what may and will be the case in the future, so that the ontology ought to be possibilistic. Finally, we would hope for an ontology in which endurants are distinguished from perdurants, which again seems to fit intuition [215]. Each of the foundational ontologies mentioned here, and shown in Table 3, has been defined with different purpose in mind and therefore subsumes different ontological choices. The closest with regards to the needs presented herein are DOLCE and SUMO. DOLCE is preferred herein over SUMO as the former commits more clearly to the ontological choices deemed appropriate with regards to the problem at hand, that is, the definition of a core ontology for RE. SUMO (Suggested Merged Upper Ontology)[1] aims to combine

[1] http://ontology.teknwledge.com/

categories and relations coming from different top-level ontologies in order to improve interoperability, communication, and search in the Semantic Web field. Given that SUMO merges different upper level ontologies, it is not influenced by an overall theoretical approach, but adopts general categories from various sources. It is neither clearly multiplicative nor clearly reductionist, and the same dilemma applies to its position with regards to the choice of either possibilism or actualism. It is classified above as descriptive, since it includes the commonsense distinction between objects and processes.

15.2 Choosing a Model of Argument

Detailed reviews of research on models of argument are available, and this section brings nothing new in this respect. Chesñevar and colleagues' overview of the literature [46] is closely followed. The interested reader will refer to their work for furhter details.

A model of argument, or argumentative system is a formalization of the process of defeasible reasoning: an argument A for a hypothesis h is a tentative piece of information that one would be inclined to accept as sufficient reason or explanation to accept h. As new information becomes available, the argument A may lose its support or become weakened, so that h may no longer be acceptable as valid. It is in this respect that nonmonotonicity arises in argumentation [287]. Recent work is influenced by Stephen Toulmin's [310] conceptual model of argumentation in law introduced in 1958. A disputation is visualized in his framework by chaining instances of four kinds of basic concepts, that appear in arguments and counter-arguments: the claim, the warrant (a nondemonstrative reason that allows the claim), the datum (the evidence for using the warrant), and the backing (the grounds underlying the reason). The notion of defeasibility is technically introduced in Hart's "Ascription of Responsibility and Rights" [120]. Mathematical approaches to argument models are related to some contributions in intuitionistic, paraconsistent, conditional, deontic, and dialogue logics, as well as belief revision. Pollock developed a theory of defeasible reasoning for computer science [260], in which reasons are assembled to form arguments, whereby there are defeasible and nondefeasible reasons. Reasons – called defeaters – can attack the justificatory power of defeasible reasons that support some conclusion. Pollock distinguishes rebutting defeaters, which attack a conclusion by supporting the opposite one, whereas an undercutting defeater is reason that attacks the connection between a reason and a conclusion. A conclusion is warranted according to Pollock only if it emerges undefeated from an iterative justification process.

Our choice of Simari and Loui's argumentation framework falls into line with the intuitive notions discussed and organized in Pollock's approach. Simari and Loui's framework presents a general theory of warrant, as introduced by Pollock [260], with Poole's notion of specificity [262]. The framework thereby effectively unifies approaches to argument-based defeasible reasoning. This framework has subsequently been extended [287] to allow the detection of various cases of fallacious argumentation so that these undesirable situations can be settled. It is this extended variant of Simari and Loui's framework that we use as the underlying model of argument throughout the thesis. The Simari-Loui framework is a simple and elegant framework in which very few assumptions are made about the content of arguments. The main interest is in establishing if an argument structure is acceptable or not. In this respect, the framework is flexible and allowed us to introduce clarification as an add-on in a fairly straightforward manner, as discussed in Part II. This flexibility further allowed us to suggest several ways to use the argumentation and justification techniques, depending on whether the content of arguments is given in an informal notation, a formal one, or a mathematical logic. This is of clear interest in RE for we rarely have formal content at the early phases when initial requirements are communicated by the stakeholders. In the last case, the fact that we rely on the Simari-Loui framework in its basic and accepted form lets the users of our techniques use subsequent advances in argumentation that rely on the same basic argument model. In summary, we chose the Simari-Loui framework primarily because it fits the RE setting well: (i) the Simari-Loui framework builds on intuitive ideas from Pollock's approach to defeasible reasoning, which in turn borrows from the philosophy of law; (ii) the framework is flexible with regards to the characteristics of the content of arguments, thus allowing us to deploy it in various ways depending on the information available during the early stages of RE; (iii) the framework allowed us to introduce clarification in a straightforward way; (iv) it is a framework that integrates established and uncontroversial ideas in the knowledge representation community. Moreover, the approaches to argumentation and justification adopted in this thesis can therefore be extended in the RE context by adopting ideas that developed as extensions to the Simari-Loui framework in the knowledge representation community.

Models of argument more advanced in some specific respects have been subsequently suggested. They are, however, not adopted herein mainly because the key basic ideas are given in the Simari-Loui framework, and in a general form, letting us commit to very few assumptions when arguing and justifying

and thus giving the requirements engineer the possibility to extend the basic approach outlined in the thesis in directions that cannot be anticipated herein. In 1993, Vreeswijk [326] introduced his abstract argumentation system, in which defeasible argumentation is modeled as a debate. His aim is to give a formal mathematical characterization of the concepts involved in the process of argumentation. He does not extensively study how argumentation should proceed. Since we are interested precisely in the latter issue, and given the apparent difficulty of formalizing arguments during RE, his approach is of interest as soon as this cost can be brought down to manageable levels, hence not at present. Brewka [41] is also primarily interested in formal properties of arguments, while focusing on the definition of specificity and extensions thereof in the aim of allowing advanced comparison – prioritization – of argument structures. Dung [79] provides a general formal theory of argumentation that relates argumentation to logic programming. Prakken and Sartor [263] introduce priorities that themselves are defeasible. Verheij [323] studies the role of rules in the argumentation process. Broadly speaking, rules give reasons that are subsequently used in arguments to support a conclusion. Bondarenko and colleagues [33] suggest a framework in which any theory formulated in a monotonic logic to be extended by a defeasible set of assumptions, and study logic programming within such a perspective. In conclusion, we can say that the Simari-Loui framework is a relevant choice when a simple formal account of argumentation and justification is needed, in which specificity is accounted for, and which can be flexibly used and extended. Using later frameworks in our setting would be warranted if further need arises for the formalization of the content of arguments in RE, and if the argumentation process in such a setting is to be automated.

15.3 The Preference Concept

The concept of "preference" appears in several places in this thesis. In Chapter 4, an attitude – in the sense of a favorable or disfavorable evaluation with some intensity – is taken to be a source of information about whether some requirement, domain assumption, or other is optional (i.e., is it necessary for the future system to satisfy it or not), or about how some requirement or otherwise compares to an alternative one. In the second case, we called the resulting information "preference". In Chapter 5, the term of "disposition" is suggested as a name for a preference order over alternative goals, be these hardgoals or softgoals. Since one of the aims in that chapter is to formally characterize dispositions, we added a new kind of formulas, *disposition formulas* to our formalism, such that if ϕ and ψ are formulas of $\mathcal{L}$ then $\phi \succeq_d \psi$ is a disposition formula. We denoted $\mathcal{L}^{\succeq_d}$ the language extended with disposition formulas. We then extended our structures to interpret defeasible formulas: we added a function υ which maps a real number (informally understood as a utility value) to non-disposition formulae. υ is then evaluated as follows: if $\phi \succeq_d \psi$, then $\upsilon(\phi) \geq \upsilon(\psi)$, which means that ϕ is preferred to ψ. In Chapter 6, the usual basic understanding of the concept of "preference" was used to model the information extracted from a softgoal, whereby such information conveys an order over alternative value of some measure. Chapter 8 uses several different order relations and interprets these as preference over alternative arguments depending on specificity, ambiguity, overgenerality, synonymy, or vagueness. Again, it is simply an order that is established, and that is informally interpreted as a preference order. Chapter 11 uses the notion of preference in the writing of so-called preference specifications, in which orders are defined over values of quality-of-service parameters. In Chapter 12, we let a service request to carry "preferences" between alternative services: more precisely, the set of preferences carries expressions that constrain the pool of potential services. These expressions are used to eliminate services that do not satisfy them. The notion of preference in that case is a simple one: whatever violates these expressions is less preferred, and the automated composer will avoid less preferred services. In Chapter 13, we understand a preference to be an order relation over alternative requirements, and introduce a additional kind of order relation that establishes an order over conflicting preference orders, and call this additional relation "priority". Priority amounts in that Chapter to a preference over conflicting preferences, which is necessary since tradeoffs appear. The preferences and priorities in the RdC framework presented in that chapter are used in algorithms for the definition of norms to establish preconditions over norms so that less preferred will be satisfied if the satisfaction of preferred norms fails.

The understanding of the concept of preference that underlies the thesis is a very simple one, and can be summarized as follows: a preference compares two alternatives to indicate which of the alternatives receives a more intense favorable (or a less intense disfavorable) evaluation by some stakeholder. Note that nothing is said of how preferences are combined, of what properties the order in question may have, of whether the favorable/disfavorable evaluation corresponds to the notion of utility (as in microeconomics). This is intended for an obvious reason: the thesis makes no contribution or critique to the notion of

preference. As the various chapters illustrate, the need for a notion of preference arises in resolving the problems raised throughout the thesis, and thus, the concept of preference cannot be avoided. None of the discussions, however, requires that more detail be given on the understanding of the concept of preference. The reason for this should be apparent: none of the results in the thesis – except the discussion in Chapter 5 – goes as far to require some specific assumptions on the properties of the preference order. In Chapter 5, the order is interpreted through utility, and hence the concept of preference takes a more precise interpretation in that chapter.

Note that the adopted understanding of preference is simpler than the one in basic theory of choice (as exposed in detail by Kreps [186]). Therein, a relation is called a preference relation if and only if it is complete and transitive. A choice function, for a set of mutually exclusive alternatives A, is a function $c : 2^X \setminus \{\emptyset\} \to 2^X \setminus \{\emptyset\}$ such that $c(A) \subseteq A$ for each $A \in 2^X \setminus \{\emptyset\}$, whereby $c(A)$ consists of the alternatives the decision maker may choose if he is constrained to A; the decision maker is assumed to choose one alternative only. For a decision maker who intends to choose the best available alternative with respect to a preference relation $\succeq$, the choice function is then: $c(A; \succeq) = \{x \in A \mid x \succeq y, \forall y \in A\}$, and since $\succeq$ is complete and transitive, $c(A; \succeq) \neq \emptyset$ whenever A is finite. A preference relation can be represented by a utility function $u : X \to \mathbb{R}$ as follows: $x \succeq y \Leftrightarrow u(x) \geq u(y), \forall x, y \in X$. Note that a relation $\succeq$ can be represented by a utility function U in the sense of $u : X \to \mathbb{R}$ (for a countable X) if and only if $\succeq$ is a preference relation. Utility is in this sense relate to preference in Chapter 5, but not in other discussions. Reasons for avoiding a more precise model of preference are compunded by the variety of discussions on how to interpret preferences. Deviations from rational choice as articulated in preference theories in economics have been observed and discussed: e.g., in situation where self-interest does not dominate [280], or when concerns for fairness are involved [86], or when the social context strongly influences choice [85]. The endowement effect – according to which the initial allocation of property rights affects an individual's valuation of a good – permits intransitive preferences [159]. Also, research suggests that preferences may not be well-defined at all: for instance, it was observed that the expressed preference depends on the way in which preference is elicited (e.g., [311]). The interested reader may consider the recent extensive discussion given by Korobkin and Ulen [183]. For technical details of the various characteristics that may be attributed to preferences depending on the assumptions that one adopts, the reader will refer to Öztürk and colleagues' overview [243]. The choice made in this thesis to adopt a simple notion of preference appears not only an appropriate one with regard to the problems that have been discussed, but a cautious choice as well.

16

Discussion of Further Validation

It has been noted in the introduction that validation of the contributions proves particularly difficult for they are conceptual and remain often at a too high level of abstraction to lend themselves to direct use and ideally experimentation. This is in general a recognized difficulty of the RE field, as industrial application often comes years after the contributions are presented to the relevant communities. Both major current RE frameworks – KAOS [71] and Tropos [50] – suffer from a lack of validation in realistic, industrial settings and across many organizations. Lack of trying is not the cause. The KAOS framework was introduced in 1993, and continues to be refined, improved, and extended. It is with the commercialization of the framework that validation thereof in industrial settings has been performed (e.g., [316]), though no thorough explanatory case studies are available that would relate the practices suggested in KAOS to perceived improvements in productivity, quality, and risk management. The same observation applies for the Tropos framework. We therefore follow the common practices of the field in terms of early validation of contributions. Appropriateness of the contributions is therefore argued on the following grounds. First, rigorous arguments are provided when identifying the deficiencies of related work and strong arguments and rigorous argumentation – as illustrated throughout the thesis – have gone into the elaboration of the suggested responses to the raised issues. Second, the various contributions are accepted by the respective fields: the chapters of the thesis have been peer-reviewed and published. Finally, the usual means of validation in the concerned fields – that is, preliminary case studies – are used throughout the thesis to provide concrete examples of how the contributions depart from related work and how they advance the available research results in the relevant fields of enquiry.

16.1 Reflections on an Exemplary Study

While it is clear that stronger validation is needed for the claims given in research on RE, rigorous studies are rare (e.g., [70, 131, 143, 164, 165, 324]). A recent study by Damian and Chisan [70] provides an exemplar and at the same time allows us to point out the difficulties that plague at present the further validation of the contributions of this thesis.

Damian and Chisan [70] conducted an explanatory case study over the course of 30 months in a medium-sized software development organization. The setting is particularly attractive since the effects of the introduction of an RE process can be observed, and therefore, involved employees questioned about the perceived benefits thereof. The study relies on data collected by way of questionnaires and interviews at the various times over the course of the study period. Previously, the organization had no systematic approach to the RE activities, so that stakeholders provided requirements in natural language and without much structure, definitions of key terms were often missing, requirements were not documented, collective understanding of requirements spread by word of mouth, design depended on the individual designers' interpretations of the requirements, and requirements still could change at the later stages of the development process. The organization then introduced systematic approaches to the decomposition of required features, documented traceability information, organized group analysis sessions, engaged participants in cross functional teams to discuss and refine the requirements, required that the rationale for requirements be specified in addition to the technical details, test scenarios were defined for the various requirements in order to guide system testing activities. Data collected by the investigators shown tangible and intangible benefits. For instance, developers perceived that they understood requirements better, their decisions were better informed, and the quality of their communications with other participants improved. Investigators' results indicate overall success of the new RE processes in terms of software

product quality, productivity, and risk management. For example, fewer defects seem to be one of the effects of the use of traceability links, peer-reviews, and requirements validation.

Interest for a similar case study setting is clear for the contributions outlined in Part II of this thesis. It would indeed be appropriate to proceed to an explanatory case study in which the effects of introducing argumentation, justification, and clarification in the group review of requirements are studied. Obstacles are, however, clear. It is necessary that there be an organization that already uses goal-oriented RE. The organization should be willing to introduce the techniques suggested in this thesis in its requirements processes. The period of use should be sufficiently long to allow results to become perceived by the participants and to reflect – positively or negatively – in the characteristics of the product and of the development process. It is clear that opportunities for such a study are at present missing, and will appear only when goal-oriented RE spreads further and the organizations using it perceive the interest of improving the controls of such the goal-oriented RE process.

It has already been observed that the contributions in Part I are conceptual and do not involve specific techniques that can be put to immediate use. Their overall novelty lies in the increased expressivity. As it has been discussed throughout the thesis, the added expressivity is not specific to some subclass of RE problems, but is relevant for any RE effort. In this sense, the validation lies in showing that the added expressivity is necessary and beneficial. Theoretical arguments for this to be the case are clear, including those that point out that RE problems involve conflicts and therefore need some form of ordering of alternatives, a service provided here by preferences over requirements or otherwise, or preferences over preferences (which amount to priorities among preferences). To practically validate this, requirements frameworks that integrate the new concepts are needed and ought to be compared to the available frameworks. To validate this rigorously, one confronts the same problems as those mentioned for contributions in Part II: sufficiently many comparable users would be needed, working on same problems, yet using different frameworks, or one set of users moving from the current frameworks to those that integrate all of the concepts introduced in Part I. As observed in Part I and further treated in Chapter 17, there are no such frameworks, and it is far from satisfactory to simply extend the current frameworks with the new concepts – they require novel RE techniques and interactions between techniques and those in available RE frameworks cannot be anticipated at this time. In this respect, attempting elaborate empirical validation of the contributions in Part I would hardly be appropriate. More work is first needed in integrating these contributions into the state of the art. It should be noted, however, that the interest in the added expressivity brought by some of these concepts has already been confirmed in preliminary case studies reported elsewhere [155, 126], and conducted by Caroline Herssens, Stéphane Faulkner, and the author of this thesis in cooperation with the European Space Agency (ESA). It was observed that the added expressivity is relevant in describing and dealing with quality-of-service considerations in a web service that the ESA maintains to allow researchers to access satellite data on vegetation indexes for various reagions of the globe. The observations did not, however, allow cost-benefit conclusions to be drawn.

16.2 Pointers to Further Validation

Practical indications can be given on how to conduct further validation of contributions in Parts II and III of the thesis. Recall that Chapter 8 advances several claims that are intuitively attractive and appropriate; they are intended to justify the introduction of argumentation, justification, and clarification techniques:

1. We have suggested that it is not apparent from a goal diagram why a particular natural requirement is represented in a particular way and not another one. A consequence thereof is that one could expect that the stakeholders reviewing the diagram will not know why another stakeholder or the requirements engineer made the given modeling decision. The result may be unnecessary reviews of the diagram, changes, or additional explaining. Such activities undoubtedly require time and resources that could have been employed in other activities.
2. It was then argued that knowing the arguments for a previous decision would facilitate future modeling decisions – arguments would better inform subsequent iterations during modeling.
3. We suggested that the failure to record the rationale of the modeling decisions – that is, the arguments behind the decisions – leads to ideas, arguments, assumptions underlying a decision to remain implicit and/or lost over time. Alternative ideas and differing views are then unavailable, although they could have led to other, possibly more appropriate modeling choices.
4. Without explicit arguments, there is hardly any guarantee that requirements are interpreted in the manner intended by the stakeholders. It was thus relevant to suggest clarification techniques as a way

to check whether the understanding is close to what was intended. We argued that leaving clarity checks implicit is likely to lead to the application of trivial and intuitive checks, while non-trivial ones will be disregarded for simplicity.

5. Finally, clarification allows inconsistencies to be detected earlier in the RE process. We observed that, the longer the misunderstandings persist, the more they will influence subsequent decisions, and the more costly it becomes to correct the consequences of the misunderstanding.

Consider the first two observations above. Both indicate that making arguments behind modeling decisions explicit will lead to a more efficient goal modeling process. The difficulty here is what construct to employ to measure the efficiency of the modeling process. To be internally valid, the construct of choice should capture the characteristics of interest of the phenomenon being studied; to be representationally valid, the construct should translate into observable phenomena. To measure improvement in the efficiency of a goal modeling process, time invested in revising a goal model can be relevant, as less time spent in revisions is undoubtedly an improvement in the efficiency of the modeling process. It can also be relevant to define the measure according to the specific context in which the measurement is made: preliminary interviews with the participants of a future study can reveal what they consider to be a relevant measure for the improvement of efficiency in the goal modeling process. While time spent revising a goal model is measurable in practice and appear to capture what the phenomenon of interest in the study, this construct suffers from issues of intenal validity: namely, it is difficult to assume that time spent in revising a goal model depends solely on the presence or absence of explicit arguments for modeling decisions. For instance, a goal model can be revised as a result of changes in the environment that is represented in the model, and that are unrelated to available arguments, revision can occur after renegotiation of previous choices; new requirements or the obsolescence of already represented requirements also leads to revisions of the model. It is difficult to isolate these variations from those that result from the misunderstanding of the model due to absence of explicit arguments. If such fine separation were possible, external validity (which asks whether the results can be generalized beyond the context of the specific study) would mostly depend on the characteristics of the organization(s) involved in the study. On the other hand, if separation is not made, specifics of the project itself (i.e., the project of developing the future system) would harm external validity: e.g., when requirements are engineered for a system with which participants have very limited experience, revisions are likely to be more frequent as omissions will be noticed as goal modeling advances and as participants learn about the future system and its environment over the course of the modeling process. A more realistic approach is to proceed by way of explanatory study, and in particular collect data on perceptions of the participants, in the first project in which they use argumentation, justification, and clarification techniques. Likert and Likert [201] point out that perceptions are a relevant indicator:

> People act on the basis of what they perceive the situation to be, whether the perceptions are accurate or grossly inaccurate. Since behavior is based on perceptions, the existence of each of them is a fact to be considered. Similarly, the frustrations, attitudes, loyalties, and hostilities felt by each member and the information and misinformation possessed by each are facts as is their evaluation of the merits and desirability of each particular course of action under consideration.

Damian and Chisan [70] encounter similar problems and resolve them by relying on engineers' perceptions of improvements due to the move from an informal to a systematic RE process. A questionnaire can then be defined, and a Likert scale used to allow respondents to state their agreement or disagreement with statements such as, e.g., "The argumentation technique reduces the time invested in revision of the goal diagram." While attractive, this approach may suffer from at least three kinds of bias: respondents may avoid using extreme responses (central tendency bias), they may strongly agree with the statements offered (acquiescence bias), or may choose answers they deem desirable in the given study (social desirability bias). Techniques exist for dealing to some extent with each of these. For instance, the questionnaire can be used in combination with the Marlowe-Crowne scale. The latter is a 33 item self-report designed to assess an individuals tendency to answer in a manner that is socially desirable. Participants rate whether each of the 33 statements are true or false. Respondents that strongly favor socially desirable responses can then be discarded from consideration in the statistical analysis.

The third claim made in relation to the argumentation, justification, and clarification techniques is that richer goal models can be obtained if clear arguments are explicit. The fourth claim indicates that better understanding results from the use of the suggested techniques, and finally, the fifth claim indicates that the techniques help detect inconsistencies earlier in the RE process. It is apparent that the practical approach to test these claims will involve perceptions and will follow the same methodology as that suggested for the first two claims. Statements can be provided and respondents asked to rate the extent to which they agree or disagree. Each statement will concern the use of a particular technique with regards

to an issue, such as the early detection of inconsistencies, or better understanding of requirements. Again, the same sources of bias ought to be controlled – guidelines can be found in any advanced textbook on research in social sciences (e.g., [61, 37, 48, 14, 221]).

The content of the actual questionnaire will depend on the specifics of the setting in which the data is collected. For instance, the statements the respondents are called to evaluate should be written so that their meaning is clear and accords with the meaning intended by those conducting the survey. In this respect, preliminary versions of the questionnaire ought to be reviewed by two or more members of the organization who share the vocabulary of the respondents. The statements in the questionnaire should be defined so that the resulting data will allow us to draw conclusions about the agreement or disagreement of respondents to the following statements:

- When argumentation and justification are used, less time is spent revising the goal diagram than when these techniques are not used.
- When clarification is used, less time is spent revising the goal diagram than when this technique is not used.
- When argumentation and justification are used, modeling decisions are better informed than when these techniques are not used.
- When clarification is used, modeling decisions are better informed than when these techniques are not used.
- When argumentation and justification are used, requirements are misunderstood less often than when these techniques are not used.
- When clarification is used, requirements are misunderstood less often than when this technique is not used.
- When argumentation and justification are used, richer goal models are defined than when these techniques are not used.
- When clarification is used, richer goal models are defined than when this technique is not used.
- When clarification is used, less inconsistencies are observed late after the RE process is completed than when this technique is not used.

All of the above should be covered since they convey the claims made in Chapter 8 and discussed above. The setting for the explanatory case study ought to be such that a sufficient number of usable responses can be collected.

Contributions in Chapters 10 and 11 require a similar approach. An explanatory case study of contributions in Chapter 8 should precede them for it already deals with the problems of using argumentation and justification in an RE process. It would in this respect provide valuable indications for problems to avoid when conducting explanatory case studies to validate claims in Chapters 10 and 11.

Further validation of contributions in Chapters 12 and 13 is a different matter. Chapter 12 has already been subjected to preliminary validation in the form of simulations and results thereof have been compared with related approaches. The next natural step is an implementation of the automated service composer in a web services environment. Contributions in Chapter 13 require that the implementation of Martin Kollingbaum's model of normative multi-agent system be completed. His PhD thesis [178] suggests the NoA Normative Agent Architecture, which is a reactive planning architecture based on an abstract model for the implementation of norm-governed practical reasoning agents. The abstract model in question describes how agents choose their actions based on explicitly represented normative concepts, namely the obligations, permissions, and prohibitions in a process of norm-governed practical reasoning. Using such a test environment, it will be possible to see how practical it is to use the algorithms we suggested and to identify limitations (then suggest responses thereto) of the RdC approach outlined in Chapter 13.

17

Summary and Perspectives

Efficient organization requires rigorous and systematic information management, which encompasses information processing and decision making. Within the efforts in management science and informatics invested towards advancing the knowledge on, and providing assistance to decision making, this thesis focuses on the conceptualizations and techniques intended to facilitate the identification, evaluation, and selection of decisions during the earliest stages of information systems engineering, whereby the systems of interest are deployed to partly or fully automate various organizational processes, including information processing ones. The overall motivating problem that drove to, and that unites the various contributions presented in this thesis is how to better inform decision making and guide it towards decisions that will increase the quality (as evaluated both by the engineer and the stakeholders) of the information system being engineered.

17.1 Revised Conceptual Foundations for Requirements Engineering

The first part of this thesis focuses on conceptualizations that facilitate the identification of relevant information and its organization for subsequent analysis, all in the aim of achieving high quality of the system being engineered. In particular, Part I discusses, shows deficiencies, and accordingly revises the conceptual foundations of requirements engineering, a field of information systems engineering that focuses on the identification and analysis of requirements communicated by the stakeholders to the engineer of the system.

A decade ago Zave and Jackson [339] observed that the field of requirements engineering (RE) left behind simplistic approaches to understanding what a system will do in favor of novel and varied terminology, methods, languages, tools, and issues considered to be critical. Despite continued progress, one particular area of RE remained weak – namely, there is limited consensus as to the precise meaning of the basic terminology used in RE. It does not require considerable effort to see that the field richly and variously defines the requirement concept. For instance, Ross and Schoman's [279] early definition of RE points out that requirements describe the purpose of a system:

> "Requirements definition must say why a system is needed, based on current or foreseen conditions, which may be internal operations or an external market. It must say what system features will serve and satisfy this context. And it must say how the system is to be constructed."

In a survey of research efforts in RE, Zave [338] elaborates:

> "Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families."

Letier and van Lamsweerde [198] concur while placing emphasis on the critical role of system goals:

> "Requirements engineering is concerned with the identification of the goals to be achieved by the system-to-be, the operationalization of such goals into specifications of services and constraints, and the assignment of responsibilities for such services and constraints among human, physical, and software components forming the system."

A different terminology is used in the IEEE Guide to the Software Engineering Body of Knowledge [133] and Kotonya and Sommerville's textbook on RE [184]:

> "[Requirements engineering] is concerned with the elicitation, analysis, specification, and validation of software requirements. [...] Software requirements express the needs and constraints placed on a software product that contribute to the solution of some real-world problem."

The above agrees with Goguen and Linde [107] that requirements express needs:

> "A basic question in Requirements Engineering is how to find out what users really need."

To go beyond needs, one might consider the IEEE 610 standard on Software Engineering Terms [132] and read:

> "[A requirement is]: (1) a condition or capability needed by a user to solve a problem or achieve an objective; (2) a condition or capability that must be met or possessed by a system component to satisfy a contract, standard, specification or other formally imposed documents; (3) a documented representation of a condition or capability as in (1) or (2)."

The requirement concept has evidently been variously defined as a purpose, a need, a goal, a function(ality), a constraint, a behavior, a service, a condition, or a capability. Surveying more definitions would give same or other interpretations, but the point remains the same: the requirement concept is richly interpreted and variously used. The problem here lies not in the need for rich conceptual foundations, but in the fact that there actually are no common conceptual foundations on which the various efforts can be compared and benefits of their accumulation and combination reaped. Definitions above are hardly synonymous.

Part I of the thesis addresses this problem. Starting from the review and discussion of conceptual foundations in RE – and this through a view focused on the usual notion of quality – the core ontology of the RE field is reviewed and revised in Chapter 4. The revised ontology has several advantages, highlighted throughout the thesis. One among these that stands out is the fact that the ontology is grounded in the communication between the stakeholders and the engineers of the information system. Having such a grounding allowed us to make explicit many important aspects of the requirements problem, which have until now either been implicit or absent from much of the related works. In particular, the core ontology should incorporate concepts needed to model the content of the beliefs, desires, intentions, and attitudes that the stakeholders communicate. More light is consequently shed on the intended interpretation of concepts such as "goal", "softgoal", and so on, commonly used in RE frameworks. Definitions grounded in a foundational ontology are given for functional and nonfunctional requirements, thus allowing their precise distinction. We have shown that quality intervenes in important respects in the requirements problem, and have been able to define the aim of the requirements engineer as that of resolving the revised requirements problem. In revising the requirements problem, we have hopefully advanced the degree of precision of the definitions of RE. Compared to the definitions mentioned above, ours does not define RE as a sequence or collection of tasks, but instead focuses on the aim of RE – the advantage is that the aim remains the same regardless of how RE proceeds, and provided the aim is precise.

Apart from revising the conceptual foundations and the formulation of the essiential problem in RE, Part I of the thesis also studies two important related problems: (1) how different variants of the goal concept fit with the aim increasing the quality of an information system, and (2) how one particular kind of goal – the softgoal – can be interpreted in order to elicitate more information about the quality expectations of the stakeholders. With regards to (1), Chapter 5 discusses the different kinds of goals that are used in the modeling of IS requirements, giving formal definitions where appropriate. Increasing quality requires a particular approach to the satisfaction of the various kinds of goals. Given the new formulation of the requirements problem, it is argued that satisficing goals no longer appropriately describes the effort invested in RE, but that a different notion, that of excelling is more appropriate. Broadly speaking, excelling amounts to continually revise the satisficing thresholds: the thresholds established for satisficing are improved over time in order to increase the quality of the IS. It is argued that this reflects current industrial practice: IS are improved through revisions and upgrades, so that quality does improve (at least, this is what is desired) over time, and this by continually aligning the IS to revised quality requirements. With regards to (2), Chapter 6 discusses th slippery softgoal concept, necessary for dealing with abstract nonfunctional (i.e., quality) requirements, such as security, safety, convenience, and so on. It is argued that the analysis of softgoals should account for the multiple facets of the concept, and guidelines are given for the construction of methodologies that aim to properly address softgoals and account for them during decision making.

17.2 Control of Modeling Decisions during Requirements Engineering

Various approaches have been suggested for the tasks involved in RE. Established RE methodologies are systematically extended and enriched with novel step-by-step instructions to the elicitation and transformation of requirements using models. A common difficulty of these available contributions lies in the fact that the rationale of decision making involved in the various steps of these approaches is not explicit. As we have argued in Part II, this may not be an issue when dealing with toy systems, but is clearly a significant concern when engineering real-life information systems. We focused on the rationale involved in the decision making when performing goal modeling. When an instance of a goal model – which representes the purpose of the information system – is constructed by the engineer and few stakeholders having similar backgrounds, during a very limited amount of time, and consequently for a relatively simple system, there is generally no need to record the details of the decision process that has led to the final goal diagram. However, as the system under scrutiny gains in complexity, the inherent inability of individual human stakeholders to grasp the full extent of interactions and interdependencies between system components, to predict with detail and/or certainty the future conditions in which the system is expected to operate and the influence of such conditions on its functioning, makes the construction of the goal diagram an intricate task, critical for the success of the subsequent IS development activities. We have argued that – given incomplete, imprecise, and often qualitative information – modeling decisions are defeasible. This fits with the perspective adopted in Part I, where we acknowledge the importance of defeasible reasoning in solving the requirements problem. This leads – in Part II – to the study of how defeasible reasoning can be incorporated into established decision making processes involved in the identification and analysis of requirements, with particular care placed on decision making during goal modeling.

It is suggested in Part II that goal modeling ought to be complemented with activities to externalize and document arguments that led the stakeholders to express some requirements as well as the arguments that led the engineer to transform these requirements into model content. Once the arguments are known, it is necessary to evaluate their appropriateness, for it is not adequate to admit any argument in the process of justifying a modeling decision. It is thus necessary to to analyze the clarity of these arguments and of the information given within the model, revising it if proves necessary. To facilitate the argumentation, justification, and clarification tasks, the thesis proposed the so-called "Goal Argumentation Method (GAM)", which integrates a decision process, an argumentation model, and techniques combined to enable the analysis of argument structure, justification of modeling choices, and clarification of information appearing in arguments and elsewhere in the the goal modeling decision process. Drawing on design rationale literature, the decision process suggests an intuitively acceptable organization of the goal modeling task. The argumentation model, inspired by work in artificial intelligence argument models, is introduced in the evaluative step of the decision process, allowing various degrees of structure and rigor in the provision of arguments for modeling choices. The analysis techniques serve for the justification of modeling choices, the study of argument interaction (e.g., defeat and counterargumentation), the detection of deficient argumentation, and the checking for unclear information and subsequent clarification. An important characteristic of GAM is that it does not integrate a particular IS model; instead, it is independent of specific syntax and semantics of the IS model, allowing its combined use with any available RE framework concerned with the modeling of the purpose of the IS using the notion of "goal".

17.3 Perspectives

Part III of the thesis focused on four specific issues in information systems engineering and studied how the contributions introduced in Parts I and II can be applied to approach and resolve existing problems from a novel perspective and using novel means. While each chapter highlights the specific issues to consider in future efforts, the thesis taken as a whole opens up an important direction for future work that may not be apparent to the reader. The aim in this section is to outline at some length a problem that naturally arises from the study of the requirements problem in the context of adaptable and open service-oriented systems, and this when the contributions of this thesis are accounted for. As it is shown below, the problem is far from trivial and requires advances in several lines of research within requirements engineering. This thesis, with its contributions outlined in Parts I and II, and specifically the application of these contributions in Chapters 11 and 12 provides first steps in this long-term research endeavor.

17.3.1 Introduction to the Problem

In industry, service-oriented computing [246] is expected to facilitate the building and evolution of everything from small- to large-scale enterprise and other applications, by ensuring that the systems in question are, among other, scalable, evolvable, interoperable, and adapt cost-effectively and quickly to their users' needs [207]. Service-orientation is a polysemous term; in standardization efforts in industry, it is broadly understood to stand for "a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains" [207]. If we take the banking sector as an example, experiences with service implementations (e.g., [129, 20]) seem to indicate that service-orientation does facilitate application integration. Experiences in the German banking sector show that facilitated integration opens new possibilities in how banking is done:

> "To decrease costs and simultaneously enhance customer utility, banks are increasingly focusing on their individual core capabilities while exploring different sourcing options for non-core capabilities. Consequently, they are disaggregating their value chain into independently operable functional units. As communication capabilities reach higher levels of performance and reliability, these functional units are combined across corporate borders, thereby increasing sourcing options and flexibility." [129]

Independent functional units can therefore perform their function both for their owning organization *and* outside firms whish require the given functionality (e.g., [135]). Broadly speaking, such units are called services within the services perspective. While services can evidently be much less elaborate than entire organizational units (as fully automated web services usually are), the same broad principles apply: both are self-describing and self-contained modular units designed to execute a well-delimited task, and have standardized interfaces to the outside environment. Continuing with the banking problematic, and assuming service-orientation, it is not difficult to picture an efficient approach, e.g., to loan provision. First, the loan institution would use separate services for each of the various activities involved in loan provision (e.g., information gathering and assembly, credit analysis, application evaluation, risk evaluation, customer service, customer administration, refinancing, etc.). Second, there would be competing services for each of the tasks, so that the loan institution can choose for each task the service which suits it best. Third, as new services become available, the loan institution would revise the composition of its value chain, using new services instead of those previously employed. Ideally, the process of selecting and coordinating the services would be automated, so that the loan institution continually uses only services that "best" fit its requirements in terms of, e.g., efficiency, privacy, security, and of the intended offering to customers who introduce loan applications.

While loans cannot be provided in the described manner at the time of writing, the potential benefits of doing so are apparent: (a) individual services may achieve economies of scale by working for more than one loan institution; (b) the loan institution may benefit from the focus (i.e., specialization) of individual services on their respective tasks; (c) the loan institution may economize on switching costs due to interoperability. Furthermore, if the service selection process is automated, the loan institution may further economize on (d) human costs, for it will not invest resources in manually selecting appropriate services, and (e) time needed to identify and switch to better services.

Whether we focus on banking, travel or some other sector, firms unanimously seek the said efficiencies (a)–(e). If we abstract from the hypothesized loan provision example, realizing these efficiencies requires solving the following problem:

> Given a large number of competing, interoperable services which describe their offerings using standardized and machine-understandable notations:
> 1. How do we express the business expectations that need to be satisfied through the use of some of these services?
> 2. How do we use the set business expectations to automatically select at all times the services that can best satisfy the said expectations?

Broadly speaking, the above is a *service composition* problem [219] with emphasis on the writing expressive service requests. Recall that, in an adaptable and open service system (AOSS), a *service request* represent stakeholders' requirements in an unambiguous and machine-understandable format. Service requests are created at runtime: stakeholders express requirements, which are then translated into service requests. Each service request is submitted to an automated service composer in the AOSS, which then identifies the individual services whose coordinated execution fulfills best the requirements represented by the service request. Two salient characteristics of any AOSS are that the pool of potential

services varies and individual services' actual (as opposed to advertised) properties are unknown to the service composers before the said services become available. Therefore (i) how and how well requirements are satisfied varies at runtime, and (ii) as new services become available, requirements unanticipated at development time can be satisfied at runtime. Requirements engineering in such a setting involves (a) translating stakeholders' requirements into service requests, and (b) informing the stakeholders' of how and how well their requirements are satisfied, and of the new abilities of the system made possible by newly available services. A requirements engineering framework appropriate with regards to the issues (a) and (b) will assist and ideally automate the elicitation and analysis of expressive stakeholders' requirements, and their translation into service requests. Moreover, it will ensure that stakeholders are informed of how and how well the requirements have been satisfied, and what new abilities the AOSS has.

17.3.2 Problem

Service comosition in an AOSS can proceed as shown in Figure 28. An arbitrary[1] idle service composer in the AOSS receives a service request (see, plate a in Figure 28), described in terms of functional (drawn as a process in this example) and quality requirements (b). The composer interprets the request, identifies services appropriate w.r.t. functional and quality requirements (c), allocates services to tasks in the process, and coordinates the execution of these services (d). As new services appear (e), the composer continues to execute the same composition (f) which gives a certain level of success (g), whereby the success rate designates the proportion of request executions which satisfy all requirements laid out in the service request. In parallel, a duplicate composer explores new compositions by randomly choosing new services among those that appeared (h), and allocates them to appropriate tasks in the process (i). New composition that gives a higher success rate is then used instead of the old composition (j). Illustrative success rate data in Figure 28 comes from experiments we presented elsewhere [148, 149].

The above is one approach to the service composition problem. It assumes that the process to execute and the quality considerations are given in the service request. We used these assumptions elsewhere [148, 149] to focus on the problem of how the service composer ought to revise prior compositions to account for adaptability and openness of the AOSS. Looking at alternative approaches, we observe that the choice of assumptions about the characteristics of the service system and of the service requests influence the solution to the composition problem as follows:

- *Are only functional (A) or both functional and nonfunctional (B) criteria used in selecting services for the composition?* When selecting relevant services from the pool of available services, the composer first needs to identify those services that can perform the needed functionality. If only functional criteria are used (A), there is no apparent way of comparing competing services which can all perform the same functionality (e.g., [340]). To add nonfunctional criteria (B), it is necessary to assume a QoS ontology for services (e.g., [69]), which the providers use to advertise services' nonfunctional characteristics, and over which the composer can compare alternative services.
- *Is the process to execute known (C) or not (D) in the service request?* If unknown, a description of the goal state is usually given. From there on, planning can be applied to identify, given a set of services whose functionality is known, the sequence of services whose execution leads to the goal state. If the process is known, the compositon problem amounts to allocating the tasks to services that can execute the given tasks.
- *Does the composer rely on advertised (E) or observed (F) values of nonfunctional characteristics of individual services?* When several services provide the same functionality, the composer compares the services based on their nonfunctional characteristics. Service providers advertise the values for their services' nonfunctional characteristics. If the composer rates the services based on advertised values (E), it is assumed that the services attain advertised performance levels in all executions. Otherwise, the composer compares services based on observed performance in prior compositions (F), consequently accounting for discrepancies between advertised and actual behavior.
- *Does the composer revise compositions as new services become available?* No revision (G) implies that each composition is optimal, and thus that there can not be new services that perform better than those already available. If compositions are revised (H), the composer may proceed to replan the composition to explore new compositions based on newly available services.

[1] The AOSS architecture is more elaborate than described above. Namely, we do not allocate requests always to arbitrary composers, but have specialized composers for requests that arrive above some threshold frequency. We do not discuss this further here; the interested reader is referred to [148].

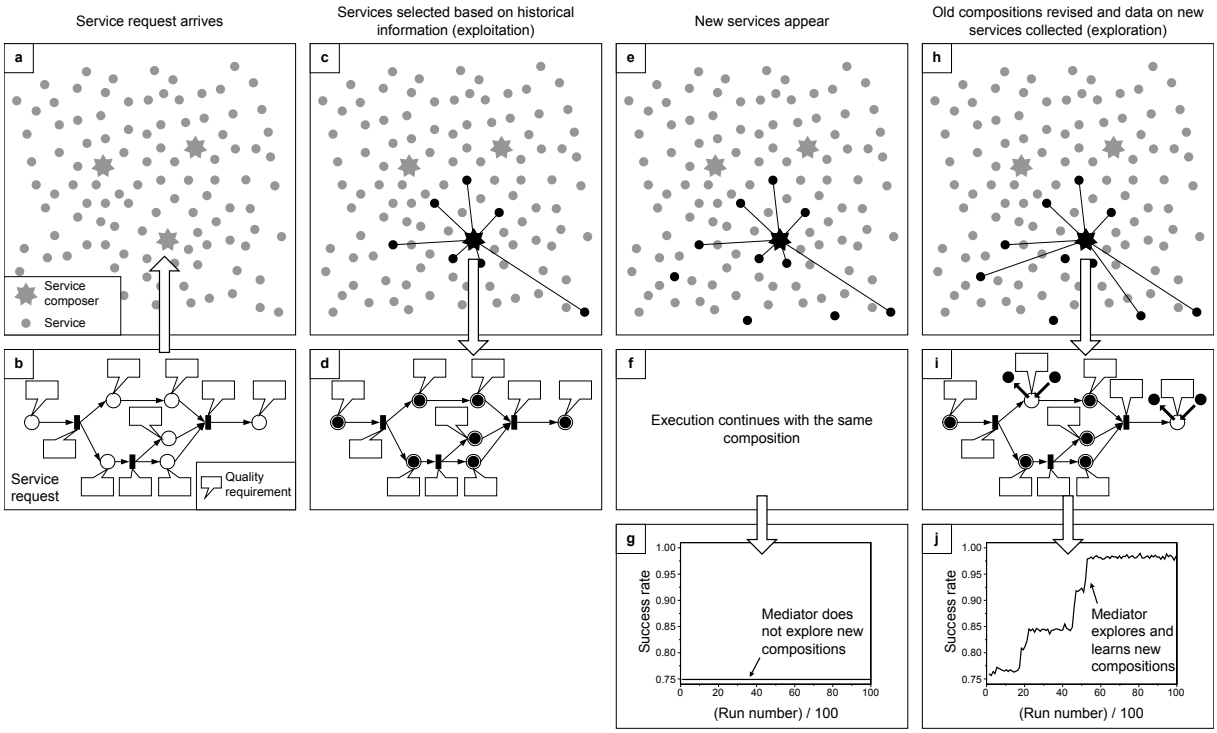


Fig. 28. An illustration of service composition in AOSS.

- *Is the set of all functional and nonfunctional characteristics for services known regardless of what services are available?* If yes (I), any new service appearing in the system necessarily performs some functionality that is known, and over which the users can express requirements. Same applies for nonfunctional characteristics – all are known in advance, so that any new service advertises its nonfunctional characteristics over those already known. On the Semantic Web, this is equivalent to say that the ontologies of functional and nonfunctional characteristics are known in advance. If the set of all possible behaviors and qualities is known, we do not encounter an important issue outlined at the outset of the paper: namely, we do not need to inform the users about the new requirements that the system may satisfy. This is somewhat unrealistic in the context of AOSS, for it would imply that no service provider can offer new functionality through its services. If unknown (J), users need to be informed of new functionality that becomes available as new services appear: ontologies are not entirely predefined, but need to be updated at runtime.

Prominent results in service composition rely on the Golog [199] logic programming language. Starting from generic process descriptions and user preferences, service compositions are planned to satisfy preferences [218, 226, 297]. In this respect, they rely on functional criteria in selecting services (A), plan compositions from goals and preferences (D)[2], rely on advetised functional characteristics (E), and either do not consider the revision problem, or when they do (H), the composition is replanned [195, 213]. The functional ontology is assumed predefined for the problem of new functionalities is not addressed (I). Another noted approach [255, 254, 167] combines the features (A, D, E, G, I); assuming that individual services are described as stateful processes with BPEL4WS [9], a goal state is partially specified in temporal logic, a plan that satisfies the goal is synthesized through planning via symbolic model checking, and the plan is translated back into BPEL4WS so that it can be submitted for execution. For other related efforts, the reader is referred to discussions in the relevant literature (in particular, see, [255, 297, 149]). Within this literature, we studied automated composition under the feature set (B, C, F, H, I) and proposed a novel reinforcement learning algorithm which allows us to account for both functional and nonfunctional considerations when allocating services to tasks given in a process model (see, Chapter 12). The algorithm takes nonfunctional criteria to optimize, a process model, a set of hard constraints on nonfunctional characteristics of services, then learns the optimal allocation of process steps to services (i.e., creates a service composition), and continually explores allocations of newly available services to process steps (thus revising compositions as new services appear). This last characteristic led us to associate the

[2] Though some generic procedure for how to achieve the goal is usually given – see, e.g., [297].

term "adaptable" to open service systems for the algorithm guarantees that new services are taken into account when compositions are revised. Still, the approach to composition in AOSS that we advocated suffers in that it requires a defined process model. In contrast then to the mentioned notable composition approaches, we achieved advanced adaptability and quality-orientation (by allowing nonfunctional criteria in selections), but lacked where these approaches are strongest: in plan synthesis based on functional requirements.

The seemingly most attractive approach bears the feature set (B, D, F, H, J). First, both functional and nonfunctional requirements are taken as service selection criteria (B), so that we can account for quality of service considerations in AOSS. Second, very few assumptions need be made as to the expertise of the users specifying the service request, so that no predefined process need be assumed (D). Third, uncertainty makes it difficult to assume that services perform as advertised – it is better to base subsequent service selections on the observed prior performance of services than their advertised performances (F). Fourth, compositions are continually revised so that users always obtain optimal solutions to their requests (H). Finally, we assume that ontologies of functional and nonfunctional characteristics are only partly known, and need to be revised as services with new characteristics appear (J). Intuitively, this is a particularly attractive aim, which fits well the loan problem we outlined earlier. The loan institution would be able to specify requests rich in functional and nonfunctional criteria for service selection; it would not need to specify the detailed process to execute, leaving it instead to its internal or expternal services and the composer to plan; inside or outside partners would be selected (and compensated) based on observed and not advertised behavior; the set of involved parties would be constantly revised for optimal results; finally, the loan institution remains open to novel approaches to providing and managing loans. Achieving the (B, D, F, H, J) combination of features requires that the following problem be resolved.

Definition 17.1. *Given:*

- *a set* $\mathbf{W}(t)$ *of services available at time* t.
- *a set* $\mathbf{W}_\infty$ *of all possible services. At time* t*, we know only the distinct services from* $\mathbf{W}_\infty$ *that have been available up to* t.
- *a set* $\mathbf{C}(t)$ *of service composers available at time* t.
- *a set* $\mathbf{R}(t)$ *of service requests submitted to service composers at time* t.
- *a cumulative task domain ontology* $\mathcal{O}^{\boldsymbol{T}}(\bar{t})$ *that defines functionalities offered by all services that have been available up to and including time* t *(hence, cumulative). We call individual functionalities tasks.* $\bar{t}$ *is a sequence ending with, and including* t.
- *a full task ontology* $\mathcal{O}^{\boldsymbol{T}}_\infty$ *of all possible tasks that services can offer. At any time* t*, we only know the part* $\mathcal{O}^{\boldsymbol{T}}(\bar{t})$ *of* $\mathcal{O}^{\boldsymbol{T}}_\infty$.
- *a task advertising function* $\mathcal{M}^{\boldsymbol{T}} : \mathbf{W}_\infty \longrightarrow \wp\ddot{\mathcal{O}}^{\boldsymbol{T}}_\infty$ *which is a total function mapping each individual service to one or more tasks that the given service can execute. We use* $\ddot{\mathcal{O}}^{\boldsymbol{T}}_\infty$ *to denote the set of all distinct entities extracted from the ontology* $\mathcal{O}^{\boldsymbol{T}}_\infty$.
- *a cumulative quality domain ontology* $\mathcal{O}^{\boldsymbol{Q}}(\bar{t})$ *that defines nonfunctional characteristics of all services that have been available up to and including time* t*. We call individual nonfunctional characteristics qualities.*
- *a full quality ontology* $\mathcal{O}^{\boldsymbol{Q}}_\infty$ *of all possible qualities that can characterize all possible services. At any time* t*, we only know the part* $\mathcal{O}^{\boldsymbol{Q}}(\bar{t})$ *of* $\mathcal{O}^{\boldsymbol{Q}}_\infty$.
- *a quality advertising function* $\mathcal{M}^{\boldsymbol{Q}} : \mathbf{W}_\infty \longrightarrow \wp\ddot{\mathcal{O}}^{\boldsymbol{Q}}_\infty$ *which is a total function mapping individual services to sets of qualities from the cumulative nonfunctional domain ontology. We use* $\ddot{\mathcal{O}}^{\boldsymbol{Q}}_\infty$ *to denote the set of all distinct entities extracted from the ontology* $\mathcal{O}^{\boldsymbol{Q}}(\bar{t})$.

such that:

- *for each service* $\mathbf{w} \in \mathbf{W}(t)$*, functional* $\mathcal{M}^{\boldsymbol{T}}(\mathbf{w})$ *(i.e., tasks) and quality characteristics* $\mathcal{M}^{\boldsymbol{Q}}(\mathbf{w})$ *are advertised.*
- *each service request* $\mathbf{r} \in \mathbf{R}(t)$*,* $\mathbf{r} \equiv \langle \mathbf{r}^{\boldsymbol{T}}, \mathbf{r}^{\boldsymbol{Q}}, \mathbf{r}^{\boldsymbol{O}} \rangle$*, defines constraints* $\mathbf{r}^{\boldsymbol{T}}$ *on tasks from* $\mathcal{O}^{\boldsymbol{T}}(\bar{t})$*, constraints* $\mathbf{r}^{\boldsymbol{Q}}$ *on qualities from* $\mathcal{O}^{\boldsymbol{Q}}(\bar{t})$*, and identifies* $\mathbf{r}^{\boldsymbol{O}}$ *the qualities from* $\mathcal{O}^{\boldsymbol{Q}}(\bar{t})$ *to optimize.*
- *the cumulative task domain ontology is updated at each* t *to include tasks that are advertised for all* $\mathbf{W}(t)$.
- *the cumulative quality domain ontology is updated at each* t *to include all qualities that are advertised for all* $\mathbf{W}(t)$.

Find:

1. *a procedure to elicit and specify users' functional and nonfunctional requirements.*

2. *a procedure to transform the specification of users' requirements into service requests, i.e.,* $\mathbf{r}^{T}$, $\mathbf{r}^{Q}$, *and* $\mathbf{r}^{O}$ *for each service request* $\mathbf{r} \in \mathbf{R}(t)$.
3. *a procedure to update the specification of users' requirements to inform the users of the extent to which their prior requests have been satisfied and of the new requirements that the AOSS can satisfy.*
4. *a procedure* $\mathcal{P}$*, which given any particular service request* $\mathbf{r}$*, returns a sequence of tasks* $\mathcal{P}(\mathbf{r})$ *to execute in order to satisfy* $\mathbf{r}^{T}$.
5. *a procedure* $\mathcal{A}$*, which given* $\mathcal{P}(\mathbf{r})$ *and the available services* $\mathbf{W}(t)$*, returns an allocation* $\mathcal{A}(\mathcal{P}(\mathbf{r}), \mathbf{W}(t))$*. The allocation indicates which service in* $\mathbf{W}(t)$ *is to execute what task in* $\mathcal{P}(\mathbf{r})$ *in order to satisfy* $\mathbf{r}^{Q}$ *and optimize* $\mathbf{r}^{O}$.
6. *a procedure* $\mathcal{R}$ *which, at some subsequent time* t'*,* $\mathbf{W}(t') \neq \mathbf{W}(t)$*, explores alternative allocations and returns* $\mathcal{A}(\mathcal{P}(\mathbf{r}), \mathbf{W}(t'))$ *which optimizes* $\mathbf{r}^{O}$.

Issues 1 and 3 are problems of requirements engineering, that is, concern the elicitation, specification, and analysis of requirements. Issue 3 mainly concerns the relationship between requirements and capabilities that realize them – one way to look at this is through traces that can be established between requirements and system behaviors. Concerns 4–6 are problems of planning. Below, we assume that the given problem is to be resolved by a framework for the engineering of requirements, which incorporates a planning approach for service composition. The aim is to identify desiderata for such a framework, and thereby guide future work on developing such a framework. To simplify the discussion, we call such a future framework *Arete*.[3]

17.3.3 Shaping a Solution

Adaptability and openness in AOSS and the focus on enabling dynamic specification and strong quality-orientation determine the feature set of Arete. We first discuss how adaptability and openness in AOSS entails the need for dynamic specification, and explain why such specification cannot be performed in requirements engineering (RE) frameworks suited to closed and non-adaptable systems (§17.3.3). We then look into why quality orientation is relevant when specifying service requests, and identify framework features needed to account for quality considerations in service requests (§17.3.3).

Engineering AOSS Requirements

Service requests are not unlike requirements specifications in that they describe requirements about the behavior of a system. Arete will therefore draw some of its intuitions from requirements engineering (RE) frameworks. Goal-oriented RE (see, e.g., [315, 316] for overviews) is of particular relevance, for it uses rich concepts for representing and reasoning about users' intentions.

It is, unfortunately, not possible to reuse as-is the contributions in goal-oriented RE. Therein, frameworks are built with non-adaptable and closed systems in mind, so that the proposals do not immediately apply to AOSS. Namely, (i) we cannot directly apply such RE frameworks to AOSS because of a scope mismatch; (ii) available frameworks do not integrate features needed to perform continual updates of users' requirements at runtime; and (iii) they lack expressive power when representing and reasoning about quality. We discuss each of these three limitations below and outline responses thereto in the form of guidelines for the construction of service request specification frameworks.

Scope Mismatch.

Requirements specification consists of describing the stimuli that the future system may encounter in its operating environment and defining the system's responses according to the stakeholders' requirements. The more potential stimuli are anticipated and accounted for, the less likely a discrepancy between the expected and the observed behavior and quality of the system. Hence a longstanding concern in RE on requirements *completeness* (e.g., [333, 315]): i.e., ensuring that the requirements specification accounts for all the relevant stakeholder requirements and environment properties and behaviors.

To engineer as complete as possible a set of requirements for some system, any established RE framework proceeds as shown on plate a in Figure 29. First, the purpose of the system is described in terms of functional and nonfunctional goals. These are refined (i.e., made more precise by, e.g., decomposition and specialization). From a set of precise goals, alternative ways of achieving goals (i.e., operationalizations) are identified. They are compared using nonfunctional goals as criteria and the most appropriate

[3] From the Greek word ἀρετή (pronounced [ærətei]) which means excellence; reaching the highest potential.

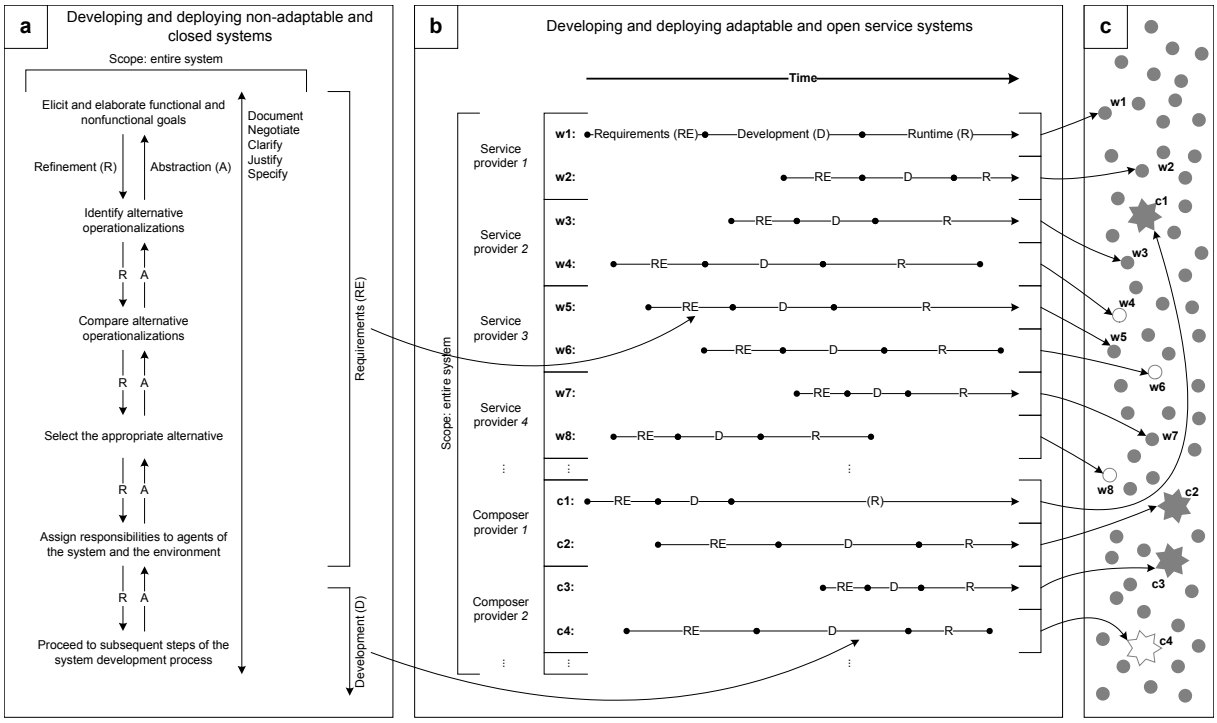


Fig. 29. The scope of RE that is appropriate for non-adaptable and closed systems does not apply for AOSS.

alternative is chosen. Finally, agents in the system and the environment are identified to achieve the precise requirements given in the chosen alternative. Note that this is not solely a top-down approach, as unanticipated lower-level concerns might give rise to higher-level considerations (e.g., some alternative operationalization can point to goals that are not explicit). We thus iterate back and forth between steps in order to obtain a sufficiently complete specification. We move down by refining the output obtained at the current step, while we move up by abstracting from outputs identified lower, which are without a correspondent higher in the chain. In parallel, activities such as the documentation, negotiation (e.g., [32]), clarification and justification (e.g., [25, 52, 153]), and specification (e.g., [242, 58, 327]) of requirements are performed.

It is unsurprising that the usual principles of problem solving and decision making are adopted in RE. It is, however, not this process that is questionable when engineering requirements for AOSS, but the *scope* of the process.

In an AOSS, services are developed by various service providers, and composers by various "composer providers". The infrastructure for AOSS is put in place by, e.g., standards bodies, industry consortia, and government initiatives. Users of AOSS are often external to service and composer providers. We cannot assume therefore that a unifying effort of all these parties can be performed under the approach described for non-adaptable and closed systems. In other words, if we treat an AOSS as a unique system and apply the standard RE approach, we would encounter, among others, severe coordination issues between the various parties. This is clearly inefficient.

This point is illustrated in plates `b` and `c` in Figure 29. Each provider proceeds to the requirements engineering, development, and deployment (`b`) separately for each service or composer that is then made available in the AOSS (`c`). Plate `b` in shows hypothetical timelines for the RE, development, and deployment phases for individual services and composers; when deployment is discontinued (e.g., **w4** in Figure 29) the service or composer is unavailable in the AOSS. We see that taking the entire AOSS within the scope of an RE framework would require the synchronization of various parties' efforts. It appears more realistic to reduce the scope to individual services and composers: the standard RE frameworks can then be applied locally by providers to individual services and composers, while proper infrastructure and the Semantic Web would ensure sufficient interoperability.

We see therefore that RE as usual can be appropriate, provided it is local to services and composers, and whatever other integrative parts of an AOSS. Otherwise, there is a scope mismatch between the scope of standard RE frameworks and the scope of AOSS.

To accept the presence of a scope mismatch is to acknowledge that the RE of AOSS proceeds in a different way than the RE of non-adaptable and closed systems. The RE of AOSS is distributed and

unlikely to be synchronized.[4] From there on, we see suggest that requirements are being engineered in an AOSS on four fronts:

- *Infrastructure RE* is the engineering of requirements about the infrastructure that enables the interaction between services, composers, and users (and any subset thereof). It is currently performed within bodies such as the World Wide Web Consortium (W3C), and more broadly, by the research community. The end user at this level are providers who deploy their applications using the available infrastructure.

  Example 17.2. As ontologies are key to enabling the sharing of machine-understandable information on the Web, the W3C presents requirements for web ontology languages [122]. These requirements highlight that, e.g., the language ought to allow the definition of complex data types, have efficient decision procedures, support different levels of complexity in ontology definition, and so on. Such requirements are not related to a particular user application domain.

- *Service RE* focuses on an individual service, and is performed at and by a service provider. Established RE frameworks, such as, e.g., KAOS [71, 197, 198] and Tropos [50, 100] appear applicable for the RE of an individual service. As the provider has apparent incentives to design the service so that it can participate in various systems, the salient characteristic of Service RE is that it should ensure the genericity of the given service. To achieve genericity, the the provider makes assumptions about users' requirements and subsequently either enables advanced parametrization of the service (i.e., to allow personalization) or favors modularity by considering only a well-delimited user requirement and building the service to cater that requirement alone. While the latter case places no strain on available RE frameworks, the former case can benefit from frameworks that carry specialized approaches to personalization (e.g., [200]).

  Example 17.3. Requirements that a service provider ought to account for include, among others, the requirements on communication and negotiation protocols and the ontology for describing its service's capabilities, behaviors, and qualities. The requirements on service's functionality will be shaped by what modular functionality the provider aims to offer to AOSS users; e.g., for an online travel booking service, a provider might be developing a service that allows flight booking for some airlines. Requirements in that case include a description of how the service interacts and changes the airlines' data on flight bookings, what data the service needs at input, what it outputs, along with nonfunctional considerations such as, e.g., most probable mean time for delivering the output to the composer. While closer to the AOSS user than infrastructure requirements, the requirements on an individual service remain generic to ensure that the service can be used in various AOSS. This does not restrict the provider in engineering requirements so that implemented functionality allows service personalization.

- *Composer RE* concerns the requirements on properties and behaviors of composers, and is performed at, and by composer providers. A salient characteristic of Composer RE is that it is concerned with enabling adaptability through the selection of procedures for composer behavior. In this respect, user requirements need be accounted for to the extent that they condition the choice of composer behavior.

  Example 17.4. In online travel booking, requirements on the composer focus on choosing the behavior that will allow the composer to interact with various services, explore as fast as feasible the various potential composition options, and communicate with various services. Such considerations were driven by the assumption that users expect "best" possible quality from the system, which can only be achieved if appropriate compositions are identified and revised as new services appear and previously used become unavailable.

- *User RE* concerns the requirements that users provide to composers. User RE has two salient characteristics:
  - *User RE is performed at runtime.* User RE deals with the engineering of requirements that end up in service requests, which in turn are provided to the system at runtime. Automation of the various steps in such the User RE process is thus of particular interest to facilitate and speed up the conversion of users' requirements into a format interpretable by the AOSS.

[4] One way in which synchronization could be achieved is that composer providers define requirements on individual services which are subsequently used by service providers willing to create services specialized for the composers. Scope mismatch would in that case be mitigated. Such synchronization is, however, unlikely since a provider normally develops a service in the aim of leaving it open to many different composers. Current industrial practices support this claim (see, e.g., [204, 136]).

– *User RE stops earlier than usual RE.* An adaptable system is usually needed when the environment is unpredictable – the requirements engineer thus cannot be expected to anticipate all possible environment and system properties and behaviors. A consequence is that it is inappropriate to extensively specify users' requirements at runtime: placing extensive constraints at the outset limits openness and adaptability. That is, it limits the potential for satisfying requirements at runtime in ways different than those anticipated and allowed during development specification. Strong restrictions on service behavior will block such unanticipated yet relevant service behaviors. User RE thus stops earlier than usual RE because ways in which users' requirements can be satisfied vary at runtime.

Example 17.5. The earlier travel booking requirements are examples of requirements that appear and are of interest at User RE. They express individual users' requrements of the system at runtime, and need to be made operational in much shorter time spans than requirements relevant for Infrastructure RE, Service RE, and Composer RE.

The following guidelines for the design of frameworks for the specification of service requests for AOSS summarize the above discussion:

Guideline 1. *Separate concerns.* Due to the scope mismatch, RE for AOSS proceeds separately for AOSS infrastructure (Infrastructure RE), individual services (Service RE), individual composers (Composer RE), and end users (User RE). In contrast to Infrastructure, Service, and Composer RE, User RE takes place at AOSS runtime. The output of User RE are service requests in a format that composers can interpret. A framework for specifying service requests ought to cover User RE.

Guideline 2. *Automate operationalization.* As User RE takes place at AOSS runtime, the process applied during User RE ought to be automated as much as feasible to facilitate and speed up the conversion of users' requirements into a format interpretable by the composers in the AOSS.

Guideline 3. *Do not over-specify.* The usual RE process (plate a in Figure 29) may be applicable locally but separately for the infrastructure, individual services, and individual service composers of AOSS. Notwithstanding such separation, shared ontologies and infrastructure ensure interoperability between the various systems produced by separate efforts. In contrast, User RE can adopt initial steps of the said RE process, but not the entire process. A framework for specifying service requests stops at most just before the assignment of responsibilities for requirements to services since such assignment is performed by the automated composer.

Variation of Requirements.

Having established that RE proceeds separately for the infrastructure, services, composers, and users, we now argue that the variation frequency is highest for requirements elicited at User RE, so that dynamic specification is needed. Requirements on the infrastructure vary, but are likely to be subjected to thorough discussion before operationalization and deployment. Requirements on services or composers vary certainly more often than those on the infrastructure, though their operationalization involves the revision of individual services or composers probably along the usual RE process and subsequent development stages. The point is that the time from the identification of a new requirement or a revision of an old one to the satisfaction of that requirement is long relative to the time from the arrival of a service request to its expected fulfillment. The more users, the higher the frequency at which requirements at User RE vary. Two kinds of variation are present in requirements at User RE:

- *Ascending variation* is driven by the adaptability of AOSS and change in the pool of candidate services. Consider plate a in Figure 30: a service request is sent to a composer in an AOSS, the composer identifies the appropriate services and executes the request. We show the result of request execution as the initial process graph with labels indicating how requirements are satisfied during execution and in the output. Differences between requirements in the service request and in the output of the request can arise for different reasons: e.g., initial requirements may have been idealistic, or some services unavailable (so that "second-best" services were used in the composition). Ascending variation thus amounts to divergence in ways in, and degrees to which requirements laid out in a service request are fulfilled by a composer.

  Example 17.6. Ascending variation in a user's requirement to book a particular flight may take two forms. If the service request states user's seating preferences and these have not been fulfilled by the

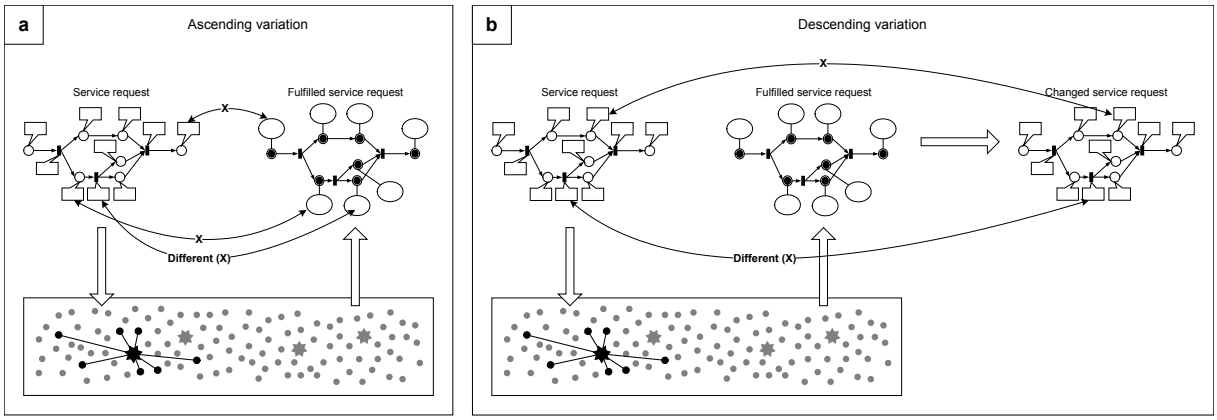


Fig. 30. Ascending and descending variation in user requirements.

composer, one kind of ascending variation is to update the requirements specification that resulted in the service request, so that the specification reflects that the requirements are not fulfilled properly. Another kind of variation arises from the possibility that the services participating in the chosen composition offer options in addition to those considered by the user in her requirements. For instance, the user might have specified only seating preferences, although the participating services offer the possibility to define entertainment preferences as well. The specification of user's requirements can then be updated to reflect this possibility.

- *Descending variation* is due to users' adaptability to observed AOSS behavior. Over time, users learn about the problem they expect the AOSS to resolve and about how the AOSS responds to their requests. As they learn, and as their environment changes, users' requirements change as well. In plate b in Figure 30, a request is sent to the AOSS, a composer handles the request, and returns an output. Having observed the output (which may or may not correspond to requirements in the initial service request), the user might change the initial service request before submitting it to the AOSS again. Descending variation amounts to change in requirements that users expect the system to fulfill.

  Example 17.7. Following Example 17.6, after the requirements are updated to reflect new possibilities (above, entertainment preferences), the user might include these additional considerations in her requirements for subsequent service requests for flight booking.

A consequence of variation is that users' requirements need not only be made operational and achieved in the usual top-down manner, but also updated bottom-up, to provide feedback (what we called the "Fulfilled service request" in Figure 30) to the user regarding whether, how, and to what extent her requirements are achieved. Such feedback helps the user revise future requirements on the AOSS. We can thus speak of *dynamic* specification, that is, one which continually changes because of ascending and descending variation in requirements. Dynamism of a requirements specification can be achieved with *ascending updates*, executed in response to ascending variation, and *descending updates* performed to adjust the specification to variation in stakeholders' requirements (i.e., in response to descending variation). We thus arrive at the following guideline:

Guideline 4. *Update user requirements.* Due to ascending and descending variation in user requirements for an AOSS, a framework for the specification of service requests needs to include concepts and techniques for updating the requirements obtained through User RE so that these requirements remain current with changes in users' requirements and changes of ways in, and degrees to which the system fulfills service requests.

Updating the user requirements specification continually with newly relevant requirements (i.e., descending update) is a way of addressing an important difficulty encountered in RE research – that of variation of users requirements over time, which leads to an increasing divergence between what the system does (in accordance with initial requirments obtained at development time) and what it is expected to do (in accordance with new/revised requirements). When engineering a non-adaptable and closed system, an initial set of requirements is identified and the system is subsequently built and deployed to achieve the given set of requirements. After deployment, as the environment of the system continues to change (being changed in part by the system), so do the requirements that users have on the system.

This interplay between the environment, the system, and its users (or more broadly, users and other system stakeholders) makes it difficult for the users to agree on a set of requirements on the system and to subsequently remain with that decision.[5] Initial requirements become irrelevant over time and the system inappropriate w.r.t. intended use. In such complex settings, mutual adjustment between the system and its users is needed, and can be facilitated by an exchange of information between the given parties. Ascending and descending updates of the requirements specification obtained in User RE enable such exchange of information.

Before we proceed, note that enabling ascending updates has a consequence that may not be evident from the above discussion, though hinted at in Example 17.7. Namely, ascending updates can result in new user requirements and not only the revision of prior ones. Recall that, in the problem definition (§17.3.2) we highlighted the need for cumulative task and quality ontologies (i.e., $\mathcal{O}^{\mathbf{T}}(t)$ and $\mathcal{O}^{\mathbf{Q}}(t)$). The need for cumulative ontologies is one response (seemingly appropriate for AOSS on the Semantic Web) to the ascending variation problem. Appearance of new services, with the ability to perform new tasks and/or being described with new qualities result in ascending update of the cumulative ontologies. If users are informed about the new functionalities, they can specify requirements that exploit the said functionalities. That new user requirements appear as the result of new capabilities of a system is certainly not novel[6] – ascending updates enable us to explicitly deal with them in User RE.

Expressing Quality Requirements.

We pointed above to two difficulties that established RE frameworks encounter when applied to AOSS. In addition to the peculiar scope of the RE for AOSS and the need for dynamic requirements specification, we argue that conceptualizations used to represent and reason about quality (i.e., nonfunctional) requirements are not expressive enough for use within User RE. We have already discussed in Part I how the established concepts lack in expressivity and how they can be usefully extended.

Quality-Orientation

Quality management for AOSS amounts to continually taking action to conform to users' specifications of requirements. With descending updates, we can also hope to exceed users' requirements by allowing them to benefit from unanticipated yet available system functionality. Regarding how quality is conceptualized in Arete, we drew many of the intuitions from long traditions of quality research in management science and software engineering. Lack of a shared understanding of what quality is (see, e.g., [270, 176] makes it difficult to determine an appropriate feature set for dealing with quality requirements in AOSS. Instead of defining the slippery concept of quality, it is through work on contributions outlined in Chapters 11 and 12 that we considered plausible AOSS use cases which carry considerations referring to abstract notions of security, safety, performance, personalization, convenience, and similar. These cases lead to the following observations and premises that shape Arete's approach to quality:

1. While quality has no unique conceptualization, pragmatic conceptualizations are desirable.
2. Quality is a subjective experience. Users' evaluation of quality depends on expectations and perceptions.
3. Quality involves trade-offs.

Metrics.

Variety of quality definitions indicates that it is restrictive to limit a quality-oriented framework for the specification of service requests to a single conceptualization of quality. However, this does not entail that any conceptualization is acceptable. Namely, it is well known that one cannot manage what one cannot measure.[7] This quantitative, pragmatic approach to quality is acknowledged in research on quality

[5] Literature on decision making in management science notes that (see, e.g., [210] and related) both individual and organizational intentions tend to suffer from problems of relevance, priority, clarity, coherence, and stability over time, all of which combine with the variation, inconsistency, and imprecision, among other, of decision makers' (here, users') preferences.

[6] Users commonly experience this when new (versions of) software are introduced: software vendors anticipate future needs, implement means to fulfill them, and advertise these to users.

[7] This business proverb has a correspondent in standards documentation. For instance, the British Standard (BS) 4778 (cit. in [277]) states: "In order to be able to assure, control or improve quality, it is necessary to be able to evaluate it".

ontologies for the Semantic Web, where metrics play a key role (see, e.g., [69, 171, 343, 217, 340]). They are a significant source of information needed for estimation, prediction, assessment, and benchmarking of quality attributes identified in a quality model, and of effort and cost (to be) invested in the various phases of the software lifecycle. Since providers advertise the quality of their services, they are likely to base their advertised quality levels on metrics defined over measurable services' properties and behaviors. Hence the following guideline:

Guideline 5. *Accommodate measured quality.* A quality-oriented framework for the specification of service requests ought to ensure that users' quality requirements are expressed in a format understandable to composers. Otherwise, the composer cannot use the quality requirements when deciding on which services participate in fulfilling service requests. This provider-oriented conceptualization of quality (i.e., $\mathcal{O}_{\infty}^{\mathfrak{Q}}$ in the definition of the problem that Arete aims to resolve) results in the need for the definition of mappings from users' quality requirements towards metrics and values thereon.

The result of the above guideline is that the quality domain ontology $\mathcal{O}_{\infty}^{\mathfrak{Q}}$ involves metrics that measure services' behaviors. Evaluating quality through measurement on the system contrasts to quality evaluation performed by individual users who may and are likely to be using criteria different from providers. Namely, there are two perspectives on quality, one outlined above, the other, user-specific one is discussed below. Both need to be accommodated and interrelated when dealing with user requirements in service requests. The use of our core ontology introduced in Part I is clearly relevant in defining a framework such as Arete, since it incorporates the quality constraints that define constraints on qualities, of which some can be defined as metrics, while including more abstract notions needed in eliciting and organizing stakeholders' expectations on quality.

Perceptions and Expectations.

The principal aim of representing and reasoning about quality is to facilitate the construction and running of systems capable of meeting and exceeding user's expectations. The quantitative, metric-based approach to quality enables a characterization of the system in terms of its measurable properties and behaviors. Within such a perspective, the system's designer can argue that the system achieves some levels of quality according to the designer's quality conceptualization or one adopted from international standards. This, however, gives no guarantee that the system's users will share the same perception on the system's quality. Indeed, it is acknowledged in management (e.g., [270, 236, 247, 248]) and software engineering research (e.g., [176]) that quality is a subjective experience: evaluations of quality will vary among users and will not necessarily correspond to evaluations based on metrics over measurable properties and behaviors of the system.[8]

A framework for User RE needs to accommodate both perspectives: that of the providers, manifest in one (i.e., $\mathcal{O}_{\infty}^{\mathfrak{Q}}$) or more quality ontologies (each aligned to $\mathcal{O}_{\infty}^{\mathfrak{Q}}$) grounded in metrics on services, and that of the users who experience quality in a subjective manner, and describe it in their own terminology. It is then apparent that a User RE framework ought to facilitate the definition of mappings between services' quality ontologies and users' descriptions of experienced quality. To enable this, we need to clarify what it means to state that quality is a subjective experience.

Marketing research that focuses on service quality highlights the importance of expectations for customers' evaluation of quality. Seminal contributions (e.g., [247]) suggest that a customer's subjective evaluation of quality is related to the gap between what she expects and what she perceives at service delivery (e.g., [111, 247]). Following this line of thinking, a recent definition suggests that service quality is "the degree and direction of discrepancy between customers' service perceptions and expectations" [248]. To better understand how consumers evaluate service quality, marketing scholars have adopted an approach similar to that taken in software quality modeling. Namely, sets of factors determining perceived service quality have been suggested, not unlike quality attributes in software quality conceptualizations.[9]

[8] That quality is a subjective experience is clear in industry. For instance, one of the criteria for U.S. Department of Commerce's award for outstanding quality states:

> "Quality is judged by the customer. All product and service attributes that contribute value to the customer and lead to customer satisfaction and preference must be the foundation of a company's value system. Value, satisfaction, and preference may be influenced by many factors throughout the customer's overall purchase, ownership, and service experiences." [234]

[9] The prominent set of determinants [247] includes: knowledge and curtesy of employees delivering the service and their ability to inspire trust and confidence (termed: assurance), caring and individualized attention providede

The common approach to measure perceived service quality is to make quality determinants measurable through questionnaires and seek trends in data based on postulated models. Models that proved relevant have an appeal in quality management for they can be used in predicting quality perceptions and therefore help in improving quality. We are careful not to transpose directly the marketing models in the present discussion for two reasons. First, the concept of service as used in marketing differs from that employed in service-oriented computing. Marketing literature contrasts the concept of service to that of product – among various distinguishing traits, the former is intangible, how it is delivered varies, and typically is tailored to a particular customer. One significant distinctive characteristic of a (web, or semantic web, or AOSS) service is that it involves significant degree of automation in delivery, so that, e.g., variability in delivery is unlikely to be as significant as that which occurs when a service is delivered by a human (as is usually understood in marketing). Second, we have noted earlier that general sets of quality attributes are difficult to accept when modeling software quality in the metric-based approach. As in software quality modeling, the generality of the determinants of perceived service quality has been questioned in marketing research (e.g., [47, 45]). As a result, we cannot operationalize perceived quality for AOSS services in the same way as for predominantly human-delivered services treated in marketing.

While there are limits to what we can learn from marketing, it is reasonable to accept that subjective experience of quality depends on user's expectations on and perceptions of service fulfillment. Indeed, in both AOSS and traditional service fulfillment, a customer/user has expectations about how the service request is to be fulfilled; she then perceives the result, and evaluates fulfillment. If we are to manage the quality perceived by the users in AOSS, we thus also face the problem of understanding how expectations and perceptions relate to provide an overall assessment of quality that affects the users' subsequent use of the AOSS. Among the various models (see, e.g., [38, 248] for further references) of consumers' assessment of service quality, dynamic models are interesting for the present discussion for they underline the relevance of prior expectations and perceptions for subsequent experience of quality. Operationalization of such models in AOSS would allow us to predict users' evaluations of service quality, and subsequently use predictions to improve perceived AOSS quality.

Expectations and perceptions are usually related by the disconfirmation of expectations paradigm [236]: in the present terminology, predictions that users make in advance of service delivery act as a standard against which they evaluate not only the service but the AOSS in its whole (or part they repeatedly interact with). User's evaluation of overall AOSS quality then results from a comparison between expectations and perceptions of service characteristics that affect the criteria over which the user judges the service. Following various models grounded in the said paradigm (see, e.g., [34, 237] for models and [96] for a discussion), users are likely to have three kinds of expectations about how their service request is fulfilled: predictive expectations on how the request *will* be fulfilled (denoted WE below); normative expectations about how the request *should* be fulfilled (SE); and expectations about how the request should ideally be fulfilled (IE). Ideal differ from should expectations: the former are not necessarily realistic or feasible, and are usually assumed more stable than SE. Regarding the formation of expectations and perceptions, it is postulated [34] that user i's expectations of how his current k-th submission of service request p will be fulfilled over i's j-th criterion depends on WE_{ipjk-1} formed at the previous request fulfillment, a vector of information that the user receives about the AOSS or some part thereof between the previous and current request ($\mathbf{I}^{\mathsf{WE}}_{ik}$), and the perception of how the current request is fulfilled (PS^{*}_{ipjk}).[10] If the user evaluates how the request is fulfilled across J_i subjective (i.e., proper to that user) criteria, then the following summarizes the postulate on the formation of WE:

$$\mathsf{WE}_{ipjk} = f\left(\mathsf{WE}_{ipjk-1}, \mathbf{I}^{\mathsf{WE}}_{ik}, \mathsf{PS}^{*}_{ipjk}\right) \tag{11}$$

Similarly, SE_{ipjk} will depend on previous normative expectations SE_{ipjk-1}, information ($\mathbf{I}^{\mathsf{SE}}_{ik}$) obtained after the last service fulfillment (e.g., a price change might raise normative expectations), and the perception of how the current request is fulfilled (PS^{*}_{ipjk}).

$$\mathsf{SE}_{ipjk} = f\left(\mathsf{SE}_{ipjk-1}, \mathbf{I}^{\mathsf{SE}}_{ipk}, \mathsf{PS}^{*}_{ipjk}\right) \tag{12}$$

---

to the customer (empathy), ability to perform the promised service dependably and accurately (reliability), willingness to help customers and provide prompt service (responsiveness), and appearance of physical facilities, equipment, personnel, and communications material (tangibles).

[10] We refer to some particular service request (i.e., p), since we allow the requirements in a service request to change and the user to submit more than one distinct service request. The asterisk in PS^{*}_{ipjk} is to separate cumulative perception from one-time perception.

From there on, a user's perception PS_{ipjk} accumulated up to the k-th request of how j-th criterion is satisfied, is updated at each request fulfillment, depending on user's expectations and available information just prior to the request:

$$\mathsf{PS}_{ipjk} = f\left(\mathsf{WE}_{ipjk-1}, \mathsf{I}^{\mathsf{WE}}_{ik}, \mathsf{SE}_{ipjk-1}, \mathsf{I}^{\mathsf{SE}}_{ik}, \mathsf{PS}^{*}_{ipjk}\right) \tag{13}$$

Overall quality the user i's experiences regarding a service request p repeatedly submitted to an AOSS (OQ_{ipk}) is a function over that user's J_i subjective dimensions:

$$\mathsf{OQ}_{ipk} = f\left(\mathsf{PS}_{ipjk}\right) \tag{14}$$

The above and comparable models have been empirically tested and is accepted in marketing research. As we need to integrate both the providers' and users' quality conceptualizations, we require the following of frameworks such as Arete:

Guideline 6. *Accommodate users' subjective evaluations of quality.* A quality-oriented framework for User RE that aims to incorporate users' subjective evaluations of qualiy ought to incorporate concepts and techniques for the elicitation of, representation of, and reasoning about expectations, perceptions, and information that can affect perceptions and expectations, and subsequently, users' subjective evaluation of quality.

One possible approach to realizing the above guideline is to adopt the model outlined in Equations 11–14 (or some derived variant thereof[11]), then determine what concepts, constructs, and techniques may be appropriate for representing and reasoning about WE, SE, IE, J_i, $\mathsf{I}^{\mathsf{WE}}_{ik}$, $\mathsf{I}^{\mathsf{SE}}_{ik}$, and PS^{*}_{ipjk}, along with the various relationships between these variables. If the framework designer follows both Guidelines 5 and 6, it will be necessary to provide means for precisely relating providers' measurements that quantify quality to descriptions of users' subjective experience of quality. That is, if the framework allows the definition of a set of metrics, then the framework needs to integrate techniques for defining mappings between the values of the various metrics and of concepts and constructs used to represent and reason about users' subjective evaluation of quality. This is necessary for it is used in determining what behaviors are desirable more than others from the users' perspective.

Guideline 7. *Map measured to subjectively evaluated quality.* A quality-oriented framework for User RE that aims to incorporate providers' perspective on quality and users' subjective evaluations of qualiy, and enable the combined use of both for quality management ought to include techniques for the definition of mappings between the representations of measured quality and of users' subjective evaluations of quality. The purpose of the mappings is to understand what measured AOSS's properties and behaviors are desirable for what users.

If we revisit the Arete problem (§17.3.2) in light of the Guideline 6, we require an ontology for subjective user quality evaluation in addition to the cumulative qualities ontology $\mathcal{O}^{\mathsf{Q}}(t)$. Guideline 7 highlights that mappings need to be established between the two ontologies if they are to be used together.

Trade-offs.

A requirement on an individual quality characteristic of some services is idealistic if the required level of quality can never be achieved by the system. A requirement on a set of distinct quality characteristics can be idealistic because individual requiremements in the set are idealistic or because the quality levels required over the given characteristics can never be achieved simultaneously. This last case involves tradeoffs: it is a setting in which it is necessary to (i) know how various quality requirements are interdependent and (ii) decide on their relative priority (equivalently: importance or desirability) (e.g., [170, 169, 168]). Information about relative priorites originates from users and is relevant for composers for it is used to deal with tradeoffs at runtime.

Guideline 8. *Accommodate interdependencies.* A quality-oriented framework for User RE ought to integrate concepts and constructs for the representation of interdependencies (and strengths thereof) between (a) metrics whose values are interdependent (i.e., correlated) and which are used to measure quality; and (b) users' subjective quality evaluation criteria mapped (see, Guideline 7) to the interdependent metrics.

[11] Different taxonomies of expectations have been suggested and different additional variables used to understand how perceptions and expectations are formed. The cited model has the benefits of being clear, dynamic, individual-specific, and extensible, while relying on the predominant paradigm (that of disconfirmation of expectations) in marketing research on service quality experience. Empirical research on the use of various models in AOSS will determine what variant of the model is the most appropriate for what (classes of) AOSS.

Example 17.8. If we observe in 80% of service requests (i.e., system runs) that the time to show the output to the user doubles if the number of simultaneous requests is in some range, then the framework ought to enable the recording of this information and its subsequent use in, e.g., anticipating output delivery times in service requests.

In terms of the Arete problem, the above means that we must be able to establish how values of metrics in the cumulative qualities ontology are interdependent at system runtime, and then establish interdependencies between corresponding quality criteria that users employ in evaluating AOSS quality (see, Guidelines 6 and 7).

Guideline 9. *Accommodate priorities.* A quality-oriented framework for User RE ought to integrate concepts, constructs, and techniques for the elicitation of, representation of, and reasoning about priorities between interdependent (a) metrics and (b) users' subjective quality evaluation criteria mapped to the interdependent metrics

Example 17.9. Let the user submit a service request, in which she indicates a bound on delivery time. Assume further that the number of service requests to fulfill simultaneously at time of submission is in the interval mentioned in Example 17.8, so that anticipated delivery time is above the one expected by the user. However, if the user pays more, the system can treat the user's request before (some of) the other requests and respect the delivery time constraint. This is a situation where the user ought to express priority between maintaining cost and delivery time. The framework would enable this information to be recorded and reused subsequently if the user is facing the same tradeoff and does not change her priorities.

Interdependencies are needed to identify when trade-offs need to be performed. Once a trade-off is identified, priorities need to be elicited between the qualities confronted in the trade-off. A priority between two qualities indicates which one to optimize at the detriment of the other.

17.4 Concluding Remarks

The overall motivating problem that drove to, and that unites the various contributions presented in this thesis is how to better inform decision making and guide it towards decisions that will lead to the engineering of high quality information systems. Topics in two key related issues are therefore addressed. The first part of this thesis focuses on conceptualizations that facilitate the identification of relevant information and its organization for subsequent analysis. Namely, the first part revisits the terminology used in the engineering of requirements, which is the first step in the engineering of an information system; the revised terminology leads to the revision of the established formulation of the core problem in the field of requirements engineering. The second part of the thesis addresses a recurrent problem in any decision making approach that aims for rigor and structure. The problem of comparing and evaluating alternative decisions is considered, when both qualitative and quantitative information, written in an informal and/or formal (possibly mathematical) notation are available and need to be accounted for together when drawing conclusions about alternatives and their relative value. Drawing from artificial intelligence, philosophy, and linguistics, nonmonotonic reasoning is put to use, and in particular argumentation and justification are combined with techniques for "clarifying" information. The third part of the thesis illustrates how the suggested conceptualizations and techniques are applied in the enigineering of information systems, including those that rely on heterogenous and distributed components, as in service-oriented and agent-oriented computing. Contributions in the thesis open up an important new direction for future work. Namely, it is necessary to work on the definition of novel frameworks for the engineering of requirements at runtime for adaptable and open service-oriented systems, as available ones do not appear to be satisfactory. This thesis provides conceptual foundations and techniques that are likely to be an integral part of such frameworks.

References

1. Sherief Abdallah and Victor Lesser. Modeling task allocation using a decision theoretic model. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 719–726, New York, NY, USA, 2005. ACM Press.
2. Sherief Abdallah and Victor Lesser. Learning the task allocation game. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 850–857, New York, NY, USA, 2006. ACM Press.
3. Y. Achbany, F. Fouss, L. Yen, A. Pirotte, and M. Saerens. Tuning continual exploration in reinforcement learning. Technical report, Université de Louvain, Information Systems Research Unit (ISYS) (http://www.isys.ucl.ac.be/staff/francois/Articles/Achbany2005a.pdf), 2005.
4. Y. Achbany, F. Fouss, L. Yen, A. Pirotte, and M. Saerens. Optimal tuning of continual online exploration in reinforcement learning. In *Proceedings of the International Conference on Artificial Neural Networks*, 2006.
5. Yoji Akao, editor. *Quality Function Deployment: Integrating Customer Requirements into Product Design*. Productivity Press Inc, 1990.
6. Tariq Al-Naeem, Ian Gorton, Muhammed Ali Babar, Fethi Rabhi, and Boualem Benatallah. A quality-driven systematic approach for architecting distributed software applications. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 244–253, New York, NY, USA, 2005. ACM Press.
7. R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. In *Proceedings of the Tenth Annual Symposium on Principles of Distributed Computing*, pages 139–152. ACM Press, 1991.
8. C. Alves, X. French, J. P. Carvallo, and A. Finkelstein. Using goals and quality models to support the matching analysis during cots selection. In *Proceedings of the International Conference on COTS-Based Software Systems*, 2005.
9. Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business process execution language for web services version 1.1. Technical report, BEA, IBM, Microsoft, SAP AG and Siebel Systems, 2003.
10. A. Anton, J. Earp, and A. Reese. Analyzing website privacy requirements using a privacy goal taxonomy. In *Proceedings of the IEEE International Conference on Requirements Engineering (RE'02)*, 2002.
11. Annie I. Anton. Goal-based requirements analysis. In *Proceedings of the International Conference on Requirements Engineering*, 1996.
12. G. Antoniol, G. Canfora, and A. de Lucia. Maintaining traceability during object-oriented software evolution: A case study. In *ICSM '99: Proceedings of the IEEE International Conference on Software Maintenance*, page 211, Washington, DC, USA, 1999. IEEE Computer Society.
13. ATLAS.ti Scientific Software Development GmbH. ATLAS.ti - The Knowledge Workbench (http://www.atlasti.com). Technical report, ATLAS.ti Scientific Software Development GmbH, 2007.
14. Earl R. Babbie. *Survey Research Methods*. Balmont, CA: Wadsworth Pub. Co., 1973.
15. K. Bach. Ambiguity. In *Routledge Encyclopedia of Philosophy Online*. Routledge, 1998.
16. J. E. Bailey and S. W. Pearson. Development of a tool for measuring and analyzing computer user satisfaction. *Management Science*, 29(5):519–529, 1983.
17. C. Barker. Vagueness. In *Encyclopedia of Language and Linguistics*. Elsevier, 2006.
18. Victor R. Basili and H. Dieter Rombach. The tame project: Towards improvement-oriented software environments. *IEEE Trans. Software Eng.*, 14(6):758–773, 1988.
19. Victor R. Basili and David M. Weiss. A methodology for collecting valid software engineering data. *IEEE Trans. Software Eng.*, 10(6):728–738, 1984.
20. Richard Baskerville, Marco Cavallari, Kristian Hjort-Madsen, Jan Pries-Heje, Maddalena Sorrentino, and Francesco Virili. Extensible architectures: The strategic value of service-oriented architecture in banking. In *ECIS*, 2005.
21. J. Bather. *Decision Theory: An Introduction to Dynamic Programming and Sequential Decisions*. Wiley, 2000.

22. B. Benatallah, M.-S. Hacid, A. Leger, C. Rey, and F. Toumani. On automating web services discovery. *VLDB Journal*, 14:84–96, 2005.
23. B. Bennett. Modal semantics for knowledge dealing with vague concepts. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, 1998.
24. D. M. Berry, B. H. Cheng, and J. Zhang. The four levels of requirements engineering for and in dynamic adaptive systems. In *Proceedings of the International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'05)*, 2005.
25. Daniel M. Berry and Erik Kamsties. The syntactically dangerous all and plural in specifications. *IEEE Softw.*, 22(1):55–57, 2005.
26. D. P. Bersekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2000.
27. Philippe Besnard and Anthony Hunter. A logic-based theory of deductive arguments. *Artif. Intell.*, 128(1-2):203–235, 2001.
28. G. Y. Bizer, J. C. Barden, and R. E. Petty. Attitudes. In *Encyclopedia of Cognitive Science*. MacMillan, 2003.
29. B. W. Boehm, P. Bose, E. Horowitz, and Ming June Lee. Software requirements negotiation and renegotiation aids: A theory-w based spiral approach. In *Proceedings of the 17th International Conference on Software Engineering*, 1995.
30. B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. J. Macleod, and M. J. Merrit. *Characteristics of Software Quality*. North-Holland, 1978.
31. B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative evaluation of software quality. In *ICSE '76: Proceedings of the 2nd international conference on Software engineering*, pages 592–605, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.
32. Barry Boehm, Prasanta Bose, Ellis Horowitz, and Ming-June Lee. Software requirements as negotiated win conditions. In *Proceedings of the International Requirements Engineering Conference (RE'94)*, 1994.
33. A. Bondarenko, P. Dung, R. Kowalski, and F. Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93:63–101, 1997.
34. William Boulding, Ajay Kalra, Richard Staelin, and Valarie Zeithaml. A dynamic process model of service quality: From expectations to behavioral intentions. *Journal of Marketing Research*, 30:7–27, 1993.
35. J. P. Bowen and M. G. Hinchey. *High-Integrity System Specification and Design*. Springer-Verlag FACIT Series, 1999.
36. Jonathan P. Bowen and Michael G. Hinchey. Ten commandments revisited: a ten-year perspective on the industrial application of formal methods. In *FMICS '05: Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, pages 8–16, New York, NY, USA, 2005. ACM Press.
37. G. E. P. Box and W. G. Hunter. *Statistics for Experimenters*. New York: John Wiley and Sons, 1978.
38. Michael K. Brady, Gary A. Knight, J. Joseph Cronin Jr, G. Thomas, M. Hult, and Bruce D. Keillor. Removing the contextual lens: A multinational, multi-setting comparison of service evaluation models. *Journal of Retailing*, 81(3):215–230, 2005.
39. Ronen I. Brafman, Carmel Domshlak, and Solomon Eyal Shimony. On graphical modeling of preference and importance. *J. Artif. Intell. Res. (JAIR)*, 25:389–424, 2006.
40. M. E. Bratman, D. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.
41. G. Brewka. A reconstruction of rescher's theory of formal disputation based on default logic. In *Proceedings of the 11th European Conference on Artificial Intelligence*, 1994.
42. L. C. Briand, Y. Labiche, and T. Yue. Automated traceability analysis for uml model refinements (tr sce-06-06). Technical report, Carleton University, 2006.
43. Lionel C. Briand, Sandro Morasca, and Victor R. Basili. An operational process for goal-driven definition of measures. *IEEE Trans. Softw. Eng.*, 28(12):1106–1125, 2002.
44. G. Brown, B. H. C. Cheng, H. Goldsby, and J. Zhang. Goal-oriented specification of adaptation semantics in adaptive systems. In *Proceedings of the International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'06)*, 2006.
45. F. Buttle. Servqual: Review, critique, research agenda. *European Journal of Marketing*, 30(1):8–32, 1996.
46. nevar Carlos Iván Ches Ana Gabriela Maguitman, and Ronald Prescott Loui. Logical models of argument. *ACM Comput. Surv.*, 32(4):337–383, 2000.
47. J. M. Carman. Consumer perceptions of service quality: An assessment of the servqual dimensions. *Journal of Retailing*, 56(3):55–68, 1990.
48. Edward G. Carmines and Richard A. Zeller. *Reliability and Validity Assessment*. Sage Publications, 1979.
49. Fabio Casati, Ming-Chien Shan, and Dimitrios Georgakopoulos. Guest editorial. *The VLDB Journal*, 10(1):1–1, 2001.
50. Jaelson Castro, Manuel Kolp, and John Mylopoulos. Towards requirements-driven information systems engineering: the tropos project. *Information Systems*, 27(6):365–389, 2002.
51. V. Cavalli-Sforza and D. D. Suthers. Belvedere: an environment for practicing scientific argumentation. In *Proceedings of the Workshop on Computational Dialectics*, 1994.
52. Francis Chantree, Bashar Nuseibeh, Anne de Roeck, and Alistair Willis. Identifying nocuous ambiguities in natural language requirements. In *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, pages 56–65, Washington, DC, USA, 2006. IEEE Computer Society.

53. C. I. Chesñevar, A. G. Maguitman, and R. P. Loui. Logical models of argument. *ACM Comput. Surv.*, 32(4), 2000.
54. N. Christofides. *Graph Theory: An Algorithmic Approach.* Academic Press, 1975.
55. Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering.* Kluwer Academic Publishers, 1999.
56. P. Cimiano, A. Eberhart, P. Hitzler, D. Oberle, S. Staab, and R. Studer. The smartweb foundational ontology. Technical report, SmartWeb Project, 2004.
57. Philipp Cimiano, Andreas Eberhart, Pascal Hitzler, Daniel Oberle, Steffen Staab, and Rudi Studer. The smartweb foundational ontology. Technical report, SmartWeb Project Report, SEP 2004.
58. Edmund M. Clarke and Jeannette M. Wing. Formal methods: State of the art and future directions. *ACM Comput. Surv.*, 28(4):626–643, 1996.
59. Jane Cleland-Huang, Raffaella Settimi, Oussama BenKhadra, Eugenia Berezhanskaya, and Selvia Christina. Goal-centric traceability for managing non-functional requirements. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 362–371, New York, NY, USA, 2005. ACM Press.
60. DAML Services Coalition. Daml-s: Web service description for the semantic web. In *Proceedings of the International Semantic Web Conference*, 2002.
61. W. G. Cochran and G. Cox. *Experimental Designs (2nd. Ed.).* New York: John Wiley, 1968.
62. P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.
63. J. Conklin, A. Selvin, S. Buckingham Shum, and M. Sierhuis. Facilitated hypertext for collective sensemaking: 15 years on from gIBIS. In *Proc. of the 8th International Working Conference on the Language/Action Perspective on Communication Modelling, Tilburg, the Netherlands, July 1-2*, 2003.
64. Jeff Conklin and Michael L. Begeman. gibis: a hypertext tool for exploratory policy discussion. *ACM Trans. Inf. Syst.*, 6(4):303–331, 1988.
65. R. Conte and C. Castelfranchi. Understanding the functions of norms in social groups through simulation. In *Artificial Societies: The Computer Simulation of Social Life.* UCL Press, 1995.
66. T. M. Cover and J. A. Thomas. *Elements of Information Theory.* John Wiley and Sons, 1991.
67. Bill Curtis, Herb Krasner, and Neil Iscoe. A field study of the software design process for large systems. *Commun. ACM*, 31(11):1268–1287, 1988.
68. Cycorp. Opencyc.
69. A. D'Ambrogio. A model-driven wsdl extension for describing the qos of web services. In *Proceedings of the International Conference on Web Services (ICWS'06)*, 2006.
70. Daniela Damian and James Chisan. An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. *IEEE Trans. Software Eng.*, 32(7):433–453, 2006.
71. A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20, 1993.
72. Robert Darimont and Axel van Lamsweerde. Formal refinement patterns for goal-driven requirements elaboration. In *SIGSOFT '96: Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering*, pages 179–190, New York, NY, USA, 1996. ACM Press.
73. W. Edwards Deming. *Quality, productivity, and competitive position.* Massachusets Institute of Technology, Center for Advanced Engineering Study, 1982.
74. Frank Dignum. Autonomous agents with norms. *Artif. Intell. Law*, 7(1):69–79, 1999.
75. Ralf Dömges and Klaus Pohl. Adapting traceability environments to project-specific needs. *Commun. ACM*, 41(12):54–62, 1998.
76. Paolo Donzelli. A goal-driven and agent-based requirements engineering framework. *Requirements Engineering*, 9:16–39, 2004.
77. R. G. Dromey. A model for software product quality. *IEEE Trans. Softw. Eng.*, 21:146–162, 1995.
78. R. Geoff Dromey. A model for software product quality. *IEEE Transactions on Software Engineering*, 21:146–162, 1995.
79. P. M. Dung. An argumentation theoretic foundation of logic programming. *Journal of Logic Programming*, 22:151–177, 1995.
80. A. Eagly and S. Chaiken. Attitude structure and function. In *The Handbook of Social Psychology (4th ed.).* New York: McGraw-Hill, 1996.
81. Alexander Egyed. A scenario-driven approach to traceability. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 123–132, Washington, DC, USA, 2001. IEEE Computer Society.
82. Marc Esteva, Julian A. Padget, and Carles Sierra. Formalizing a language for institutions and norms. In *ATAL*, pages 348–366, 2001.
83. Eurocontrol. Air traffic manageent user requirements document (vols. 1 and 2; ref fco.et1.st04.del01). Technical report, European Organisation for the Safety of Air Navigation, 1999.
84. S. Shaheen Fatima and Michael Wooldridge. Adaptive task resources allocation in multi-agent systems. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 537–544, New York, NY, USA, 2001. ACM Press.

85. Ernst Fehr and Simon Gächter. Fairness and retaliation: The economics of reciprocity. *Journal of Economic Perspectives*, 14:159–181, 2000.
86. Ernst Fehr and Klaus M. Schmidt. A theory of fairness, competition, and cooperation. *Quarterly Journal of Economics*, 114:817–868, 1999.
87. C. Fellbaum, editor. *WordNet: A Lexical Reference System and Its Applications*. MIT Press, 1998.
88. N. Fenton. Software measurement: A necessary scientific basis. *IEEE Trans. Softw. Eng.*, 20(3):199–206, 1994.
89. Norman E. Fenton and Martin Neil. Software metrics: roadmap. In *ICSE - Future of SE Track*, pages 357–370, 2000.
90. M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, New York, 1990.
91. Foundation for Intelligent Physical Agents. Fipa quality of service ontology specification (doc.no. sc00094a). Technical report, FIPA, 2002.
92. International Organization for Standardization. *ISO 8402 Quality management and quality assurance - Vocabulary*. International Organization for Standardization, 1986.
93. International Organization for Standardization. *ISO 9126-1:2001, Software engineering - Product quality, Part 1: Quality model*. International Organization for Standardization, 2001.
94. M. Ford and D. Billington. Strategies in human nonmonotonic reasoning. *Computational Intelligence*, 16:446–468, 2000.
95. I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Int' J. Supercomputer Applications*, 15(3):209–235, 2001.
96. Susan Fournier and David Glen Mick. Rediscovering satisfaction. *Journal of Marketing*, 63(4):5–23, 1999.
97. Christopher Fox and William Frakes. The quality approach: is it delivering? *Commun. ACM*, 40(6):24–29, 1997.
98. Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *ECAI '96: Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, pages 21–35, London, UK, 1997. Springer-Verlag.
99. Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. In *LICS '02: Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, pages 215–224, Washington, DC, USA, 2002. IEEE Computer Society.
100. Ariel Fuxman, Lin Liu, John Mylopoulos, Marco Pistore, Marco Roveri, and Paolo Traverso. Specifying and analyzing early requirements in tropos. *Requirements Engineering*, 9(2):132–150, 2004.
101. F. G. Galletta and A. L. Lederer. Some cautions on the measurement of user information satisfaction. *Decision Sciences*, 20(3):419–438, 1989.
102. A. García-Camino, J.-A. Rodriguez-Aguilar, C. Sierra, and W. W. Vasconcelos. A Distributed Architecture for Norm-Aware Agent Societies. In *DALT 2005*, volume 3904 of *LNAI*. Springer-Verlag, 2005.
103. M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, 1987.
104. Tom Gilb. *Software Metrics*. Chartwell-Bratt, 1976.
105. B. S. Gillon. Ambiguity, generality, and indeterminacy: Tests and definitions. *Synthese*, 85:391–416, 1990.
106. M. Glinz. On non-functional requirements. In *Proceedings of the 15th IEEE International Requirements Engineering Conference*, 2007.
107. J. A. Goguen and C. Linde. Techniques for requirements elicitation. In *Proceedings of the International Symposium on Requirements Engineering*, pages 152–164, Los Alamitos, California, 1993. IEEE CS Press.
108. O. C. Z. Gotel. *Contribution Structures for Requirements Engineering*. PhD thesis, Imperial College of Science, Technology, and Medicine, 1996.
109. O. C. Z. Gotel and A. C. W. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of the International Conference on Requirements Engineering*, 1994.
110. D. Graff. Shifting sands: An interest-relative theory of vagueness. *Philosophical Topics*, 20:45–81, 2000.
111. Christian Groenroos. Strategic management and marketing in the service sector. Technical report, Marketing Science Institute, Boston, MA, 1983.
112. Object Management Group. UML 2.0 superstructure specification (final adopted specification ptc/03-08-02). Technical report, Object Management Group, 2003.
113. Object Management Group. Uml profile for modeling qos and fault tolerance characteristics and mechanisms specification, v1.0 (, May 2005.
114. S. Haag, M.K. Raja, and L.L. Schkade. Quality function deployment usage in software development. *Communications of the ACM*, 39(1):41–49, 1996.
115. J. Richard Hackman and Ruth Wageman. Total quality management: Empirical, conceptual, and practical issues. *Administrative Science Quarterly*, 40(2):309–342, June 1995.
116. J. Y. Halpern. Intransitivity and vagueness. In *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR'04)*, 2004.
117. Hosam Hanna and Abdel-Illah Mouaddib. Task selection problem under uncertainty as decision-making. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1303–1308, New York, NY, USA, 2002. ACM Press.
118. Mikel Harry. *The Vision of Six Sigma. Roadmap for Breakthrough*. Sigma Publishing Company, 1994.

119. Mikel Harry. Six sigma: A breakthrough strategy for profitability. *Quality Progress*, pages 60–64, May 1998.
120. H. Hart. Ascription of responsibility and rights. In *Logic and Language*. Blackwell, 1951.
121. P. Haumer, K. Pohl, K. Weidenhaupt, and M. Jarke. Improving reviews by extending traceability. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, 1999.
122. Jeff Heflin. Owl web ontology language: Use cases and requirements.
123. K. B. Hendricks and V. Singhal. Does implementing an effective tqm program actually improve operating performance? empirical evidence from firms that have won quality awards. *Management Science*, 43:1258–1274, 1997.
124. K. B. Hendricks and V. Singhal. The long-run performance of firms with effective tqm programs. *Management Science*, 47:359–368, 2001.
125. T.A. Henzinger, J.-F. Raskin, and P.-Y. Schobbens. The regular real-time languages. In *ICALP 98: Automata, Languages, and Programming*, Lecture Notes in Computer Science 1443, pages 580–591. Springer-Verlag, 1998.
126. Caroline Herssens, Ivan J. Jureta, and Stéphane Faulkner. Capturing and using qos relationships to improve service selection. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, 2008.
127. R. Hirschheim and H. Klein. Realizing emancipatory principles in information systems development: The case for ethics. *MIS Quarterly*, 18(1):83–109, 1994.
128. D. Hitchcock. The concept of argument, and informal logic. In *Philosophy of Logic, Handbook of the Philosophy of Science 5*. Elsevier, 2006.
129. Ulrich Homann, Michael Rill, and Andreas Wimmer. Flexible value structures in banking. *Commun. ACM*, 47(5):34–36, 2004.
130. J. Hospers. *An Introduction to Philosophical Analysis*. Prentice-Hall, 1953.
131. Anthony F. Hutchings and Steve T. Knox. Creating products customers demand. *Commun. ACM*, 38(5):72–80, 1995.
132. IEEE. IEEE 610—standard glossary of software engineering terminology. Technical report, IEEE, 1991.
133. IEEE. Guide to the software engineering body of knowledge (www.swebok.org). Technical report, IEEE, 2004.
134. IEEE Standard Upper Ontology Working Group. Sumo (suggested upper merged ontology).
135. John Hagel III and Marc Singer. Unbundling the corporation. *Harvard Business Review*, March–April 1999.
136. Google Inc. Google's developer network.
137. Carnegie Mellon University Software Engineering Institute. *Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley, 1995.
138. Kaoru Ishikawa. *What is total quality control? The Japanese way*. Prentice Hall, 1985.
139. V. Issarny, C. Bidan, and T. Saridakis. Achieving middleware customization in a configuration-based development environment: Experience with the aster prototype. In *CDS '98: Proceedings of the International Conference on Configurable Distributed Systems*, page 207, Washington, DC, USA, 1998. IEEE Computer Society.
140. A. Itai and J. A. Makowshy. Unification as a complexity measure for logic programming. *J. Log. Program.*, 4(2):105–117, 1987.
141. D. Jackson. Structuring z specifications with views. *ACM Transactions on Software Engineering and Methodology*, 4:365–389, 1995.
142. J. Jackson. A keyphrase based traceability scheme. In *Proceedings of the International Colloque on Tools and Techniques for Maintaining Traceability During Design*, 1991.
143. Stephan Jacobs. Introducing measurable quality requirements: A case study. In *RE '99: Proceedings of the 4th IEEE International Symposium on Requirements Engineering*, pages 172–179, Washington, DC, USA, 1999. IEEE Computer Society.
144. Nicholas R. Jennings. On agent-based software engineering. *Artif. Intell.*, 117(2):277–296, 2000.
145. Nicholas R. Jennings, Timothy J. Norman, Peyman Faratin, P. O'Brien, and Brian Odgers. Autonomous agents for business process management. *Applied Artificial Intelligence*, 14(2):145–189, 2000.
146. Joseph M. Juran. *Quality control handbook*. McGraw-Hill, 1951.
147. I. J. Jureta, S. Faulkner, and P-Y. Schobbens. Clear justification of modeling decisions for goal-oriented requirements engineering. *Req. Eng.*, Forthcoming.
148. Ivan J. Jureta, Stephane Faulkner, Youssef Achbany, and Marco Saerens. Dynamic task allocation wihin an open service-oriented mas architecture. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multi-Agents Systems (AAMAS'07)*, 2007.
149. Ivan J. Jureta, Stephane Faulkner, Youssef Achbany, and Marco Saerens. Dynamic web service composition within a service-oriented architecture. In *Proceedings of the International Conference on Web Services (ICWS'07)*, 2007.
150. Ivan J. Jureta, Stephane Faulkner, and Pierre-Yves Schobbens. Justifying goal models. In *Proceedings of the 14th IEEE International Conference on Requirements Engineering (RE'06)*, 2006.
151. Ivan J. Jureta, Stephane Faulkner, and Pierre-Yves Schobbens. A more expressive softgoal conceptualization for quality requirements analysis. In *Proceedings of the 25th International Conference on Conceptual Modelling (ER'06)*, 2006.

152. Ivan J. Jureta, Stephane Faulkner, and Pierre-Yves Schobbens. Achieving, satisficing, excelling. In *Proceedings of the 1st International Workshop on Requirements, Intentions and Goals in Conceptual Modeling (RIGiM'07)*, 2007.
153. Ivan J. Jureta, Stephane Faulkner, and Pierre-Yves Schobbens. Clear justification of modeling decisions for goal-oriented requirements engineering. *Requirements Engineering*, Forthcoming.
154. Ivan J. Jureta, Stephane Faulkner, and Philippe Thiran. Dynamic requirements specification for adaptable and open service-oriented systems, 2007. Working paper, Information Management Research Unit, University of Namur.
155. Ivan J. Jureta, Caroline Herssens, and Stéphane Faulkner. A comprehensive quality model for service-oriented systems. *Software Quality Journal*, Forthcoming.
156. Ivan J. Jureta, John Mylopoulos, Stéphane Faulkner, and Pierre-Yves Schobbens. Core ontology for requirements engineering. Technical report, Information Management Research Unit, University of Namur (Available at http://www.jureta.net/papers/COREIIIdraft.pdf), 2007.
157. S. Kaci and L. van der Torre. Preference-based argumentation: Arguments supporting multiple values. *International Journal of Approximate Reasoning*, In press.
158. D. Kahneman, I. Ritov, and D. Schkade. Economic preferences or attitude expressions?: An analysis of dollar responses to public issues. *J. Risk and Uncertainty*, 19, 1999.
159. Daniel Kahneman, Jack L. Knetsch, and Richard H. Thaler. Experimental tests of the endowment effect and the coase theorem. *The Journal of Political Economy*, 98:1325–1348, 1990.
160. Yannis Kalfoglou and Marco Schorlemmer. Ontology mapping: the state of the art. *Knowl. Eng. Rev.*, 18(1):1–31, 2003.
161. E. Kamsties, D. Berry, and B. Peach. Detecting ambiguities in requirements documents using inspections. In *Proceedings of the International Workshop on Inspection in Software Engineering*, 2001.
162. J. N. Kapur and H. K. Kesavan. *Entropy optimization principles with applications*. Academic Press, 1992.
163. Nikos Karacapilidis and Dimitris Papadias. Computer supported argumentation and collaborative decision making: the hermes system. *Inf. Syst.*, 26(4):259–277, 2001.
164. Marjo Kauppinen and Sari Kujala. Starting improvement of requirements engineering processes: An experience report. In *PROFES '01: Proceedings of the Third International Conference on Product Focused Software Process Improvement*, pages 196–209, London, UK, 2001. Springer-Verlag.
165. Marjo Kauppinen, Sari Kujala, Tapani Aaltio, and Laura Lehtola. Introducing requirements engineering: How to make a cultural change happen in practice. In *RE '02: Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, pages 43–51, Washington, DC, USA, 2002. IEEE Computer Society.
166. E. Kavakli. *Goal-Driven Requirements Engineering: Modeling and Guidance*. PhD thesis, University of Manchester, 1999.
167. Raman Kazhamiakin, Paritosh K. Pandya, and Marco Pistore. Representation, verification, and computation of timed properties in web. In *ICWS*, pages 497–504. IEEE Computer Society, 2006.
168. Donald L. Keefer, Craig W. Kirkwood, and James L. Corner. Perspective on decision analysis applications, 1990–2001. *Decision Analysis*, 1(1):5–24, 2004.
169. Ralph L. Keeney. Decision analysis: An overview. *Operations Research*, 30(5):808–838, 1982.
170. Ralph L. Keeney and Howard Raiffa. *Decisions with Multiple Objectives: preferences and value tradeoffs*. Wiley, 1976.
171. A. Keller and H. Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *Journal of Network Systems Management*, 11(1), 2003.
172. Steven E. Keller, Laurence G. Kahn, and Roger B. Panara. Specifying software quality requirements with metrics. In *Tutorial: System and Software Requirements Engineering*. IEEE Computer Society Press, 1990.
173. Christopher Kennedy. Vagueness and grammar: The semantics of relative and absolute gradable adjectives. *Linguistics and Philosophy*, (To appear).
174. J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.
175. B. Kitchenham and S. L. Pfleeger. Software quality: The elusive target. *IEEE Softw.*, 13(1):12–21, 1996.
176. Barbara Kitchenham and Shari Lawrence Pfleeger. Software quality: The elusive target. *IEEE Softw.*, 13(1):12–21, 1996.
177. Florian Klein and Matthias Tichy. Building reliable systems based on self-organizing multi-agent systems. In *SELMAS '06: Proceedings of the 2006 international workshop on Software engineering for large-scale multi-agent systems*, pages 51–58, New York, NY, USA, 2006. ACM Press.
178. Martin J. Kollingbaum. *Norm-Governed Practical Reasoning Agents*. PhD thesis, Dept. of Computer Science, University of Aberdeen, 2005.
179. Martin J. Kollingbaum and Timothy J. Norman. Supervised interaction for electronic markets. In *Working Notes of the 4th UK Workshop on Multi-Agent Systems (UKMAS)*, 2001.
180. Martin J. Kollingbaum and Timothy J. Norman. Supervised interaction: creating a web of trust for contracting agents in electronic environments. In *AAMAS*, pages 272–279. ACM, 2002.
181. M.J. Kollingbaum and T.J. Norman. Supervised Interaction - A Form of Contract Management to create Trust between Agents. In *Trust, Reputation and Security: Theories and Practice*, volume 2631 of *Lecture Notes in Artificial Intelligence*, pages 108–122. Springer-Verlag, 2003.

182. Manuel Kolp, Paolo Giorgini, and John Mylopoulos. Multi-agent architectures as organizational structures. *Autonomous Agents and Multi-Agent Systems*, 13(1):3–25, 2006.
183. Russell B. Korobkin and Thomas S. Ulen. Law and behavioral science: Removing the rationality assumption from law and economics. *California Law Review*, 88:1051–1144, 2000.
184. G. Kotonya and I. Sommerville. *Requirements Engineering: Processes and Techniques*. John Wiley & Sons, 2000.
185. D. H. Krantz, R. D. Luce, P. Suppes, and A. Tversky. *Foundations of measurement*. New York: Academic Press, 1971 (Vol. 1), 1989 (Vol. 2), 1990 (Vol. 3).
186. D. Kreps. *Notes on the Theory of Choice*. Westview Press, Boulder, 1988.
187. Institute of Cognitive Science Laboratory for Applied Ontology and Italian National Research Council Technology. Dolce : a descriptive ontology for linguistic and cognitive engineering.
188. D. Landes and R. Studer. The treatment of non-functional requirements in mike. In *Proceedings of the European Software Engineering Conference*, 1995.
189. Kevin Lano. *Formal Object-Oriented Development*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1995.
190. Alexei Lapouchnian, Sotirios Liaskos, John Mylopoulos, and Yijun Yu. Towards requirements-driven autonomic systems design. *SIGSOFT Softw. Eng. Notes*, 30(4):1–7, 2005.
191. J. Lee and J.-Y. Kuo. New approach to requirements trade-off analysis for complex systems. *IEEE Trans. Knowl. Data Eng.*, 10:551–562, 1998.
192. J. Lee and K.-Y. Lai. What's in the design rationale? *Human-Computer Interaction*, 6:251–280, 1991.
193. Jintae Lee. Extending the potts and bruns model for recording design rationale. In *ICSE '91: Proceedings of the 13th international conference on Software engineering*, pages 114–125, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.
194. Yih-Jiun Lee. A dynamic virtual organization solution for web-services based grid middleware. In *16th Int'l Workshop on Database and Expert Systems Applications (DEXA)*. IEEE Computer Society, 2005.
195. Y. Lespérance and H.-K. Ng. Integrating planning into reactive high-level robot programs. In *Proceedings of the Second International Cognitive Robotics Workshop*, 2000.
196. P. Letelier. A framework for requirements traceability in uml-based projects. In *Proceedings of the International Workshop on Traceability in Emerging Forms of Software Engineering*, 2002.
197. Emmanuel Letier. *Reasoning about Agents in Goal-Oriented Requirements Engineering*. PhD thesis, Dept. d'Ingenierie Informatique, Universite de Louvain, 2001.
198. Emmanuel Letier and Axel van Lamsweerde. Reasoning about partial goal satisfaction for requirements and design engineering. In *Proceedings of the International Conference on Foundations of Software Engineering*, 2004.
199. Hector J. Levesque, Raymond Reiter, Yves Lesperance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83, 1997.
200. Sotirios Liaskos, Alexei Lapouchnian, Yijun Yu, Eric Yu, and John Mylopoulos. On goal-based variability acquisition and analysis. In *Proceedings of the International Requirements Engineering Conference (RE'06)*, 2006.
201. R. Likert and J. G. Likert. *New Ways of Managing Conflict*. New York: McGraw-Hill, 1976.
202. Lin Liu and Eric Yu. Designing information systems in social context: a goal and scenario modeling approach. *Information Systems*, 29:187–203, 2004.
203. X. F. Liu and J. Yen. An analytic framework for specifying and analyzing imprecise requirements. In *Proceedings of the International Conference on Software Engineering (ICSE'96)*, 1996.
204. Amazon Web Services LLC. Amazon s3 developer guide.
205. Panagiotis Louridas and Pericles Loucopoulos. A generic model for reflective design. *ACM Trans. Softw. Eng. Methodol.*, 9(2):199–237, 2000.
206. Ludwig Wittgenstein (C. K. Ogden Trans.). *Tractatus Logico-Philosophicus*. Mineola, New York: Dover Publications, Inc. (Originally published in 1922, London: Kegan Paul Ltd), 1999.
207. C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F. Brown, and Rebekah Metz. Reference model for service oriented architecture 1.0 (committee specification 1). Technical report, Organization for the Advancement of Structured Information Standards (OASIS), 2006.
208. A. Maclean, R. M. Young, V. M. E. Belotti, and T. P. Moran. Questions, options, and criteria: Elements of design space analysis. *Human-Computer Interaction*, 6:201–250, 1991.
209. Z. Manna and A. Pnuelli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.
210. James G. March. Bounded rationality, ambiguity, and the engineering of choice. *The Bell Journal of Economics*, 9(2):587–608, 1978.
211. A. Margalit. A review of scheffler (1979). *Journal of Philosophy*, 80:129–137, 1983.
212. M. L. Markus. Technochange management: using it to drive organizational change. *Journal of Information Technology*, 19:4–20, 2004.
213. E. Martínez and Y. Lespérance. Web service composition as a planning task: Experiments using knowledge-based planning. In *Proceedinds of the ICAPS-2004 Workshop on Planning and Scheduling for Web and Grid Services*, 2004.

214. C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltamari, and L. Schneider. DOLCE : a Descriptive Ontology for Linguistic and Cognitive Engineering. Technical report, Institute of Cognitive Science and Technology, Italian National Research Council, 2003.
215. Claudio Masolo, Stefano Borgo, Aldo Gangemi, Nicola Guarino, Alessandro Oltamari, and Luc Schneider. The wonderweb library of foundational ontologies—wonderweb deliverable d17 (dolce : a descriptive ontology for linguistic and cognitive engineering). Technical report, Laboratory for Applied Ontology, Institute of Cognitive Science and Technology, Italian National Research Council, 2003.
216. Claudio Masolo, Stefano Borgo, Aldo Gangemi, Nicola Guarino, and Alessandro Oltramari. Ontology library (final). Technical report, WonderWeb Deliverable D18, December 2003 Available at: http://wonderweb.semanticweb.org.
217. E. M. Maximilien and M. P. Singh. Toward autonomic services trust and selection. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'04)*, 2004.
218. Sheila McIlraith and Tran Cao Son. Adapting golog for composition of semantic web services. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR'02)*, 2002.
219. Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.
220. Christopher Menzel. Actualism. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2007.
221. Lawrence S. Meyers, Anthony Guarino, and Glenn Gamst. *Applied Multivariate Research: Design and Implementation*. Sage Publications, 2005.
222. N. H. Minsky and V. Ungureanu. A mechanism for establishing policies for electronic commerce. In *ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems*, page 322, Washington, DC, USA, 1998. IEEE Computer Society.
223. Naftaly H. Minsky and Victoria Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Trans. Softw. Eng. Methodol.*, 9(3):273–305, 2000.
224. T. M. Mitchell. *Machine learning*. McGraw-Hill, 1997.
225. John Mylopoulos, Lawrence Chung, and Brian Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Trans. Softw. Eng.*, 18(6):483–497, 1992.
226. S. Narayanan and S. A. McIlraith. Simulation, verification and automated composition of web services. In *Proceedings of the International Conference on the World Wide Web (WWW 2002)*, 2002.
227. Leila Naslavsky, Thomas A. Alspaugh, Debra J. Richardson, and Hadar Ziv. Using scenarios to support traceability. In *TEFSE '05: Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, pages 25–30, New York, NY, USA, 2005. ACM Press.
228. J. Noppen, P. van der Broek, and M. Aksit. Dealing with imprecise quality factors in software design. In *Proceedings of the Workshop on Software Quality*, 2005.
229. Timothy J. Norman, Alun D. Preece, Stuart Chalmers, Nicholas R. Jennings, Michael Luck, Viet Dung Dang, Thuc Duong Nguyen, Vikas Deora, Jianhua Shao, W. A. Gray, and N. J. Fiddian. Agent-based formation of virtual organisations. *Knowl.-Based Syst.*, 17(2-4):103–111, 2004.
230. B. Nuseibeh, J. Kramer, and A. Finkelstein. A framework for expressing the relationships between multiple views in requirements specifications. *IEEE Trans. Softw. Eng.*, 20:760–773, 1994.
231. Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. Expressing the relationships between multiple views in requirements specification. In *ICSE*, pages 187–196, 1993.
232. Daniel Oberle. *Semantic Management of Middleware, Vol.1 of the Semantic Web and Beyond*. Springer, 2006.
233. Daniel Oberle, Anupriya Ankolekar, Pascal Hitzler, Philipp Cimiano, Michael Sintek, Malte Kiesel, Babak Mougouie, Stephan Baumann, Shankar Vembu, and Massimo Romanelli. Dolce ergo sumo: On foundational and domain models in the smartweb integrated ontology (swinto). *J. Web Sem.*, 5(3):156–174, 2007.
234. United States Department of Commerce. *The Malcolm Baldridge National Quality Award: 1993 Award Criteria*. Technology Administration, National Institute of Standards and Technology, 1993.
235. Daniel E. O'Leary, Daniel Kuokka, and Robert Plant. Artificial intelligence and virtual organizations. *Commun. ACM*, 40(1):52–59, 1997.
236. Richard Oliver. Effect of expectation and disconfirmation on post-exposure product evaluation: An alternative interpretation. *Journal of Applied Psychology*, 62:480–486, 1977.
237. Richard Oliver. *Satisfaction: A Behavioral Perspective on the Consumer*. McGraw-Hill, 1996.
238. Andrea Omicini. Soda: Societies and infrastructures in the analysis and design of agent-based systems. In Paolo Ciancarini and Michael Wooldridge, editors, *AOSE*, volume 1957 of *Lecture Notes in Computer Science*, pages 185–193. Springer, 2000.
239. G. Oppy. Propositional attitudes. In *Routledge Encyclopedia of Philosophy*. London: Routledge, 1998.
240. Wanda J. Orlikowski. The duality of technology: Rethinking the concept of technology in organizations. *Organization Science*, 3(3):398–427, 1992.
241. Leon Osterweil. Strategic directions in software quality. *ACM Comput. Surv.*, 28(4):738–750, 1996.
242. J. S. Ostroff. Formal methods for the specification and design of real-time safety critical systems. In Krishna M. Kavi, editor, *Real-Time Systems: Abstractions, Languages, and Design Methodologies*, pages 60–87. IEEE Computer Society Press, Los Alamitos, California, 1992.

243. Meltem Öztürk, Alexis Tsoukias, and Philippe Vincke. Preference modelling. In Gianni Bosi, Ronen I. Brafman, Jan Chomicki, and Werner Kießling, editors, *Preferences: Specification, Inference, Applications*, number 04271 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.
244. Olga Pacheco and José Carmo. A role based model for the normative specification of organized collective agency and agents interaction. *Autonomous Agents and Multi-Agent Systems*, 6(2):145–184, 2003.
245. Pietro Panzarasa, Nicholas R. Jennings, and Timothy J. Norman. Social mental shaping: Modelling the impact of sociality on the mental states of autonomous agents. *Computational Intelligence*, 17(3):738–782, 2001.
246. M. P. Papazoglou and D. Georgakopoulos. Service-oriented computing. *Commun. ACM*, 46(10):24–28, 2003.
247. A. Parasuraman, Leonard L. Berry, and Valarie A. Zeithaml. Servqual: A multiple-item scale for measuring consumer perceptions of service quality. *Journal of Retailing*, 64:12–40, 1988.
248. A. Parasuraman and Valarie A. Zeithaml. Understanding and improving service quality: A literature review and research agenda. In B. Weitz and R. Wensley, editors, *Handbook of Marketing*. Sage Publications, 2006.
249. H. Van Dyke Parunak and James Odell. Representing social structures in uml. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 100–101, New York, NY, USA, 2001. ACM Press.
250. Jigar Patel, W. T. Luke Teacy, Nicholas R. Jennings, Michael Luck, Stuart Chalmers, Nir Oren, Timothy J. Norman, Alun D. Preece, Peter M. D. Gray, Gareth Shercliff, Patrick J. Stockreisser, Jianhua Shao, W. Alex Gray, N. J. Fiddian, and Simon G. Thompson. Conoise-g: agent-based virtual organisations. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *AAMAS*, pages 1459–1460. ACM, 2006.
251. Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
252. A. Pease and I. Niles. Ieee standard upper ontology: A progress report. *Knowledge Engineering Review, Special Issue on Ontologies and Agents*, 17:65–70, 2002.
253. Francisco A. C. Pinheiro and Joseph A. Goguen. An object-oriented tool for tracing requirements. *IEEE Softw.*, 13(2):52–64, 1996.
254. Marco Pistore, Paolo Traverso, and Piergiorgio Bertoli. Automated composition of web services by planning in asynchronous domains. In Susanne Biundo, Karen L. Myers, and Kanna Rajan, editors, *ICAPS*, pages 2–11. AAAI, 2005.
255. Marco Pistore, Paolo Traverso, Piergiorgio Bertoli, and Annapaola Marconi. Automated synthesis of composite bpel4ws web services. In *ICWS*, pages 293–301. IEEE Computer Society, 2005.
256. K. Pohl. The three dimensions of requirements engineering. In *Proceedings of the International Conference on Advanced Information Systems Engineering*, 1993.
257. K. Pohl. *Process-Centered Requirements Engineering*. J.Wiley & Sons Ltd., 1996.
258. Klaus Pohl. Pro-art: Enabling requirements pre-traceability. In *ICRE '96: Proceedings of the 2nd International Conference on Requirements Engineering (ICRE '96)*, page 76, Washington, DC, USA, 1996. IEEE Computer Society.
259. Klaus Pohl, Ralf Dömges, and Matthias Jarke. Towards method-driven trace capture. In *CAiSE '97: Proceedings of the 9th International Conference on Advanced Information Systems Engineering*, pages 103–116, London, UK, 1997. Springer-Verlag.
260. J. Pollock. Defeasible reasoning. *Cognitive Science*, 11:481–518, 1987.
261. John L. Pollock. *Thinking about Acting: Logical Foundations for Rational Decision Making*. Oxford University Press, Forthcoming.
262. D. Poole. A logical framework for defeault reasoning. *Artificial Intelligence*, 36(1):27–47, 1988.
263. H. Prakken and G. Sartor. A system for defeasible argumentation, with defeasible priorities. In *Proceedings of the International Conference on Formal Aspects of Practical Reasoning*, 1996.
264. H. Prakken and G. Vreeswijk. Logical systems for defeasible argumentation. In *Handbook of Philosophical Logic*. Kluwer, 2002.
265. B. Ramesh, T. Powers, C. Stubbs, and M. Edwards. Implementing requirements traceability: a case study. In *RE '95: Proceedings of the Second IEEE International Symposium on Requirements Engineering*, page 89, Washington, DC, USA, 1995. IEEE Computer Society.
266. Balasubramaniam Ramesh and Vasant Dhar. Supporting systems development by capturing deliberations during requirements engineering. *IEEE Trans. Softw. Eng.*, 18(6):498–510, 1992.
267. T. Ravichandran and A. Rai. Quality management in systems development: An organizational system perspective. *MIS Quarterly*, 24(3):381–415, 2000.
268. Y. Ravin and C. Leacock, editors. *Polysemy: Theoretical and Computational Approaches*. Oxford University Press, 2000.
269. S. L. Reed and D. B. Lenat. Mapping ontologies into cyc. In *Proceedings of AAAI*, 2002.
270. Carol A. Reeves and David A. Bednar. Defining quality: Alternatives and implications. *The Academy of Management Review, Special Issue: Total Quality*, 19(3):419–445, July 1994.
271. G. Regev and A. Wegeman. Where do goals come from: the underlying principles of goal-oriented requirements engineering. In *Proceedings of the International Requirements Enginering Conference (RE'05)*, 2005.

272. R. J. Richman. Ambiguity and intuition. *Mind, New Series*, 68:87–92, 1959.
273. H. W. J. Rittel and M. M. Webber. Dilemmas in a general theory of planning. *Policy Science*, 4:155–169, 1973.
274. J-A. Rodríguez-Aguilar. *On the Design and Construction of Agent-mediated Electronic Institutions*. PhD thesis, Institut d'Investigació en Intel·ligència Artificial (IIIA), Consejo Superior de Investigaciones Científicas (CSIC), Campus UAB, Bellaterra, Spain, Spain, 2001.
275. K. Rohanimanesh and S. Mahadevan. Learning to take concurrent actions. In *Proceedings of the Sixteenth Annual Conference on Neural Information Processing Systems*, 2002.
276. Colette Roland, Carine Souveyet, and Camille Ben Achour. Guiding goal modeling using scenarios. *IEEE Transactions on Software Engineering, Special Issue on Scenario Management*, pages 1055–1071, 1998.
277. Denis Rooke. Technology lecture: What is quality and how is it maintained? *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 381(1781):249–261, 1982.
278. N. S. Rosa, G. R. R. Justo, and P. R. F. Cunha. A framework for building non-functional software architectures. In *Proceedings of the ACM Symposium on Applied Computing*, 2001.
279. Douglas T. Ross and Kenneth E. Schoman Jr. Structured analysis for requirements definition. *IEEE Trans. Software Eng.*, 3(1):6–15, 1977.
280. Alvin E. Roth. Bargaining experiments. In *The Handbook of Experimental Economics 253*. 296–302, 1995.
281. L. Schneider. Designing foundational ontologies: The object-centered high-level reference ontology ochre as a case study. In *Proceedings of the International Conference on Conceptual Modeling (ER)*, 2003.
282. Luc Schneider. Designing foundational ontologies: The object-centered high-level reference ontology ochre as a case study. In Il-Yeol Song, Stephen W. Liddle, Tok Wang Ling, and Peter Scheuermann, editors, *ER*, volume 2813 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 2003.
283. John R. Searle. *Speech acts: An essay in the philosophy of language.* Cambridge: Cambridge University Press, 1969.
284. John R. Searle. A taxonomy of illocutionary acts. In *Language, Mind, and Knowledge (Studies in the Philosophy of Science, Vol.7)*. Minneapolis: University of Minnesota Press, 1975.
285. John R. Searle. The background of meaning. In J. R. Searle, F. Kiefer, and M. Bierwisch, editors, *Speech Act Theory and Pragmatics*, pages 221–232. Dordrecht, Netherlands: D. Reidel Publishing Co., 1980.
286. Simon Buckingham Shum and Nick Hammond. Argumentation-based design rationale: what use at what cost? *Int. J. Hum.-Comput. Stud.*, 40(4):603–652, 1994.
287. G. R. Simari, C. I. Chesnevar, and A. J. Garcia. The role of dialectics in defeasible reasoning. In *Proceedings of the International Conference of the Chilean Society for Computer Science*, 1994.
288. G. R. Simari and R. P. Loui. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence*, 53(2-3):125–157, 1992.
289. Guillermo R. Simari and Ronald P. Loui. A mathematical treatment of defeasible reasoning and its implementation. *Artif. Intell.*, 53(2-3):125–157, 1992.
290. Herbert A. Simon. A behavioral model of rational choice. *The Quarterly Journal of Economics*, 69(1):99–118, 1955.
291. Herbert A. Simon. *The Sciences of the Artificial*. MIT Press, Cambridge, Massachusetts, first edition, 1969.
292. Herbert A. Simon. Organizations and markets. *The Journal of Economic Perspectives*, 5(2):25–44, 1991.
293. B. Smith and P. Grenon. Basic formal ontology (bfo). Technical report, Institute for Formal Ontology and Medical Information Science, Saarland University, 2003.
294. Barry Smith and Pierre Grenon. Basic formal ontology (bfo). INFOMIS Reports.
295. Graeme Smith and Luke Wildman. Model checking z specifications using sal. In *ZB*, pages 85–103, 2005.
296. P. Smolensky, B. Fox, R. King, and C. Lewis. Computer-aided reasoned discourse, or how to argue with a computer. In *Cognitive Science and its Applications for Human-Computer Interaction*. Erlbaum, 1987.
297. S. Sohrabi, N. Prokoshyna, and S. McIlraith. Web service composition via generic procedures and customizing user preferences. In *Fifth International Semantic Web Conference (ISWC2006)*, pages 597–611, November 2006.
298. R. Sorensen. Vagueness. In *The Stanford Encyclopedia of Philosophy*. Stanford, 2003.
299. J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International, 1992.
300. E. Stenius. Mood and language game. *Synthese*, 17:254–274, 1967.
301. A. Stylianou, R. Kumar, and M. A. Khouja. A tqm-based systems development process. *DATABASE for Advances in Information Systems*, 28(3):59–71, 1997.
302. L. P. Sullivan. Quality function deployment. *Quality Progress*, 19(6):39–50, 1986.
303. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
304. Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999.
305. K. Sycara, J. Lu, and M. Klusch. Interoperability among heterogeneous software agents on the internet. Technical report, Carnegie Mellon University, PA, Technical Report CMU-RI-TR-98-22, 1998.
306. Katia P. Sycara, Seth Widoff, Matthias Klusch, and Jianguo Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5(2):173–203, 2002.
307. David Tennenhouse. Proactive computing. *Commun. ACM*, 43(5):43–50, 2000.

308. Gerald Tesauro, David M. Chess, William E. Walsh, Rajarshi Das, Alla Segal, Ian Whalley, Jeffrey O. Kephart, and Steve R. White. A multi-agent systems approach to autonomic computing. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 464–471, Washington, DC, USA, 2004. IEEE Computer Society.
309. Marco Toranzo and Jaelson Castro. A comprehensive traceability model to support the design of interactive systems. In *Proceedings of the Workshop on Object-Oriented Technology*, pages 283–284, London, UK, 1999. Springer-Verlag.
310. S. Toulmin. *The Uses of Arguments.* Cambridge University Press, 1958.
311. Amos Tversky and Richard H. Thaler. Anomalies: Preference reversals. *The Journal of Economic Perspectives*, 4:201–211, 1990.
312. Naoyasu Ubayashi, Shinji Sano, Yusaku Maeno, Satoshi Murakami, and Tetsuo Tamai. Model evolution with aspect-oriented mechanisms. In *IWPSE '05: Proceedings of the Eighth International Workshop on Principles of Software Evolution*, pages 187–194, Washington, DC, USA, 2005. IEEE Computer Society.
313. A. van Lamsweerde, R. Darimont, and Ph. Massonet. The meeting scheduler problem: Preliminary definition. Technical report, Université catholique de Louvain, 1992.
314. Axel van Lamsweerde. Divergent views in goal-driven requirements engineering. In *Proceedings of the ACM SIGSOFT Workshop on Viewpoints in Software Development*, 1996.
315. Axel van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of the International Symposium on Requirements Engineering (RE'01)*, 2001.
316. Axel van Lamsweerde. Goal-oriented requirements engineering: A roundtrip from research to practice. In *Proceedings of the International Requirements Engineering Conference (RE'04)*, 2004.
317. Axel van Lamsweerde, Robert Darimont, and Emmanuel Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Trans. Software Eng.*, 24(11):908–926, 1998.
318. Axel van Lamsweerde and Emmanuel Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Trans. Softw. Eng.*, 26(10):978–1005, 2000.
319. D. Vanderveken. Illocutionary logic and discourse typology. *Revue internationale de philosophie*, 216:243–255, 2001.
320. J. vanHeijenoort, editor. *From Frege to Goedel: A Source Book in Mathematical Logic, 1879-1931.* Cambridge, MA: Harvard University Press, 1967.
321. Wamberto Vasconcelos, Martin J. Kollingbaum, and Timothy J. Norman. Resolving conflict and inconsistency in norm-regulated virtual organizations. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2007.
322. Javier Vázquez-Salceda, Virginia Dignum, and Frank Dignum. Organizing multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 11(3):307–360, 2005.
323. B. Verheij. *Rules, Reasons, Arguments: Formal Studies of Argumentation and Defeat.* PhD thesis, Maastricht University, Holland, 1996.
324. Jose A. Calvo-Manzano Villalón, Gonzalo Cuevas Agustín, Tomás San Feliu Gilabert, Antonio De Amescua Seco, Luis García Sánchez, and Manuel Pérez Cota. Experiences in the application of software process improvement in smes. *Software Quality Control*, 10(3):261–273, 2002.
325. P. Vincke. *Multicriteria Decision-Aid.* Wiley, 1992.
326. G. A. Vreeswijk. *Studies in Defeasible Argumentation.* PhD thesis, Vrije University, Holland, 1993.
327. Roel Wieringa. A survey of structured and object-oriented software specification methods and techniques. *ACM Comput. Surv.*, 30(4):459–527, 1998.
328. T. Williamson. *Vagueness.* Routledge, 1994.
329. M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10:115–152, 1995.
330. J. Yen and W. A. Tiao. A systematic tradeoff analysis for conflicting imprecise requirements. In *Proceedings of the IEEE International Conference on Requirements Engineering (RE'97)*, pages 87–96, 1997.
331. Eric Yu. *Modeling Strategic Relationships for Process Reengineering.* PhD thesis, Dept. of Computer Science, University of Toronto, 1994.
332. Eric Yu. Towards modeling and reasoning support for early requirements engineering. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, 1997.
333. K. Yue. What does it mean to say that a specification is complete? In *Proceedings of the International Workshop on Software Specification and Design (IWSSD'87)*, 1987.
334. G. Zacharia and P. Maes. Trust management through reputation mechanisms. *Applied Artificial Intelligence*, 14, 2000.
335. L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.
336. Dietmar Zaefferer. Deskewing the searlean picture—a new speech act ontology for linguistics. In *Proceedings of the 32nd Annual Meeting of the Berkeley Linguistic Society (BLS-32)*, 2006.
337. Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Trans. Softw. Eng. Methodol.*, 12(3):317–370, 2003.
338. Pamela Zave. Classification of research efforts in requirements engineering. *ACM Comput. Surv.*, 29(4):315–321, 1997.
339. Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Trans. Softw. Eng. Methodol.*, 6(1):1–30, 1997.

340. Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *Proceedings of the International World Wide Web Conference (WWW'03)*, 2003.
341. Ji Zhang and Betty H. C. Cheng. Specifying adaptation semantics. In *WADS '05: Proceedings of the 2005 workshop on Architecting dependable systems*, pages 1–7, New York, NY, USA, 2005. ACM Press.
342. Ji Zhang and Betty H. C. Cheng. Model-based development of dynamically adaptive software. In *ICSE '06: Proceeding of the 28th international conference on Software engineering*, pages 371–380, New York, NY, USA, 2006. ACM Press.
343. C. Zhou, L.-T. Chia, and B.-S. Lee. DAML-QoS ontology for web services. In *Proceedings of the International Conference on Web Services (ICWS'04)*, 2004.