Software Requirements Engineering: A Risk-Driven Approach

Antoine Cailliau

Thesis submitted in partial fulfilment of the requirements for the Degree of Doctor in Engineering Sciences.

Institute of Information & Communication Technologies, Electronics and Applied Mathematics (ICTEAM institute)
Louvain School of Engineering
Université catholique de Louvain
Louvain-la-Neuve
Belgique

Examining board:

Prof. Axel van Lamsweerde (UCL), *Supervisor*
Prof. Peter Van Roy (UCL), *Chairperson*
Prof. Charles Pecheur (UCL), *Secretary*
Prof. Jeff Kramer (Imperial College London, UK)
Prof. Carlo Ghezzi (Politecnico di Milano, Italy)
Prof. Emmanuel Letier (UCL London, UK)

February 2018

Abstract

Requirements Engineering (RE) is concerned with the elicitation, evaluation, specification, analysis and evolution of the requirements on a software-intensive system. Risk analysis at RE time aims at anticipating adverse conditions preventing the system from achieving its mission. In goal-oriented RE, a *goal model* shows how the system's objectives contribute to each other. *Obstacle analysis* is a goal-oriented form of risk analysis aimed at increasing requirements completeness. An *obstacle* to a goal is a precondition for the non-satisfaction of this goal. An *obstacle model* shows how obstacles contribute to each other in preventing goals from being satisfied. Obstacle analysis iterates on the *identification* of obstacles, the *assessment* of their likelihood and criticality, and the *control* of likely and critical obstacles.

The thesis presents a quantitative framework for assessing and controlling obstacles to *probabilistic goals*. The latter must be satisfied in a specified percentage of cases at least. In this framework, domain experts estimate the likelihood of fine-grained obstacles together with their uncertainty margins. These estimates are up-propagated through the obstacle and goal models in order to quantitatively determine the likelihood of obstacles and the severity of their consequences. Comparing the computed satisfaction rate of high-level goals in the model with their required satisfaction rate yields measures of obstacle criticality. Countermeasures to most likely and critical obstacles are then identified. Those maximizing the satisfaction rate of high-level goals while minimizing their cost are selected for integration into the goal model. Our techniques are extended to support runtime system self-adaptation towards better satisfaction of high-level goals.

Acknowledgements

Beyond technical work and contributions, a thesis is a human adventure. During these years, I have met, been supported, and been challenged by many people I respect, admire and enjoy.

Foremost, I am very grateful to Axel van Lamsweerde, my supervisor during all these years, for his guidance, encouragement, questions, remarks, suggestions and the many corrections improving our papers and this thesis. Axel always shared his ideas, intuitions, and experience on the technical, methodological, and presentation aspects. Those years of trying to listen and apply all his wisdom now shape my vision of Computer Science and research.

I thank the members of the Jury, Charles Pecheur, Emmanuel Letier, Carlo Ghezzi, Jeff Kramer and Peter van Roy for their useful comments and feedback on earlier versions of this thesis. Let me also thank Dalal Alrajeh for our fruitful collaboration and our interesting discussions.

I would also like to thank the RE community for its kindness, interesting questions, discussions and remarks during conferences or through e-mail, and for the constructive reviews of our submitted papers. This community has been very welcoming from my very first conference; this encouraged me to pursue my Ph.D. work more than once. I take this opportunity to thank Lilianna Pasquale for the many conference talks and lunchtime breaks we shared.

Thanks are due to the IBA company, and in particular to Eric Forton, Gaëlle Gérard, Trung Truc Huynh, Ilyass Agram, together with Pierre, Vincent, Cedric, Dominique, Nathalie, David, Philippe, and Prokopios, that helped confronting our techniques and academic beliefs to the harsh industry reality. Let me also thank EMTs for sharing their views, knowledge and estimates about the Brussels Ambulance Despatching system.

I started my journey in Computer Science a long time ago. I will never be grateful enough to Yves De Saedeleer for introducing me to this lively and interesting field that today fills most of my time. His belief that CS should be taught at an early age found another adept with me.

I could not go on without mentioning Bernard Lambeau and Christophe Damas for the years we shared at the UCL INGI department; they were very supportive from the very first day, answering my dumb questions early on. I have

really enjoyed our discussions about Software and Requirements Engineering, sometimes heated, but also about students, how computer science should be taught, and society in general. Their invaluable advice on how research should be done, how to write, handle remarks, and their critical viewpoint on society and research are likely to continue to play a role in the rest of my career.

I'm deeply indebted to Simon Busard, a now long-time friend. Sharing our school projects, and then our research ideas as well as our opinions on computer science, on teaching, and on the many topics we could spend many hours to speak about, sometimes very directly, impacted this thesis.

The computer science department at UCL is filled with wonderful colleagues. During more than 7 years, I have been lucky enough to share many conversations that contributed to shape my personality and my opinion on many topics. I could not cite all colleagues I had, but I know that our discussion during lunchtime helped staying on track and often offered the breathing space needed. Special thanks to Benjamin Hessmans with whom I shared many (though not enough) runs in the woods, and to my 12-year friends and classmates Xavier Carpent, Adrien Dessy, Jean-Baptiste Mairy, Jerôme Paul, and Simon van der Linden for their support.

My thesis was definitely eased by the support staff at the INGI department. I would like to take this opportunity to thank Chantal, Corinne, Ludovic, Nicolas, Pierre, Sophie, Stéphanie, Viviane, and Vanessa one more time. Many thanks are in particular due to Vanessa Maons who was always supportive both logistically and morally, and an attentive ear when critically needed.

Last but not least, a very big "thank you" to my loved ones, friends and family. A Ph.D. is filled with ups and downs. Their unconditional support, interest and questions cannot be acknowledged enough. Answering their questions forced me to deeply understand my contributions and state these as clearly as I could. Special thanks are due, in particular, to my parents, Claire and Philippe, and to my beloved sister, Mathilde. My last thanks obviously go to Céline, for her support and all the rest.

Contents

1 Introduction **1**
1.1 Context and motivation 1
1.2 Objectives of the thesis 6
1.3 Overview of the contributions 10
1.4 Dissemination of results 12
1.5 Organisation of the thesis 13
1.6 Running example 14

2 Background: Goal-Oriented Requirements Engineering **17**
2.1 Modelling system objectives with KAOS 17
2.1.1 Goals, requirements, and environment expectations 17
2.1.2 Domain properties and domain hypotheses 19
2.1.3 Specifying goals in Metric Linear Temporal Logic 19
2.1.4 Modelling goals as AND/OR graphs 20
2.1.5 Goal conflicts 22
2.1.6 Soft goals as optimisation criteria 23
2.2 Anticipating what could go wrong 23
2.2.1 What are obstacles? 23
2.2.2 Specifying obstacles in Metric Linear Temporal Logic 24
2.2.3 Modelling obstacles as AND/OR graph 25
2.2.4 Obstacle analysis 26
2.3 Summary 29

3 A Probabilistic Framework for Goal-Oriented Requirements Engineering 31
3.1 Formal background 33
3.1.1 σ-Algebras and probability spaces 33
3.1.2 Probabilistic behavior models and measurable statements 35
3.1.3 The PCTL and PCTL* formalisms 38
3.2 Probabilistic goals 39
3.2.1 Defining probabilistic goals 39
3.2.2 Probabilistic goal models 44
3.2.3 Independence among probabilistic goals 45
3.3 Probabilistic obstacles 47
3.3.1 Defining probabilistic obstacles 47
3.3.2 Probabilistic obstacles models 49
3.3.3 Independence among probabilistic obstacles 51
3.4 Formal specification of probabilistic goals and obstacles 51
3.5 Summary 53

4 Assessing the Likelihood and Criticality of Obstacles 55
4.1 Estimating the satisfaction rate of leaf obstacles 56
4.1.1 Acquiring single fine-grained estimates 56
4.1.2 Combining estimated satisfaction rates 57
4.2 Computing satisfaction rates of higher-level obstacles and goals 61
4.2.1 BDD-based computation of satisfaction rates 61
4.2.2 Pattern-based computation of satisfaction rates 70
4.2.3 Discussion 73
4.3 Identifying most likely and critical obstacles 75
4.4 Summary 80

5 Controlling Likely and Critical Obstacles 81
5.1 Valid countermeasures 82

5.2 Iterating obstacle analysis 86
5.3 Selecting valid countermeasures to obstacle 87
5.3.1 Assessing countermeasures impact 87
5.3.2 What are most appropriate countermeasures? 89
5.3.3 Selecting most appropriate countermeasures 90
5.4 Integrating selected countermeasures in the goal model 92
5.4.1 Integration schemas for ensuring minimal changes 93
5.4.2 Change propagation for preserving refinement correctness 96
5.4.3 Obstacle resolution as exception handling 99
5.5 Summary 104

6 Handling Uncertainty in Satisfaction Rates 107
6.1 Capturing knowledge uncertainty 108
6.1.1 Uncertainty about satisfaction rates 109
6.1.2 Uncertainty metrics for goal satisfaction 110
6.1.3 Eliciting distributions for leaf obstacle satisfaction rates 112
6.2 Obstacle assessment with knowledge uncertainty 114
6.2.1 Eliciting more accurate estimates for leaf obstacles 115
6.2.2 Computing goal satisfaction rates and their uncertainty 118
6.2.3 Finding critical obstacles 119
6.3 Selecting countermeasures with knowledge uncertainty 121
6.4 Summary 123

7 Handling Obstacles At System Runtime 125
7.1 Overview of the approach 126
7.2 Running example for runtime adaptation 128
7.3 Monitoring probabilistic obstacles 130
7.3.1 Background: Monitoring non-probabilistic LTL assertions 130
7.3.2 Monitoring-based estimation of satisfaction rates 131

7.3.3 Bounding inaccuracy margins 137
7.4 Obstacle-driven system adaptation 139
7.5 Runtime deployment of most appropriate countermeasures 140
7.6 Updating uncertain probability values at runtime 142
7.7 Summary 144

8 Tool support 147
8.1 KAOSTools: The declaration language 148
8.2 KAOSTools: Architecture 161
8.3 KAOSTools: Tool suite 163
8.4 Summary 164

9 Evaluation 165
9.1 Correctness 166
9.2 Performance and scalability 169
9.3 Applicability 171
9.3.1 A car pooling system 172
9.3.2 The IBA yoke lifting system 179
9.3.3 The Brussels ambulance dispatch system 185
9.3.4 Discussion 202
9.4 Utility 204
9.5 Usability 205
9.6 Summary 206

10 Related Work 209
10.1 Risk Analysis 209
10.2 Quantitative Requirements Frameworks 214
10.3 Risk-Driven Requirements Engineering Approaches 219
10.4 Requirements Uncertainty Handling 223

10.5 Approaches for Controlling Requirement-Related Risks 226
10.6 Requirements-Driven Runtime Adaptation 229
10.7 Summary 236

11 Conclusion 237
11.1 Summary of contributions 237
11.2 Strengths and limitations 239
11.3 Open issues and perspectives 244

References 249

A Case-studies 271
A.1 Case-Study: Brussels Ambulance Dispatching System 271
A.1.1 Goal specifications 271
A.1.2 Obstacle specifications 291
A.2 Case-Study: Car Pooling System 311
A.2.1 Goal specifications 311
A.2.2 Obstacle specifications 323

B KAOSTools Formal Specification Grammar 347

Chapter 1

Introduction

This chapter introduces the context, motivation, objectives, main contributions and organization of the thesis. The running example used throughout the thesis is also introduced.

1.1 Context and motivation

The world increasingly relies on complex technological systems. Computers drastically changed how such systems are designed and operate. Computing systems are responsible for co-piloting aircrafts, tracking the heart rate of patients to deliver appropriate electrical shocks, controlling the dosage of drugs and gamma rays to cure tumors, adjusting the speed and driving wheel of cars and trucks, controlling nuclear power plants, regulating energy supply and demand, and so forth. People rely on these services and are less inclined to accept failures of those systems. The increased usage of software systems results in critical failures with regard to health, economy, and environment. A key problem is to identify what a software system should do, and what it should not; and thereby what is a failure.

This problem is at the core of Requirements Engineering. *Requirements Engineering* is a branch of System Engineering concerned with eliciting, evaluating, specifying, and analyzing the objectives, functionalities, qualities, and constraints to be achieved by a software system [Lam09].

> '*Simple improvements in these methodologies [reliability, fault tolerance, software engineering, testing] will still not solve the problem [of knowing exactly what a program will do]. Even the best design strategy depends upon some initial determination of what the program is expected to do. The critical*

problem is our inability, as humans, to flawlessly describe complex systems. Unfortunately, software analysis techniques, such as proofs and testing, depend upon the correct specification of the program for their results to have any meaning. So, even if it were possible to prove correctness, or to test all permutations of a critical program, this would not serve to demonstrate the safety of the software.' —[Lev83]

To mitigate our inability to flawlessly describe complex systems, today's engineers rely on models. Models are increasingly recognized to be an effective way of identifying what software systems shall do and shall not and why [Lam09]. A *model* is an abstract representation of the software system where key elements are highlighted, specified, and connected to others. Using models for Requirements Engineering has multiple benefits: models allow analysts to abstract from details and focus on essential aspects of the software system; they force stakeholders to be more precise about their expectations; it provide a structure for detecting errors, early in the process; and so forth.

Building a 'good' model is hard. A good model shoud be:

- *Adequate*: the model should capture what is really desired by the stakeholders while taking into account specific constraints from the domain.
- *Complete*: the model may not overlook important objectives, functionalities, qualities or constraints the system should meet.
- *Consistent*: the model does not contain conflicting objectives, functionalities, qualities or constraints.
- *Precise*: ambiguities in interpreting the model should be reduced as much as possible so that everyone understands the model the same way.

In particular, incompleteness is recognized among the major causes of software failure [Lam09]. Missing requirements and assumptions often result from unanticipated conditions under which the software should behave adequately. Our natural inclination to conceive systems that are too ideal prevents adverse conditions from being properly identified and, when likely and critical, resolved through appropriate countermeasures.

Risk analysis should be at the heart of the requirements engineering process [Ant98; Fea03; Lam00; Lam09; Lun10]. A *risk* is commonly defined as an uncertain factor whose occurrence may result in some loss of satisfaction of some corresponding objective. A risk has a *likelihood of occurrence*, and one or several undesirable *consequences* associated with it. Each consequence is uncertain as well; it has a likelihood of occurrence if the risk occurs. A consequence has a severity in terms of degree of loss of satisfaction of the corresponding objective.

Depending on the category of objective being obstructed, risks may correspond to *safety hazards* [Lev01; Lev11], *security threats* [Lam04; Pie13; Sch11], *inaccuracy conditions* on software input/output variables with respect to their environment counterpart [Lam00], and so forth.

Risks must be identified, assessed against likelihood and severity, and controlled through countermeasures [Boe91; Jon94; Lam09; Lun10].

- At requirements engineering time, risks can be systematically identified from prescriptive requirements and descriptive domain properties [Lam09].

- For risk assessment, qualitative scales can be used for quick but rough estimation of likelihood and severity [Kle99; Lev01; Lun10], possibly in relation with a requirements model [Asn11]—e.g., from '*unlikely*' to '*very likely*' and from '*low*' to '*highly critical*', respectively. Alternatively, quantitative scales can be used to capture such estimates more precisely [Bed01; Fea03], possibly in relation with a requirements model [Asn07b; Sab11; Sie14].

- For risk control, analysts may explore different countermeasures [Asn06b; Lam00; Sut98] and select the most effective ones [Asn06a; Fea03]. Such exploration may be driven by risk-reduction tactics such as *reduce risk likelihood*, *avoid risk*, *reduce consequence likelihood*, *avoid risk consequence*, or *mitigate risk consequence* [Lam09].

In goal-oriented modeling frameworks, obstacles were introduced as a natural abstraction for risk analysis [Ant98; Lam09]. An *obstacle* to a goal is a precondition for the non-satisfaction of this goal. Similarly to risk analysis, obstacle analysis includes three steps [Lam00]:

(1) *Identification.* Obstacles to every leaf goal in the goal refinement graph should be identified from relevant domain properties. Techniques are available for identifying obstacles systematically from goals and domain properties [Alr12; Lam00]. Such identification should be as exhaustive as possible.

(2) *Assessment.* The likelihood and severity of each obstacle should be determined. This may be achieved quantitatively by calculations over obstacle refinement trees and goal refinement trees [Sab11].

(3) *Control.* Likely and critical obstacles should be resolved by systematic model transformations in order to integrate appropriate countermeasures in the goal model. For obstacle resolution, operators encoding risk resolution tactics were proposed to explore alternative resolutions [Lam00]. A learning-based approach may also be used for generating alternative countermeasures from observed behaviors traces [Alr16].

Obstacle analysis has been successfully used in a variety of mission-critical systems, see, e.g., [Dar07; Lam09; Let02; Lut07; Pon07].

The *obstacle assessment* step is crucial for focusing the *control* step on those risks that are determined to be likely, and have likely and severe consequences. No systematic techniques are however available to date to support this step. In particular, it is unclear how the likelihood of obstacles should be precisely estimated and how their criticality should be assessed against the system's high-level objectives.

The *obstacle control* step is also crucial; it directly impacts the adequacy and completeness of the goal model. However, little support is currently available for the control step beyond resolution operators for countermeasure exploration. In particular, it is totally unclear what most appropriate countermeasures are, how such countermeasures are selected from the ones produced by the resolution operators, and where those selected countermeasures should be integrated in the goal model to increase its completeness and robustness.

Beyonds those two types of limitations of the current state of the art, uncertainty needs to be properly managed.

A first type of uncertainty concerns the adequacy and 'sufficient' completeness of the identified software requirements, environment assumptions and relevant domain properties [Lam09].

Besides, the satisfaction rate of those requirements and assumptions in the running system is uncertain as well [Bar10; Fea03; Let04]; satisfaction rates may vary due to unexpected events and conditions that may occur in the environment [Lam00]. The likelihood of such events and conditions occurring is also uncertain. This gives rise to the notion of *probabilistic requirements*, that is requirements that need to be satisfied in at least X% of cases.

In traditional risk analysis, uncertainty during risk assessment arises under two forms [OHa06; Vos08].

- *Physical uncertainty* refers to system phenomena. This domain-level form of uncertainty, such as the chance of a sensor breaking down, can be reduced by modifying the system—typically by changing the design or adding new requirements during the *risk control* step.

- *Knowledge uncertainty* refers to the assessment of physical uncertainty by domain experts. Our imperfect knowledge of the exact probability of a sensor breaking might lead us to estimate it with some uncertainty margin. This form of uncertainty is reduced through further data collection, consultation of more domain experts, runtime monitoring, etc.

Risk analysis techniques for RE have so far addressed physical uncertainty only [Asn11; Fea03; Lam00; Lun10; Sie14]. They rely on expert judgement at some point or another, however, to estimate the likelihood of fine-grained events or conditions emerging from their analysis. Such estimates are typically based on experience or on historical data [Asn11; Fea03; Lam00; Lun10; Sie14], sometimes refined through runtime event monitoring [Epi09; Fea98]. The extent to which those estimates are accurate remains uncertain. Expert's judgements might be subjective or biased; relevant data might not be available; accurate data might be too expensive to obtain; collected data might no longer be relevant in view of technology changes; data might be too sparse; and so on. Handling knowledge uncertainty is therefore critical for accurate obstacle analysis.

A last source of uncertainty concerns changes that may occur in the software environment and system runtime. Today's software systems are increasingly deployed in unpredictably varying environments, e.g., for autonomous vehicle control [Che09b; DeV17a], disaster management [Golo8], or adaptive security [Pas14; Pas16]. In these domains, the system must adapt to changing environments to guarantee its goals.

For runtime system adaptation, Monitor-Analyse-Plan-Execute (MAPE) cycles are often followed [Che09a; Kep03; Lem10]. The *Monitor* step collects, filters and aggregates data from the running system such as performance metrics and configuration characteristics. The *Analyze* step determines whether a change is required or not based on data analysis and reasoning about the running system. The *Plan* step structures the actions to apply in order to guarantee that the system will subsequently meet its objectives. During the *Execute* step, the system is updated with the planned actions.

At system runtime, probabilistic requirements may not be satisfied due to adverse conditions; MAPE cycles are therefore required to ensure that these requirements remain satisfied. The selection of most appropriate countermeasures is normally based on environment assumptions and obstacle satisfaction rates determined at RE time. However, these influencing factors may turn to be different at system runtime. Some assumptions might no longer hold, or the estimates provided by the experts at RE time might prove inaccurate at system runtime. In addition, other estimates might not be available at RE time, preventing the selection of most appropriate countermeasures before runtime. For better fit to the system's goals under changing or originally unknown conditions, it might prove better to defer decisions on selecting appropriate countermeasures to system runtime by monitoring adverse conditions, analyzing whether adaptations are required, selecting the appropriate countermeasures and integrating these; the running system is thus dynamically adapted accordingly.

1.2 Objectives of the thesis

Obstacle assessment. A simple yet effective technique should be available for quantitative assessment of probabilistic obstacles. The proposed quantitative technique is intended to meet the following objectives.

- *Probabilistic goals and obstacles for risk analysis.* Goals should be extended to prescribe properties to hold in at least X% of cases. Such goals should be integrated within the obstacle analysis framework together with probabilistic obstructions. (In the currently supported framework, X = 100 for all goals.)

- *Model-based assessment.* Unlike [Fea03], the assessment process should take advantage of the refinement structure provided by the goal/obstacle model in order to allow for more accurate estimation of the satisfaction rate of coarser-grained statements from finer-grained ones.

- *Formal semantics for statements to be assessed.* Unlike [Asn11; Fea03; Lun10], the specification of goals and risks should have a clear, precise semantics in terms of desirable/undesirable system behaviors. Such semantics enables their precise interpretation and the integration with other techniques for risk generation [Alr12; Lam00], countermeasure derivation [Lam00] and goal model analysis, including goal refinement checking, operationalization checking and behavior model synthesis [Lam09].

- *Measurable statements.* Unlike [Asn11; Fea03], the specification of goals and risks should be grounded on application-specific phenomena that are measurable in the environment of the software-to-be. This attenuates the common problems with subjective estimations. For the importance of making requirements measurable, see [Rob12].

- *Separation between likelihood of occurrence and criticality.* The likelihood of occurrences of an obstacle should be explicitely separated from the criticality of its consequences. Critical obstacles should be highlighted with regard to the importance and the likelihood of the obstructed high-level objectives, independently from the likelihood of occurrences of the obstacles.

Obstacle control. To address the problem of obstacle resolution, systematic techniques for selecting and integrating appropriate countermeasures should be available. These techniques are intended to satisfy the folllowing objectives.

- *Convergence towards more complete models.* The techniques for integrating countermeasures should guarantee that the model is increasingly robust and complete as resolutions are being integrated.

- *Cost-effective countermeasure selection.* The techniques for selecting countermeasures should guarantee that the target satisfaction rate of high-level objectives is satisfied while not costing more than necessary.

- *Normal behavior.* The system behaviors not affected by the obstacles should be preserved by the integration process.

- *Correctness preservation.* The correctness of goal refinements in the model should be preserved by the integration process.

- *Separation of concerns.* For higher readability and better visibility, the ideal model containing all functional and non-functional goals in normal situations should be kept visible. The specification of these goals should not be cluttered with items referring to exceptional situations.

- *Incremental integration.* Exceptional situations should be identifiable and integrated incrementally. The handling of each single situation should be isolated from the others.

- *Combinatorial mitigation.* Without any restructuring mechanism, the integration of multiple countermeasures to multiple risks considered in combination might produce a large number of cases. The model restructuring and specification should not exhibit any combinatorial blow-up of exceptional cases.

- *Traceability.* The traceability of exceptional cases should be supported from requirements to architecture.

Handling uncertainty of estimates. The quantitative obstacle analysis framework should be extended to explicitly capture and reason about uncertainties on estimated likelihoods.

- *Obstacle assessment and resolution under uncertainty.* The techniques for highlighting likely and critical obstacles should cope with knowledge uncertainty. In alignment with new-generation reliability databases that provide ranges of estimates [Akho1], single-value and multi-value estimates of the likelihood of fine-grained obstacles should be supported.
- *Highlighting of problematic uncertainty margins.* The techniques should allow analysts to highlight critical obstacles (in terms of severity of their consequences) whose knowledge uncertainty must be reduced.
- *Knowledge uncertainty reduction.* Problematic uncertainty margins for critical obstacles should be reduced to enable finer-grained obstacle assessment.

Obstacle-driven runtime system adaptation. For increased satisfaction of probabilistic system goals at system runtime, techniques should be available for dynamically deploying most appropriate countermeasures under varying conditions.

- *Monitoring of satisfaction rates.* The satisfaction rate of probabilistic goals and obstacles should be monitored at runtime. Monitoring helps determining whether probabilistic goals are currently satisfied, and what are the current likely and critical obstacles.
- *No need for explicit behavior model.* Building a consistent and complete behavior model for large distributed systems with many complex states and parallelism is often quite challenging. Unlike [Gheo9; Ghe13], the model used should therefore be declarative; system behaviors need not be explicitly modeled.
- *Model-based adaptation.* The adaptation process should be driven by a goal/obstacle model; only those adaptations which are required to meet the probabilistic assertions from this model should be made.
- *Traceability of monitored indicators and deployed countermeasures.* Unlike [Giao1], the monitored indicators and decision criteria for system adaptation should be traceable to system objectives; why such or such monitored information is required should thereby be documented.

1.3 Overview of the contributions

A probabilistic framework for goal oriented risk analysis. The thesis presents a quantitative risk assessment technique. This technique is model-based and anchored on an existing goal-oriented framework for requirements engineering.

- Probabilistic goals and obstacles are introduced. Goals may be specified to prescribe some property to hold in X% of the cases. Probabilistic obstacles provide more evidence-based answers to questions such as, e.g., what obstacles should be resolved in view of higher-level, mission-critical goals?

- A formal characterization of satisfaction rates of probabilistic goals and obstacles is provided in terms of observed states and behaviors. Beyond obstacle assessment and control, this characterization enables other techniques such as goal/obstacle monitoring at system runtime.

- A formal behavioral semantics for specification of probabilistic goals and their obstacles allows probabilities to be grounded on measurable, application-specific phenomena. This reduces subjective assessments by experts.

Assessing the likelihood and criticality of obstacles. The thesis presents a technique for propagating the satisfaction rate of leaf obstacles and for computing the severity of their consequences. This technique exploits the goal/obstacle refinement structure.

- The severity of obstacle consequences in terms of degree of goal violation is determined quantitatively and systematically by probability propagations through the obstacle and goal models.

- Most critical obstacle combinations are determined for obstacle prioritization. As a consequence, appropriate countermeasures may be identified and then selected to increase requirements completeness.

Controlling likely and critical obstacles. The thesis presents systematic techniques for selecting most appropriate countermeasures to the assessed obstacles and integrating these countermeasures into ideal goal models.

- A cost-benefit trade-off analysis guides the selection so as to maximize satisfaction rates of high-level goals under cost constraints.

- An integration mechanism is introduced as a model transformation operator to ensure *progress* towards a more complete model, *minimal change* of the original model, and *refinement correctness preservation*. Anchor goals are introduced to define where countermeasure goals should be integrated together with appropriate refinement schemas.

- To support separation between normal and exceptional situations, the thesis extends the goal language with semantics-preserving constructs for specifying exceptions and their 'handlers'—that is, the countermeasures associated with them. Model transformation operators are then provided for attaching and detaching exceptions to/from associated goals in the goal model.

Handling uncertainty in satisfaction rates. The thesis introduces knowledge uncertainty as a first-class citizen for the *obstacle assessment* and *control* steps.

- Uncertainties about obstacle estimates and goal satisfaction rates are explicitely captured in the specification of probabilistic goals and obstacles. The extended framework supports both single-point and multi-point value estimates.

- Two metrics are provided for measuring problematic knowledge uncertainties about goal satisfaction: the goal violation uncertainty and the uncertainty spread.

- A quantitative technique is provided for highlighting those obstacles with most severe consequences on the goal model and with most problematic knowledge uncertainties.

- For increased accuracy, uncertainty margins about estimates are reduced by methodical integration of estimates from multiple sources or multiple experts.

Handling obstacles at system runtime. The thesis presents a technique for driving system adapations at runtime by the satisfaction rates of high-level goals.

- Probabilistic obstacles are monitored at system runtime. The technique extends an existing monitoring technique, introduced in [Bau11] for non-probabilistic Linear Temporal Logic (LTL), to monitor probabilistic LTL assertions at system runtime. From such monitoring of leaf obstacles in obstacle refinement trees, the satisfaction rate of high-level goals is obtained by up-propagation through obstacle/goal refinement trees.

- Alternative countermeasures are selected on the fly, among those identified at RE time, when the satisfaction rate obtained for high-level goals is below their required threshold.

- The selected countermeasures are integrated into the goal model at runtime and deployed in the running software system.

1.4 Dissemination of results

The presented techniques were published in peer-reviewed conferences and in the Requirement Engineering journal.

- Chapter 3, Chapter 4: A. Cailliau and A. van Lamsweerde, "A probabilistic framework for goal-oriented risk analysis", *Proc. of RE'12—the 20th International Requirements Engineering Conference*, IEEE, 2012. (Best Paper Award.)

- Chapter 3, Chapter 4: A. Cailliau and A. van Lamsweerde, "Assessing requirements-related risks through probabilistic goals and obstacles", *Requirements Engineering*, Vol. 18, No. 2, 2013, 129-146.

- Chapter 3, Chapter 4: A. Cailliau, C. Damas, B. Lambeau, and A. van Lamsweerde, "Modeling car crash management with KAOS", *Proc. of CMA@RE—International Comparing Requirements Modeling Approaches Workshop*, IEEE, 2013.

- Chapter 5: A. Cailliau and A. van Lamsweerde, "Integrating exception handling in goal models", *Proc. of RE'14—the 22d International Requirements Engineering Conference*, IEEE, 2014.

- Chapter 6: A. Cailliau and A. van Lamsweerde, "Handling knowledge uncertainty in risk-based requirements engineering", *Proc. of RE'15—the 23rd International Requirements Engineering Conference*, IEEE, 2015.

- Chapter 7: A. Cailliau and A. van Lamsweerde, "Runtime monitoring and resolution of probabilistic obstacles to system goals", *Proc. of SEAMS'17—the 12th International Symposium on Self-adaptation and self-managing systems*, IEEE, 2017. (Best Paper Award.)

1.5 Organisation of the thesis

The thesis is structured as follows.

- Chapter 2 presents the necessary background on goal-oriented requirements engineering and obstacle analysis. It provides a description of the KAOS framework extended in the next chapters.

- Chapter 3 describes the probabilistic approach for goal-oriented requirements engineering. It provides precise definitions for probabilistic goals and obstacles, and shows how these relate to each other and how they can be formally specified.

- Chapter 4 discusses the assessment of probabilistic obstacles. It describes how leaf obstacles are estimated and how the satisfaction rate of higher-level goals is obtained by up-propagation through the obstacle and the goal models. This chapter also shows how likely and critical obstacles are highlighted.

- Chapter 5 describes the techniques for controlling likely and critical obstacles through appropriate countermeasures. It discusses what appropriate countermeasures are, how these are selected and how they are integrated in the original goal model.

- Chapter 6 describes how knowledge uncertainties about probabilistic values are integrated within the *obstacle assessment* and *control* step.
- Chapter 7 extends the proposed techniques for runtime use to drive the adaptation of software systems. The technique guarantees the required satisfation rate for high-level goals based on the monitoring of low-level obstacles.
- Chapter 8 presents the tools supporting the various techniques presented in the thesis. It also describes the text-based declaration language used to represent goal and obstacle models.
- Chapter 9 discusses the correctness, performance and scalability, applicability, utility, and usability of the proposed techniques. Applicability is evaluated on three non-trivial case-studies.
- Chapter 10 comparatively reviews work relevant to the techniques presented in the thesis. It focuses on quantitative RE and risk-driven RE. The chapter also discusses other approaches for handling uncertainties at RE level, together with approaches for requirements-driven runtime adaptation.
- Chapter 11 summarizes our results, their strengths and limitations, and discusses open issues and perspectives.

1.6 Running example

This section briefly introduces a simple case study used as running example throughout the thesis. This specific case study was selected for the availability of documented probability estimates by experts that will be used as input to our techniques.

This case study was inspired by a spent fuel pool system during decomissionning, that is, not during full power operations, as described in [Colo1]. The examples used throught the thesis were extracted from the safety guide provided by IAEA [IAEA12], from a public report produced by the committee on the Safety and Security of Commercial Spent Nuclear Fuel Storage and the

National Research Council [Cou06], from a technical study published by the U.S. Nuclear Regulatory Commision [Col01], and from various media-related sources relating the incidents and responses at the Fukushima Daiichi nuclear power plant.

> *Most nuclear power plants use reactor fuel under the form of small uranium dioxide pellets. These pellets are arranged in fuel rods, which are protected by a zirconium metal alloy. The rods are bundled to form square arrangements measuring approximately 15 to 25 centimeters on the side and 3.5 to 4.5 meters high.*
>
> *Once in the reactors, the fuel undergoes nuclear fission and generates both heat and radioactive products that require both cooling and shielding. When the fuel has been consumed (usually after 4 to 6 years), the rods are removed from the core and stored.*
>
> *The consumed nuclear fuel is often stored first at nuclear power plants in water-filled pools, called spent fuel pools. Water in these pools provides cooling of the spent fuel and shielding against radiation for the first years—typically, 5 years. The spent fuel is then moved to dry cask storage, providing passive cooling and shielding, for the following years.*
>
> *According to standards [IAEA12] and recommendations [Cou06], these pools and their operation must satisfy a large set of safety and security requirements. The three main objectives for such systems are:*
>
> - *Cooling the fuel in order to prevent over-heating.*
> - *Shielding workers and the public from radioactive emmissions.*
> - *Preventing critical accidents.*

The goal and obstacle model fragments provided in the thesis are directly inspired from the event and fault trees presented in [Col01].

Chapter 2

Background: Goal-Oriented Requirements Engineering

This chapter introduces some necessary background in goal-oriented requirements engineering. It introduces the KAOS framework extended in the following chapters. Section 2.1 presents how system objectives are modeled as goals in the KAOS framework. Section 2.2 introduces obstacle analysis as a mean for anticipating what could go wrong.

2.1 Modelling system objectives with KAOS

Goal-oriented requirements engineering is now recognized as a major modeling technique for engineering requirements of complex, mission-critical software systems. Unlike other approaches, it also capture the *why* behind the resulting software specifications [Lam09]. Among the available goal-oriented frameworks, KAOS supports a formal approach where the objectives of a system can be specified in a precise, clear-cut sense [Lam09].

KAOS also supports a multi-view modeling approach where objects, agent responsibilities, operations, and behaviors are represented in connection with goals. We focus here on goal modeling only; a detailed presentation of the framework can be found in [Lam09].

2.1.1 Goals, requirements, and environment expectations

A *goal* is a prescriptive statement of intent to be satisfied by the agents forming the system. An *agent* is an active system component having responsibilities in goal satisfaction. An agent has capabilities; these are conditions the agent can

monitor or control. The word *system* refers to the software-to-be together with its environment, including pre-existing software, devices such as sensors and actuators, people, etc.

For example, a goal in a nuclear spent fuel pool system may express that makeup water shall be provided when there is a loss of cooling in the pool:

Goal Make Up Water Provided When Loss Of Cooling
Definition Cold makeup water shall be provided when cooling is no longer guaranteed.

A goal may be *behavioral* or *soft* dependent on whether it can be satisfied in a clear-cut sense or not. In the context of risk analysis, the thesis focuses on behavioral goals. We briefly present soft goals in a later subsection.

A behavioral goal captures a maximal set of intended behaviors declaratively and implicitly. A *system behavior* is composed of parallel behaviors of the agents forming the system; an *agent behavior* is captured by a sequence of state transitions for the variables that the agent controls [Lam09]. A *state* for a variable x is a pair (x, v) where v denotes a particular value for a variable x. The global state of the system correspond to the aggregation of the state for all variables.

A behavior violates a goal if it is not among those prescribed by the formal specification of the goal [Lam09]. The notation $\pi \vDash G$ expresses that the behavior π satisfies the goal G.

Goals should be refined until they are assignable as responsibilities of single agents. In such refinements, leaf goals assigned to a software agent are *requirements* whereas leaf goals assigned to an environment agent are *expectations*.

Each goal has a *name*, a natural language *definition* and an optional *formal specification* (we present formal specification later in this section). The above goal is named MakeUp Water Provided When Loss Of Cooling.

2.1.2 Domain properties and domain hypotheses

Unlike goals, *domain properties* are descriptive statements about the problem world (such as physical laws). A domain property is considered valid in the system-to-be. As seen later, domain properties are often involved in refinements of goals and obstacles. An example of domain property in our running example is the fact that an electrical pump cannot operate without electrical power.

DomProp Power Needed For Motor Pump On
Definition Electrical power is needed to turn the motor on.

Expectations, as seen in the previous section, are prescriptive assumptions on the environment. Descriptive assumptions, called *domain hypotheses*, may be needed as well. Domain hypotheses differ from domain properties as they are not necessarily always valid in the system—but are assumed, by the analyst, to be true. For example, the fact that a pump might not be turned on when its internal electrical relay is broken is captured as a domain hypothesis:

DomHyp No Electrical Failure For Motor Pump On
Definition The pump motor being on assumes no electrical failure.

To be realizable, a behavioral goal must be consistent with all known domain properties, that is,

$$\{G, Dom\} \not\vdash false \qquad \text{(domain-consistency)}$$

2.1.3 Specifying goals in Metric Linear Temporal Logic

Metric Linear Temporal Logic (M-LTL) [Koy92; Man92] may be used for formalizing behavioral goals to enable their analysis. The goals then take the general form

$$C \Rightarrow \Theta T$$

where Θ represents a LTL operator such as: $\bigcirc$ (in the next state), $\Diamond$ (sometimes in the future), $\Diamond_{\leq d}$ (sometimes in the future before deadline d), $\Box$ (always in the future), $\Box_{\leq d}$ (always in the future up to deadline d), $\mathcal{W}$ (always in the future

unless), $\mathcal{U}$ (always in the future until), and where $C \Rightarrow \Theta T$ means $\Box(C \rightarrow \Theta T)$. The following standard logical connectives are used: $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\rightarrow$ (implies), $\leftrightarrow$ (equivalent). Note that C and T may also contain LTL operators.

A behavioral goal can be of type *Achieve* or *Maintain/Avoid* [Lam09]. The specification pattern for an *Achieve* goal is:

$$\textbf{if } \mathsf{C} \textbf{ then sooner-or-later } \mathsf{T}, \quad \text{that is,} \quad C \Rightarrow \Diamond T,$$

where C denotes a current condition and T a target condition, with obvious particularizations to *Immediate Achieve* goals ($C \Rightarrow \bigcirc T$) and *Bounded Achieve* goals ($C \Rightarrow \Diamond_{\leq d} T$) [Lam09].

The specification pattern for a *Maintain* goal is:

$$\textbf{always } [\textbf{if } \mathsf{C} \textbf{ then}] \ \mathsf{G}, \quad \text{that is,} \quad \Box\,([C \rightarrow]\, G);$$

the specification pattern for an *Avoid* goal is:

$$\textbf{never } [\textbf{if } \mathsf{C} \textbf{ then}] \ \mathsf{B}, \quad \text{that is,} \quad \Box\,([C \rightarrow]\, \neg B),$$

where G and B denote a good condition and a bad condition, respectively. (Brackets are used here to delimit optional parts in those patterns.)

2.1.4 Modelling goals as AND/OR graphs

A goal model is an AND/OR graph showing how goals contribute positively or negatively to each other [Gio03; Lam09]. Parent goals are obtained by abstraction, e.g., through *why* questions, whereas child goals are obtained by refinement, e.g., through *how* questions. Refinement paths connect parent goal nodes in this graph to their descendant goal nodes, possibly together with domain properties and hypotheses. Leaf goals are assigned to single system agents as *software requirement* or *environment expectation*. Graphically, goals are represented by parallelograms, domain properties by 'home' shapes and agents by hexagons; see the goal model fragment in Figure 2.1.

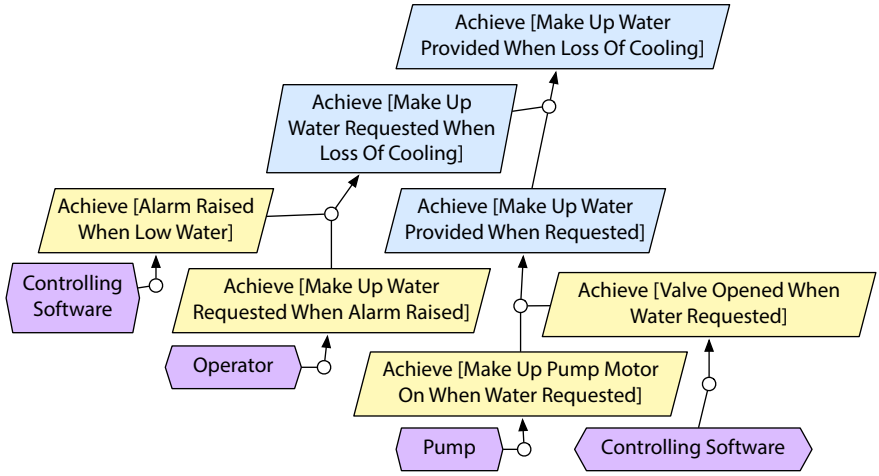


Figure 2.1 Refinement of Achieve [Make Up Water Provided When Loss Of Cooling].

Refinement patterns are available to help building goal models. They encode different refinement tactics, e.g., the *Milestone-Driven*, *Case-Driven*, *Guard-Introduction*, *Unmonitorability-Driven*, *Uncontrollability-Driven*, or *Divide-and-Conquer* refinement patterns [Dar96; Lam09].

In Figure 2.1, the top refinement is *Milestone-Driven* (with Make Up Water Requested as the milestone condition). The next left refinement follows the *Milestone-Driven* refinement pattern (with Alarm Raised as the milestone condition). The right refinement follows the *Divide-and-Conquer* refinement pattern.

The top goal is more precisely specified as follows:

$$LossOfCooling \Rightarrow \Diamond_{\leq 55h} MakeUpWaterProvided$$

Its two subgoals are obtained by application of a formalized version of the *Milestone-Driven* refinement pattern with $WaterRequested$ as milestone condition:

$$LossOfCooling \Rightarrow \Diamond_{\leq 5h} WaterRequested$$
$$WaterRequested \Rightarrow \Diamond_{\leq 50h} WaterProvided,$$

where h denotes the 'hour' time unit.

AND-refinements in a goal model should be *correct*, that is, complete, consistent and ideally minimal [Lamo9].

- A refinement is *complete* if the satisfaction of all subgoals is sufficient for the satisfaction of the parent goal in view of known domain properties:

$$\{SG_1, SG_2, ..., SG_n, Dom\} \vDash PG \qquad \text{(complete refinement)}$$

- A refinement is *consistent* if no subgoal contradicts other subgoals in the domain:

$$\{SG_1, SG_2, ..., SG_n, Dom\} \nvDash false \qquad \text{(consistent refinement)}$$

- A refinement is *minimal* if all the subgoals are needed for the satisfaction of the parent goal:

$$\{\bigwedge_{j \neq i} SG_j, Dom\} \nvDash G \quad (1 \leq i \leq n) \qquad \text{(minimal refinement)}$$

A goal refinement obtained by instantiation of a refinement pattern is formally guaranteed to be complete, consistent and minimal [Dar95; Lamo9]; the correctness proof is done once for all on the temporal logic formalization of the pattern. The partial goal model in Figure 2.1 shows three AND-refinements that are complete, consistent, and minimal.

2.1.5 Goal conflicts

The techniques for assessing and controlling probabilistic obstacles in Chapter 4 and Chapter 5 are easily extendable for assessing and resolving probabilistic conflicts.

Goals are potentially conflicting (or divergent) if there exists a satisfiable and non-trivial boundary condition B making them logically inconsistent in the domain [Lamo9], that is:

$$\{B, SG_1, SG_2, ..., SG_n, Dom\} \vDash false, \qquad \text{(conflict)}$$
$$\{B, Dom\} \nvDash false$$

As the last condition indicates, the condition $B \wedge Dom$ has to be realizable by the environment.

2.1.6 Soft goals as optimisation criteria

The selection of most appropriate countermeasures requires to comparatively evaluate alternative designs. Soft goals document the criteria for such comparison.

In constrast with behavioral goals, soft goals are goals that are not satisfied in a clear-cut sense [Myl92; Myl99]—e.g., User interface usable or Fast computation. Soft goal are more or less satisfied depending on alternative subgoal combinations contributing positively or negatively to the soft goal.

Soft goals are used as optimisation criteria to enable the comparison and selection alternative system designs [Hea11; Lam09; Let02]. Typically, they are used in minimization/maximization functions to be optimized by the system-to-be, e.g., Maximize[Computation Speed] or Minimize[Energy consumption].

2.2 Anticipating what could go wrong

In goal-oriented modeling frameworks, obstacles were introduced as a natural abstraction for risk analysis [Ant98; Lam00; Lam98a; Pot95]. This section defines obstacles, how they relate to goals and other obstacles. It also briefly reviews techniques for identifying, assessing and controling obstacles preventing the system from meeting its goals.

2.2.1 What are obstacles?

An *obstacle* to a goal is a domain-satisfiable precondition for the non-satisfaction of this goal [Lam00; Lam09]:

$$\{O, Dom\} \models \neg G \qquad \text{(obstruction)}$$

$$\{O, Dom\} \not\models false \qquad \text{(domain-feasibility)}$$

For our spent fuel pool system, an obstacle to the goal Achieve [Make Up Pump Motor On When Water Requested] is No Power Available:

Obstacle No Power Available
Definition No electrical power is available at the pump.

This obstacle satisfies both the obstruction condition (if there is no power, the motor pump cannot be on) and the domain-feasibility condition (it might be possible that no power is available.)

As a *consequence* of an obstacle occuring, some goals and ancestor goals are obstructed. As in risk analysis, an obstacle is more or less *likely* and its consequences or more or less *severe*. A likely obstacle with severe consequence is said to be *critical*.

Each obstacle has a *name*, a natural language *definition* and an optional *formal specification*. The above obstacle's name is No Power Available.

2.2.2 Specifying obstacles in Metric Linear Temporal Logic

Similarly to goals, obstacles may be formalized using Metric Linear Temporal Logic (M-LTL) [Koy92; Man92]. An obstacle to a goal of form $C \Rightarrow \Theta T$ takes the general form

$$\Diamond(C \wedge \neg\Theta T)$$

where Θ represents a LTL operator. The general form $\Diamond(C \wedge \Theta OC)$ is often used where OC denotes the *obstacle condition*.

An obstacle to an *Achieve* goal has the specification pattern:

sooner-or-later C **and never** T, that is, $\Diamond(C \wedge \Box\neg T)$,

where C denotes a current condition and T a target condition [Lam09]. An obstacle to a *Maintain* goal has the specification pattern:

sooner-or-later [C **and**] **not** G, that is, $\Diamond([C\wedge] \neg G)$

whereas an obstacle to an *Avoid* goal has the specification pattern:

sooner-or-later [C **and**] B, that is, $\Diamond([C \wedge]\ B)$

where G and B denote a good condition and a bad condition, respectively.

2.2.3 Modelling obstacles as AND/OR graph

Similarly to goals, obstacles can be AND/OR refined into subobstacles, resulting in a goal-anchored form of a risk refinement tree [Lamo9]. In such obstacle tree,

- the root obstacle is the negation of the associated leaf goal in the goal model; for a root obstacle RO to a leaf goal LG, we have:

$$\{RO, Dom\} \vDash \neg LG \qquad \text{(obstruction)}$$

- an AND-refinement captures a combination of subobstacles entailing the parent obstacle;
- an OR-refinement captures alternative ways of entailing the parent obstacle—and, recursively, of obstructing the corresponding leaf goal;
- the *leaf obstacles* are single, fine-grained obstacles whose likelihood can be estimated.

OR-refinements are ideally disjoint. This condition is useful for exploring specific resolutions to distinct non-overlapping causes.

Considering n OR-Refinements to a parent obstacle PO, each with a single subobstacle SO_i (for $1 \leq i \leq n$), the conditions above can be formally stated as follows [Lamo9]. (The generalization to more subobstacles in a given OR-Refinement is straightforward.)

- The subobstacle SO_i in an OR-refinement must entail the parent obstacle:

$$\{SO_i, Dom\} \vDash PO \qquad \text{(entailment)}$$

- OR-refinements should ideally be domain complete and disjoint:

$$\{\neg SO_1, ..., \neg SO_n, Dom\} \vDash \neg PO \qquad \text{(domain-completeness)}$$

$$\text{for all } i \neq j, \{SO_i, SO_j, Dom\} \vDash false \qquad \text{(disjointness)}$$

2.2.4 Obstacle analysis

Obstacle analysis consists of anticipating the conditions under which goals from the goal model might not be satisfied. Its main objective is to increase requirements completeness through countemeasure goals enriching the goal model. It consists of three steps [Lamoo]:

(a) *Obstacle Identification:* as many obstacles as possible to every leaf goal in the goal refinement graph should be identified from relevant domain properties.

(b) *Obstacle Assessment:* the likelihood and severity of each obstacle should be determined.

(c) *Obstacle Resolution:* likely and critical obstacles should be resolved through appropriate countermeasures to be integrated into the goal model.

Countermeasure goals need be refined until their leaf goals can be assigned to single agents. A new cycle of obstacle analysis may then be performed on those leaf countermeasure goals.

Obstacle Identification. Formal and heuristic techniques are available for the identification of obstacles [Alr12; Ant98; Lamoo; Lamo9; Sut98].

In particular, for the *Achieve* and *Maintain/Avoid* goal specification patterns introduced in Section 2.1, we may look for specific domain properties taking the form:

$$\textbf{always if } Q \textbf{ then } N, \quad \text{that is,} \quad Q \Rightarrow N,$$

where N denotes a necessary condition for the consequent Q of a leaf goal $P \Rightarrow \Theta Q$; Q is the target condition of an *Achieve* goal $C \Rightarrow \Diamond Q$ or the good condition of a *Maintain* goal $C \Rightarrow Q$. Such domain properties are equivalent to $\neg N \Rightarrow \neg Q$ (by contraposition). A one-step regresion through them of the goal negations $\Diamond(C \wedge \Box\neg Q)$ (respectively $\Diamond(C \wedge \neg Q)$) yields the obstacle:

$$\textbf{sooner-or-later } (C \textbf{ and never } N), \quad \text{that is,} \quad \Diamond(C \wedge \Box\neg N),$$

for an *Achieve* goal, or

sooner-or-later (C **and not** N), that is, $\Diamond(C \wedge \neg N)$,

for a *Maintain* goal. See [Lam09] for details.

Consider the goal Achieve [Make Up Pump Motor On When Water Requested] in Figure 2.1 whose target condition is PumpMotorOn:

$$WaterRequested \Rightarrow \Diamond_{\leq 5m} PumpMotorOn$$

Negating this goal yields the root obstacle:

$$\Diamond(WaterRequested \wedge \Box_{\leq 5m} \neg PumpMotorOn)$$

A necessary condition for the target is the following:

$$PumpMotorOn \Rightarrow \neg PumpElectricalFailure$$

This yields the bottom left subobstacle in Figure 2.2, namely:

$$\Diamond(WaterRequested \wedge \Box_{\leq 5m} PumpElectricalFailure)$$

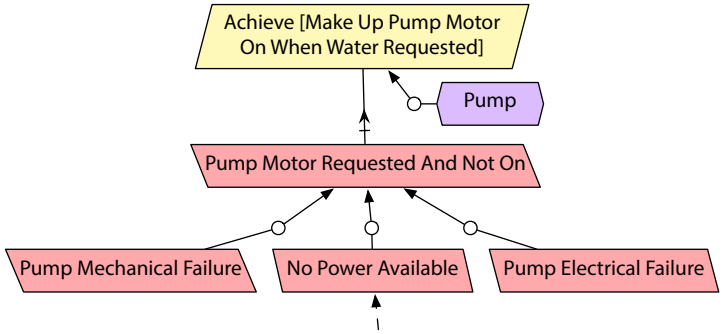


Figure 2.2 Obstacles to Achieve [Make Up Pump Motor On When Water Requested].

Obstacle Assessment. This step is aimed at identifying the likely and critical obstacles to be resolved during the next step of obstacle control. Very little work is available regarding obstacle assessment; Chapter 4 will detail our contribution to this step.

Obstacle Control. The identified obstacles, if likely and critical, shall be resolved through appropriate countermeasures to be integrated in the goal model. The countermeasure goals needs to be refined in turn until their descendants are assignable to single agents. Alternative countermeasures are to be identified; 'Most appropriate' ones need then to be selected according to a variety of criteria such as the number of obstacles resolved, the likelihood and criticality of resolved obstacles, the countermeasure cost, etc. [Lam09]. The selected countermeasures are then deployed at RE time or deferred until runtime.

Formal and heuristic techniques are available for the identification of alternative countermeasure goals [Alr16; Lam00]. In particular, a variety of tactics are available for exploring alternative ways of resolving obstacles to a goal such as:

- *Avoid obstacle* (adds a new goal requiring the obstacle to be avoided),
- *Reduce obstacle likelihood* (adds a new goal that reduces the likelihood of occurrence for the obstacle),
- *Mitigate obstacle* (adds a new goal to mitigate the consequence of the obstacle, ensuring a weakened version of the obstructed goal or of one of its ancestor),
- *Weaken goal* (weakens the obstructed goal so that the obstruction no longer occurs),
- *Substitute goal* (replace the obstructed goal by a new goal, not obstructed by the obstacle under consideration),
- *Restore goal* (adds a new goal to ensure that the obstructed goal is satisfied in a reasonable future), or
- *Substitute agent* (replace the agent responsible for the obstructed goal so that the obstacle is no longer feasible)

See [Lam00; Lam09] for details.

For example, the obstacle mitigation tactic applied to the preceding obstacle Pump Electrical Failure generates the following countermeasure goal:

Goal Achieve [Redundant Pump Motor On When Primary Pump Failure]
FormalDef $WaterRequested \wedge PumpElectricalFailure \Rightarrow \Diamond_{\leq 5m} RedundantPumpMotorOn$

This countermeasure goal ensures the ancestor goal Achieve [Make Up Water Provided When Requested] in Figure 2.1 is guaranteed even when the primary pump fails. In this specific case, the mitigation does not require a weaking of the ancestor goal.

Very little work is available on countermeasure selection and integration; Chapter 5 and Chapter 7 will propose obstacle resolution techniques both at RE time and runtime.

2.3 Summary

This chapter introduced the KAOS goal-oriented requirement engineering framework together with its obstacle analysis component. Next chapter will introduce a probabilistic extension of this framework on which the obstacle assessment and control techniques presented in the following chapters will rely.

Chapter 3

A Probabilistic Framework for Goal-Oriented Requirements Engineering

Many software failures originate from our tendency to conceive over-ideal software systems. Missing assumptions and unanticipated risks may lead to incomplete, innaccurate or inadequate requirements. Risk analysis helps the latter and should therefore be at the core of the requirements engineering process [Ant98; DeL95a; DeL95b; Lam09; Lev02]. Risk management is known to decrease the number of potential defects [Jon08]. It aims at identifying, assessing and resolving risks preventing the software-to-be from fulfilling its intended mission. For the assessment step, risks might be assessed qualitatively or quantitatively. A common weakness comes from subjective estimates used in such assessment that might be innaccurate. Few requirement engineering frameworks support quantitative risk analysis and quantitative requirements as first class citizens; even fewer propose precise characterizations for quantitative risks.

For accurate assessment of risk likelihood in our goal-oriented framework, we need to reason in terms of probability that an obstacle will occur, that is, the probability that the obstacle assertion will be satisfied. On another hand, certain classes of requirements anticipate the possibility of partial requirements satisfaction; they are often formulated in a 'probabilistic' way, requiring some target property to be satisfied in X% of cases according to some standards or regulation.

For example, the ORCON standards for ambulance dispatching requires that ambulance respond timely in X% of calls [Fin96]. Recently updated, the standard applied to NHS ambulance dispatching requires that '*all ambulance trusts to respond to 75% of Category A calls within eight minutes and to respond*

to 95% of Category A calls within 19 minutes of a request being made for a fully equipped ambulance vehicle (care or ambulance) able to transport the patient in a clinically safe manner.' [NHS17].

In view of the behavioral semantics of goals in our goal-oriented framework, requiring that '*P be satisfied in X% of cases*' amounts to require that '*X behaviors out of 100 possible behaviors satisfy P.*' The notions of probabilistic goal, probabilistic obstacle, and goal satisfaction rate introduced in this chapter are motivated by the handling of this class of requirements and the accurate assessment of their corresponding obstacles. Other classes of partial goal satisfaction might be considered, e.g., '*P shall be satisfied X% of time.*' We will come back to this when discussing future work in Chapter 11.

This chapter introduces probabilistic goals and obstacles. Our precise characterization is intended to reduce subjective estimates. The formal foundations and definitions in this chapter provide a basis for the techniques presented in the following subsequent chapters.

The satisfaction rate of a goal (resp. obstacle) is defined from the probability that the system states satisfy the specification of the goal (resp. obstacle); this probability is called *state probability*. It is precisely defined as a measure over a probabilistic behavior model. The satisfaction rate of a goal is defined as the lowest state probability to satisfy the goal's specification whereas the satisfaction rate of an obstacle is defined as the highest state probability to satisfy the obstacle's specification. The refinement structure connecting non-probabilistic goals and obstacles is generalized to probabilistic goals; the satisfaction arguments and conditions imposed by the refinement structure are thereby extended to the probabilistic framework. Specification patterns for probabilistic goals are also introduced to ease formal specification. As a result, the precise definitions for probabilistic goals and obstacles and the refinement structure connecting probabilistic goals and obstacles provide the basis for the quantitative obstacle assessment and control techniques in the next chapters.

The chapter is organized as follows. Section 3.1 introduce the necessary formal background supporting the definitions presented in the following sections. Section 3.2 introduces a precise definition for probabilistic goals and shows how probabilistic goals relate to each other in a generalized refinement structure.

Section 3.3 defines probabilistic obstacles and shows how probabilistic obstacles relate to each other and how they connect to probabilistic goals. Section 3.4 shows how probabilistic goals and obstacles are formally specified.

3.1 Formal background

The satisfaction rate of a probabilistic assertion depends on the probability that a system state satisfies this assertion, as introduced in Section 3.2. Intuitively, this state probability corresponds to the ratio between the number of behaviors satisfying the assertion and the total number of possible behaviors.

As the number of such behaviors might be infinite, a ratio between numbers of behaviors is not well defined. A proper definition requires measurability for such sets of behaviors. This requires the notions of σ-algebra and probability spaces, introduced in Section 3.1.1. To support practical ways for computing satisfaction probabilities, Section 3.1.2 introduces Markov chains as a probabilistic behavioral model. Section 3.1.3 then introduces Probabilistic Computation Tree Logic (PCTL and PCTL*) for specifying probabilistic assertions.

The reader familiar with σ-algebras, probability spaces, Markov chains and PCTL/PCTL* logic may skip this section. The background recalled here is largely inspired from [Baio8] where details and proofs can be found.

3.1.1 σ-Algebras and probability spaces

Intuitively, the probability that an assertion about system behaviors is satisfied corresponds to the ratio between the number of behaviors satisfying the assertions and the total number of possible behaviors. For example, the probability of '*Every 30 minutes, the water level shall be 25 cm above the LOW limit*' corresponds to the ratio between the number of possible behaviors where the water level is above the LOW limit and the total number of all possible behaviors. Assuming that 6 behaviors satisfy the assertion over 10 possible behaviors, the probability of the assertion would be .6. However, such ratio is not well defined, e.g. the number of possible behaviors might be infinite, not all behaviors are equally likely, and so forth. The following provides the necessary foundation for an appropriate definition.

Definition 3.1. (Measurable set) A set is *measurable* if a corresponding σ-algebra exists; a probability measure over that σ-algebra then quantifies the probability of satisfying an assertion.

Definition 3.2. (σ-algebra) A σ-algebra over a set of behaviors $\{\pi_0, \pi_1, ...\}$ is a collection Σ of subsets of behaviors $\Sigma = \{B_0, B_1, B_2, ...\}$ such that:

- The collection contains the empty set

$$\emptyset \in \Sigma$$

- The collection is closed under complementation, that is,

$$\text{if } B_i \in \Sigma, \text{ then } (\Sigma \backslash B_i) \in \Sigma$$

- The collection is closed under countable union, that is,

$$\text{if } B_0, B_1, B_2, ... \in \Sigma, \text{ then } (B_0 \cup B_1 \cup B_2 \cup ...) \in \Sigma$$

In the following, we use π to denote a behavior and B to denote a set of behaviors. For example, consider the set of possible behaviors $\{\pi_0, \pi_1\}$. The following collection Σ forms a σ-algebra for $\{\pi_0, \pi_1\}$:

$$\Sigma = \{\emptyset, \{\pi_0\}, \{\pi_1\}, \{\pi_0, \pi_1\}\}$$

It contains the empty set, is closed under complementation and is closed under countable union.

The powerset 2^B generates a σ-algebra over B. In the thesis, we always consider the σ-algebra to be generated by the powerset 2^B.

The σ-algebra Σ is equipped with a function that maps the subsets to a real number between 0 and 1, providing a probability to each element of the σ-algebra.

Definition 3.3. (Probability measure) A *probability measure* for a σ-algebra $\Sigma = \{B_0, B_1, ...\}$ is a function mapping subsets $B_0, B_1, ...$ to a real number between 0 and 1, such that:

- The probability of the subset containing all the behaviors is 1, that is,

$$Pr(B) = 1$$

- If B_i and B_j are subsets that do not share a behavior (that is $B_i \cap B_j = \emptyset$), their probability is obtained by summing the probability of the subsets:

$$Pr(B_i \cup B_j) = Pr(B_i) + Pr(B_j)$$

Assume that we want to obtain the probability of observing the single behavior $\{\pi_0\}$. We might define the following probability measure function Pr, over the generated σ-algebra Σ provided above:

$$Pr(\emptyset) = 0, \qquad Pr(\{\pi_0\}) = .4,$$
$$Pr(\{\pi_0, \pi_1\})) = 1, \qquad Pr(\{\pi_1\}) = .6,$$

In this case, the probability to observe $\{\pi_0\}$ over all possible behaviors is $.4$.

As a result, the informal ratio over behaviours is now more precise and well defined. This, however, is not practical as it requires the probability of each subset of the possible behaviors to be specified. In addition, it requires all the behaviors satisfying an assertion to be enumerated. These limitations are raised in the following subsections.

3.1.2 Probabilistic behavior models and measurable statements

The characterisation of the probability of satisfaction of an assertion presented in the previous section requires a probability measure function for all possible combinations of behaviors. Fortunately, if the behaviors of the system can be described by a probabilistic behavior model (such as a Markov chain), a σ-algebra and probability measure are provided [Baio8]. A Markov chain no longer requires a probability measure function to be provided for all combinations. This makes our characterization usable in practical applications.

Note that the probability measure is defined here over a Markov chain that must not necessarily be explicitely specified; we assume that such Markov chain exists and captures the behaviors exhibited by the system in order to make the semantics further precise.

A *Markov Chain* is a state machine where transitions between states are decorated by probabilities. The successor of a state is chosen according to probabilities on the outgoing transitions. The probability of a transition only depends on the source and the target node. It does not depend on the history for reaching the source node. This is also known as *memoryless* property.

Definition 3.4. (Markov Chain) A *Markov Chain* is a tuple $(S, \mathcal{P}(s, s'), Init, AP, L)$ where

- S is a set of states,
- $\mathcal{P}(s, s')$ specifies the probability to move from a state s to a state s' in a simple step,
- $Init$ specifies the possible initial states, with their respective probabilities,
- AP is a set of atomic propositions,
- L is labelling function mapping states to a set of atomic propositions.

A Markov Chain is often depicted by a graph where states are nodes and transitions are edges. Edges are annotated with the corresponding probability. Probabilities 1 on edges are omitted. Figure 3.1 shows a Markov chain corresponding to a simple system controlling a pump with 2 states: *WaterPumpOn* and *WaterPumpOff*.

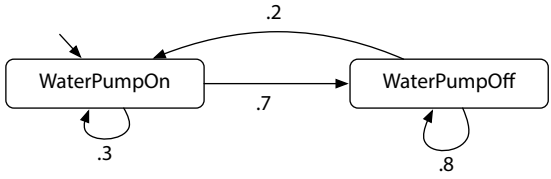


Figure 3.1 A simple pump control system

For a given Markov chain, we are interested in the corresponding probability measure function. This probability measure is defined over the σ-algebra generated from the cylinder sets for the possible behaviors. The set of possible behaviors B described by the Markov chain is the set of infinite sequence of

states $s_0 s_1 s_2 s_3 ...$ such that $\mathcal{P}(s_i, s_{i+1}) > 0$. Given a prefix, the cylinder set is the set containing all the behaviors that share the prefix. Formally, a cylinder set spanned by a finite behavior $\hat{\pi}$ is the set of infinite behaviors with the prefix $\hat{\pi}$:

$$Cyl(\hat{\pi}) = \{\pi \in B \mid \hat{\pi} \in pref(\pi)\}$$

where $pref(\pi)$ denotes all the prefixes for the behavior π.

Using the Markov chain in Figure 3.1, the cylinder set for (WaterPumpOn, WaterPumpOff) contains the behaviors (WaterPumpOn, WaterPumpOff, WaterPumpOn, WaterPumpOff,...) and (WaterPumpOn, WaterPumpOff, WaterPumpOn, WaterPumpOn, ...) but not (WaterPumpOn, **WaterPumpOn**, WaterPumpOff,...).

The σ-algebra associated with the Markov chain is the smallest σ-algebra that contains all the cylinder sets spanned by all finite behavior fragments from the Markov Chain. The probability measure associated with this σ-algebra enables a practical computation of the probability of a cylinder set:

$$Pr(Cyl(s_0...s_n)) = Init(s_0) \times \prod_{0 \leq i < n} \mathcal{P}(s_i, s_{i+1})$$

For our example, the probability of the cylinder set for (WaterPumpOn, WaterPumpOff) will be given by

$$Pr(Cyl(\text{WaterPumpOn}, \text{WaterPumpOff})) = 1 \times .7$$

This characterization provides a pratical way of determining the probability of observing a set of behaviors described by a cylinder. However, this leaves us with the problem of enumerating the behaviors (or the cylinder sets) corresponding to an assertion.

Hopefully, the cylinders can be generated from an LTL assertion. For example, the assertion **sooner-or-later** GC generates the cylinder sets $Cycl(s_0...s_i)$ where $s_i \vDash GC$. The probability is then given by $\sum_i P(Cycl(s_0...s_i))$. It is possible to analytically compute the value resulting from the infinite sum. However, efficient algorithms exist for computing the probability that behaviors of a Markov Chain starting from a state s satisfy an LTL assertion. More details about the measurability of the set of behaviors and dedicated algorithms are available in [Baio8]. In the thesis, we use the following notation for the probability that behaviors starting in s satisfy the assertion ϕ:

$$Pr^s(\phi) = Pr(\{\pi \in Behaviors(s) \mid \pi \vDash \phi\})$$

where ϕ is an LTL expression and $Behaviors(s)$ refers to the set of possible behaviors starting in s.

This section introduced the formal semantics that will be used for probabilistic goals and obstacles in the next section.

3.1.3 The PCTL and PCTL* formalisms

The previous sections introduced a precise definition enabling the computation of the probability of an LTL-like assertion. This section introduces formal logics enabling the specification of constraints on the probability of satisfying LTL-like assertions. As Section 3.4 will show, these logics can be used to formally specify probabilistic goals and obstacles.

PCTL and PCTL* are based on CTL. CTL differs from LTL as CTL is interpreted over states and not over behaviors. In a CTL formula, behaviors are universally quantified—i.e., all behaviors starting from the state satisfy the formula—or existentially quantified—i.e., there is a behavior starting from the state that satisfies the formula. PCTL formulas can be seen as quantitative counterpart of CTL where behaviors are quantified according to their probability.

PCTL *state formulas* are built using standard logic connectives (such as $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$) over atomic propositions. A state formula can include the probability operator $\mathbb{P}_I(\psi)$ where ψ is a behavior formula, and $I \subseteq [0,1]$ is a non-empty interval; $\mathbb{P}_I(\psi)$ is *true* for a state s if $Pr^s(\psi) \in I$. The *behavior formulas* are built using the temporal operators (such as $\Diamond$, $\Box$, $\mathcal{W}$, $\mathcal{U}$) over state formulas. Boolean combinations of behavior formulas are not allowed in PCTL, but are allowed in PCTL*. PCTL* is strictly more expressive than PCTL. All formulas expressed in LTL can be expressed in PCTL* but not in PCTL [Baio8].

For convenience, intervals for the probabilistic operators are often abbreviated, e.g. $\mathbb{P}_{\geq .5}(\psi)$ denotes $\mathbb{P}_{[.5,1]}(\psi)$; $\mathbb{P}_1(\psi)$ denotes $\mathbb{P}_{[1,1]}(\psi)$; etc.

The duality between the $\Diamond$- and $\Box$-operators relates their respective lower and upper bound:

$$\mathbb{P}_{\leq p}(\Diamond\phi) = \mathbb{P}_{>1-p}(\Box\neg\phi)$$

More generally, we have:

$$\mathbb{P}_{\leq p}(\psi) = \neg\mathbb{P}_{>1-p}(\psi)$$

In our framework, PCTL formulas are interpreted over states and behaviors of a Markov chain. The set of behaviors specified by a PCTL or a PCTL* formula is also measurable (see [Bai08] for a proof); we can then compute the probability of satisfying a probabilistic assertion. Section 3.4 will show how probabilistic goals and obstacles are formally specified using PCTL and PCTL*.

3.2 Probabilistic goals

Probabilistic requirements, such as '*the water level shall be above the LOW limit in 95% of the cases*', often arise from standards, regulations or guidelines. Few model-driven RE frameworks to date handle these as first-class citizens. As goals are partially satisfied due to obstacles, it seems important that such probabilistic statements are handled in the same framework.

Section 3.2.1 provides a formal definition for probabilistic goals based on the formal background presented in previous section. Section 3.2.2 generalizes for probabilistic goals the notion of refinement structure relating non-probabilistic goals to each other. Section 3.2.3 discusses the independence of goals.

3.2.1 Defining probabilistic goals

An *Achieve* goal $\Box(C \rightarrow \Diamond T)$ requires behaviors where all possible system states satisfy $C \rightarrow \Diamond T$. In case of a partially satisfied goal, behaviors rooted in a system state s might not satisfy the inner formula $C \rightarrow \Diamond T$. The state probability correspond to the chances of satisfying $C \rightarrow \Diamond T$, that is, the chances of observing a positive behavior among all possible ones.

Definition 3.5. (State probability) The *state probability* of a non-probabilistic formula ϕ in state s, denoted by $P^s(\phi)$, is defined as the value of the probability measure $Pr^s(\phi)$ over the underlying Markov chain.

The state probability $P^s(\phi)$ intuitively corresponds to the ratio between *(a)* the number of possible behaviors rooted on s satisfying ϕ, and *(b)* the number of possible behaviors rooted on s.

Regulations, norms, standards or best practices may imposes lower thresholds on probabilities, such as in [Colo1; NHS17]. To remain on the safe side, we are interested in guaranteeing lower thresholds on these probabilities; considering the lowest state probability ensures that we focus on the worst case. We thus take a conservative approach here for goals, and require a *lower* bound on the state probabilities.

Definition 3.6. (Goal satisfaction rate) The *satisfaction rate of a goal* of form $\Box(P \rightarrow \Theta Q)$ is the lowest state probability $P^s(P \rightarrow \Theta Q)$ for any possible state s.

In the following, $P(G)$ denotes the satisfaction rate of a goal G; thus, for a goal $P \rightarrow \Theta Q$:

$$P(G) = \min_{s \in S} P^s(P \rightarrow \Theta Q)$$

where S is the set of possible states. A goal is *fully satisfied* if its satisfaction rate is equal to 1.

For example, consider a system with three states and the goal Achieve [Alarm Raised When Low Water], specified by the assertion

$$WaterLevel \leq LOW \Rightarrow \Diamond_{<5\text{min}} AlarmRaised$$

Assume that this assertion has a state probability .1 in s_1; .2 in s_2; and .3 in s_3. That is, we observe 10 behaviors rooted in s_1 out of 100 possible ones that satisfy the goal specification; 20 out of 100 rooted in s_2; and 30 out of 100 rooted in s_3. The satisfaction rate of this assertion is its lowest state probability, that is, .1. We then expect that, whatever the state we observe, at least 10 behaviors out of 100 rooted in that state satisfy the goal specification. For the goal Achieve [Alarm Raised When Low Water], it means that in the worst possible state of the system with a low water level, if 100 equiprobable behaviors are possible, the alarm is raised within 5 minutes in 10 of them. The system satisfies

$$\Box\left[\mathbb{P}_{\geq .1}(WaterLevel \leq LOW \Rightarrow \Diamond_{<5\text{min}} AlarmRaised)\right]$$

The same applies to a *Maintain* goal of form $C \Rightarrow G$; with satisfaction rate x; It prescribes that all system states satisfy the assertion $C \rightarrow G$ with at least a probability x.

$$P(C \Rightarrow G) = \min_{s \in S} P^s(C \rightarrow G)$$

However, such formulation causes the satisfation rate of a *Maintain* goal to often equal 0. Indeed, only one state satisfying $C \wedge \neg G$ is sufficient for this. In practice, the formulation is often too strong and needs to be relaxed. For example, the goal $\Box(C \rightarrow G)$ might be relaxed to $\Box(C \rightarrow \Box_{\leq t} G)$. Back to our example, the goal Maintain [Water Level Above LOW] might be formalized as

$$\Box(WaterLevel > LOW)$$

Such formulation is however not realistic; it is likely to have a satisfaction rate of 0. A more realistic formalization would be

$$\Box\Box_{\leq 30\,min}(WaterLevel > LOW)$$

This deidealized goal states that the water level shall be above LOW for at least 30 minutes. States are more likely to satisfy this assumption partially.

Note that our definition for the satisfaction rate of a goal considers all states as equal; the time spent in a specific state is not taken into account. It might be very challenging, at RE time, to estimate the time spent in all possible states. Considering the lowest state probability regardless of time spent ensure that the threshold is met in all states.

We are sometimes interested in *conditional satisfaction rate*, considering the satisfaction rate of a goal to be satisfied given that some properties hold. Recalling the definition of $P^s(P \Rightarrow \Theta Q)$ from Section 3.1.2, for a goal G formalized as $P \Rightarrow \Theta Q$, we have

$$P(G) = \min_{s \in S} P^s(\phi) = \min_{s \in S} P^s(\{\pi \in Behaviors(s) \mid \pi \vDash P \rightarrow \Theta Q\})$$

The set of $Behaviors(s)$ might be restricted to a subset of behaviors satisfying some property H.

Definition 3.7. (Conditional satisfaction rate of a goal) The *conditional satisfaction rate* for a goal G under a property H, denoted by $P(G \mid H)$, is the satisfaction rate of G over all behaviors satisfying the property H. For a goal G of form $P \rightarrow \Theta Q$, it is given by

$$P(G \mid H) = \min_{s \in S} \frac{P^s(\{\pi \in Behaviors(s) \mid \pi \vDash H \wedge (P \rightarrow \Theta Q)\})}{P^s(\{\pi \in Behaviors(s) \mid \pi \vDash H\})}$$

The preceding definitions provide a basis for probabilistic goals considered in the thesis. The latter are annotated with a *computed* satisfaction rate and a *required* satisfaction rate.

Definition 3.8. (Computed satisfaction rate of a goal) The *computed satisfaction rate* (CSR) of a goal is the satisfaction rate of this goal to be determined in view of its possible obstructions.

The CSR of a goal is computed from the goal and obstacle models; Chapter 4 will describe how computed satisfaction rates for high-level goals are computed from the satisfaction rate of leaf obstacles estimated by domain experts.

In our running example, the water level might not be within the tolerable limits for various reasons, e.g., a leak in the pool, some evaporation, an unnecessary refill, a dysfunctional water pump, etc. Due to such obstacles, there is a chance of this goal not being fully satisfied. Its CSR is likely to be lower than 1.

The required satisfaction rate of a goal captures its minimal admissible satisfaction rate; the CSR should ideally be above it.

Definition 3.9. (Required satisfaction rate of a goal) The *required satisfaction rate*[1] (RSR) of a goal is the minimal satisfaction rate admissible for this goal. It is imposed by elicited requirements, existing regulations, norms, standards, and the like.

[1] Previous papers [Cai12; Cai13a; Cai14; Cai15] refer to the required satisfaction rate as *required degree of satisfaction*. The terminology was changed to avoid confusion with degree of satisfaction in fuzzy logic frameworks.

For example, we might require that, for any possible system state, 95% of the possible behaviors from this state are such that the temperature of our spent fuel pool does not exceed 54°C for more than 20 minutes. This is captured by annotating the goal Maintain [Water Temperature Below 54] with a RSR of .95.

Definition 3.10. (Probabilistic goal) A goal G is *probabilistic* if

$$0 \leq RSR(G) \leq 1$$

Note that non-probabilistic goals are generalized here; for such goals we have: $RSR(G) = 1$.

Given the computed satisfaction rate $P(G)$ and the required satisfaction rate $RSR(G)$ of a goal, we can measure the gap between these satisfaction rates. If $P(G) \geq RSR(G)$, the goal's required satisfaction threshold is reached; if $P(G) < RSR(G)$, it is not; the gap should then be as low as possible. The difference, called violation severity, allows us to measure how severe the goal violation is. Figure 3.2 shows how RSR and CSR relate.

Definition 3.11. (Violation severity) The severity of violation of a goal G is defined by:

$$SV(G) = RSR(G) - P(G)$$

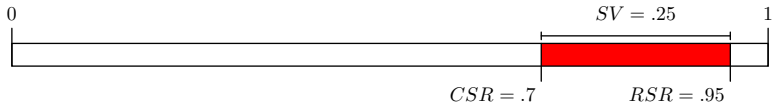


Figure 3.2 Violation Severity $SV(G)$ measures the gap between the Required Satisfation Rate $RSR(G)$ and the Computed Satisfaction Rate $P(G)$.

Similarily to non-probabilistic goals, a probabilistic goal shall be consistent with its domain. The domain-consistency condition introduced in Section 2.1 is generalized accordingly; it states that there is a least one behavior that satisfies both the goal and the domain properties:

Definition 3.12. (Domain consistency) A goal G is consistent with the domain Dom iff

$$P(G \mid Dom) > 0$$

3.2.2 Probabilistic goal models

In a goal model, goals are connected to other goals through refinement links. Refinements are generalized here to probabilistic goal satisfaction. The completeness, consistency and minimality conditions in Section 2.1 are generalized as follows.

- A refinement of probabilistic goals is *complete* if the satisfaction of the subgoals is sufficient for the satisfaction of the parent goal.

  Definition 3.13. (Complete refinement) A refinement of goal G into subgoals $SG_1, ..., SG_n$ is said to be *complete* iff

  $$P(G \mid SG_1, ..., SG_n, Dom) = 1$$

- A refinement of probabilistic goals is *consistent* if at least one behavior satisfies all subgoals and domain properties.

  Definition 3.14. (Consistent refinement) A refinement of a goal G into subgoals $SG_1, ..., SG_n$ is said to be *consistent* iff

  $$P(SG_1, ..., SG_n, Dom) > 0$$

- A refinement of probabilistic goals is *minimal* if removing any subgoal from the refinement hinder its completeness.

  Definition 3.15. (Minimal refinement) A refinement of a goal G into subgoals $SG_1, ..., SG_n$ is said to be *minimal* iff

  $$P(G \mid \bigwedge_{j \neq i} SG_j, Dom) < 1 \qquad \text{for all } i \text{ s.t. } 1 \leq i \leq n$$

A refinement of probabilistic goals is said to be *correct* if it satisfies the completeness, consistency and minimality conditions.

The completeness condition could be relaxed to allow for partial refinements. In a partial refinement, the satisfaction of the subgoals does not fully guarantee the satisfaction of the parent goal.

Definition 3.16. (Partial refinement) A refinement of a goal G into subgoals $SG_1, ..., SG_n$ is said to be *partial* iff

$$0 < P(G \mid SG_1, ..., SG_n, Dom) < 1$$

A partial refinement is minimal if removing a subgoal from the refinement reduces the satisfaction rate of the parent goal. In other words, a partial refinement is minimal if each goal contributes to the probabilistic satisfaction of the parent goal.

Definition 3.17. (Minimal partial refinement) A partial refinement of goal G into subgoals $SG_1, ..., SG_n$ is said to be *minimal* iff

$$P(G \mid \bigwedge_{j \neq i} SG_j, Dom) < P(G \mid \bigwedge_{j} SG_j, Dom) \qquad \text{for all } i \text{ s.t. } 1 \leq i \leq n$$

In a goal model, goals are connected by *conflict links* if the goals are conflicting, as recalled in Section 2.1.5. Goals are conflicting if there exists a non-trivial boundary condition making them inconsistent within the domain. In our probabilistic framework, goals are conflicting if there exist at least one behavior satisfying the boundary condition that prevents the satisfaction of the parent goal.

Definition 3.18. (Conflicting goals) A set of goals $G_1, ..., G_n$ is conflicting if there exists a boundary condition B such that

$$P(G_i \mid B \wedge \bigwedge_{i} G_{i \neq j}, Dom) = 0, \qquad P(B \mid Dom) > 0$$

3.2.3 Independence among probabilistic goals

In our example, the set of behaviors satisfying the goal Maintain [Water Level Above LOW] does not necessarily satisfy or deny the goal Achieve [Alarm Raised When Fire Detected]. Whether the level of the water is above the LOW limit does not depend on whether the software raises an alarm when a fire is detected. On the other hand, the goals Achieve [Make Up Water Provided When Requested]

and Achieve [Valve Opened When Water Requested] are not independent from each other; some behaviors satisfying the latter also satisfy the former. Goal (in)dependence is defined more precisely as follows.

Definition 3.19. (Goal independence) Two goals are *dependent* if the set of behaviors that satisfy one of them impacts the satisfaction or non-satisfaction of the other. Two goals are *independent* if they are not dependent.

In terms of conditional probabilities, the *independence* of goals G_1 and G_2 is characterized by the following conditions:

$$P(G_1 \mid G_2) = P(G_1 \mid \neg G_2) = P(G_1),$$
$$P(G_2 \mid G_1) = P(G_2 \mid \neg G_1) = P(G_2).$$

In other words, two probabilistic goals are *independent* if their satisfaction rates do not change when the other goal is satisfied. This is, however, a difficult condition to guarantee. For example, the goals Achieve [Valve Opened When Water Requested] and Achieve [Make Up Pump Motor On When Water Requested] share the obstacle No Power Available. If Achieve [Valve Opened When Water Requested] is satisfied, the obstacle No Power Available is not. Therefore, given that G is Achieve [Make Up Pump Motor On When Water Requested], we have

$$P(G \mid \neg \text{NoPowerAvailable}) > P(G)$$

These two goals are not independent.

Proposition 3.1. Two goals are dependent if they are conflicting.

Proof. If a goal G_1 conflicts with a goal G_2, we can find a boundary condition B such that $P(G_2 \mid G_1, B, Dom) = 0$ (see the conflict relation in Section 3.2.2). The set of behaviors satisfying G_1 under such circumstances necessarily denies G_2, and the goals are by definition dependent. □

3.3 Probabilistic obstacles

Requirements may be only partially satisfied due to the adverse conditions preventing their satisfaction. In our probabilistic framework, the satisfaction rate of a goal depends on the satisfaction rate of the obstacles preventing its satisfaction.

The section defines probabilistic obstacles (Section 3.3.1) and generalizes the conditions presented in Section 2.2 that relate obstacles to goals and obstacles to each other (Section 3.3.2). Section 3.3.3 then discusses conditions for obstacle independence.

3.3.1 Defining probabilistic obstacles

An obstacle $\Diamond(C \wedge \Theta OC)$ requires behaviors where at least a system state satisfy $C \wedge \Theta OC$. As obstacles might be partially satisfied, a state s has a probability to satisfy $C \wedge \Theta OC$. Dually to goals, we take a conservative approach requiring an upper bound as we need to pessimistically consider the highest chance of goal violation.

Definition 3.20. (Obstacle satisfaction rate) The satisfaction rate of an obstacle of form $\Diamond(C \wedge \Theta OC)$ is the highest state probability of the assertion $C \wedge \Theta OC$ in any possible state s.

The satisfaction rate of an obstacle O is denoted by $P(O)$. For an obstacle of form $\Diamond(C \wedge \Theta OC)$:

$$P(O) = \max_{s \in S} P^s(C \wedge \Theta OC)$$

For example, consider a system with three states and the obstacle Pump Electrical Failure, obtained in Section 2.2.4 and as seen in Figure 2.2:

Obstacle Pump Electrical Failure
FormalDef $\Diamond(WaterPumpRequested \wedge \Box_{\leq 5m} PumpElectricalFailure)$

Assume that

$$WaterPumpRequested \wedge \Box_{\leq 5m} PumpElectricalFailure$$

has a state probability .9 in s_1; .8 in s_2; and .7 in s_3. The satisfaction rate of this obstacle is the largest state probability, that is, .9. The system satisfies

$$\Box\ \mathbb{P}_{\leq .9}(WaterPumpRequested \wedge \Box_{\leq 5m} PumpElectricalFailure)$$

Similarily to goals, the probability of O over all behaviors satisfying some property H is denoted by $P(O \mid H)$.

Definition 3.21. (Conditional satisfaction rate of an obstacle) The *conditional satisfaction rate* for an obstacle O under a property H, denoted by $P(O \mid H)$, is the satisfaction rate of O over the behaviors satisfying the property H. For an obstacle O of form $\Diamond(C \wedge \Theta OC)$, it is given by,

$$P(O \mid H) = \max_{s \in S} \frac{P^s(\{\pi \in Behaviors(s) \mid \pi \vDash H \wedge (C \wedge \Theta OC)\})}{P^s(\{\pi \in Behaviors(s) \mid \pi \vDash H\})}$$

In our probabilistic framework, leaf obstacles are annotated with an estimated satisfaction rate.

Definition 3.22. (Estimated satisfaction rate of a leaf obstacle) The *estimated satisfaction rate* (ESR) of a leaf obstacle is the satisfaction rate obtained from fine-grained estimates provided by domain experts.

The ESR of leaf obstacles in obstacle refinement trees is provided by domain experts based on their experience or available data. The satisfaction rate of higher-level obstacles is computed from those estimates. Chapter 4 will show how such computation proceeds by up-propagation through the refinement trees.

A leaf obstacles can be specified precisely; it ideally capture one single exceptionnal situation that is easier for the domain experts to estimate. Their opinions and beliefs can be challenged with a precise characterization in terms of behaviors and states. Sources of subjectivity are thereby reduced.

Obstacles are related to goals through the *obstruction* relation. An obstacle has to obstruct some goal, i.e., prevent the full satisfaction of the goal; and has to be consistent with the domain, i.e., the obstacle can happen. The obstruction and domain-consistency conditions for the non-probabilistic framework in Section 2.1 are generalized as accordingly:

- The *obstruction* condition now states that the obstacle prevents the satisfaction of the obstructed goal.

  Definition 3.23. (Strong goal obstruction) A goal G is *strongly obstructed* by an obstacle O iff

  $$P(\neg G \mid O, Dom) = 1$$

- The *domain-consistency* condition states that there is a chance for the obstacle to occur.

  Definition 3.24. (Domain consistency) An obstacle O is consistent with the domain iff

  $$P(O \mid Dom) > 0$$

There again, the obstruction condition can be relaxed to account for partial obstruction. The relaxed condition states that there is a chance for the obstacle to violate the goal:

Definition 3.25. (Weak goal obstruction) A goal G is *weakly obstructed* by an obstacle O iff

$$P(\neg G \mid O, Dom) > P(\neg G \mid Dom)$$

3.3.2 Probabilistic obstacles models

For an obstacle AND-refinement, the completeness, consistency and minimality conditions are similar to those introduced in Section 3.2.2 for probabilistic goals.

- An AND-refinement of a probabilistic obstacle is *complete* if the satisfaction of the subobstacles is sufficient for the satisfaction of the parent obstacle.

  Definition 3.26. (Complete AND-refinement) A refinement of an obstacle O into subobstacles $SO_1, ..., SO_n$ is *complete* iff

$$P(O \mid SO_1, ..., SO_n, Dom) = 1$$

- An AND-refinement of a probabilistic obstacle is *consistent* if at least one behavior satisfies all subobstacles and domain properties.

  Definition 3.27. (Consistent AND-refinement) A refinement of an obstacle O into subobstacles $SO_1, ..., SO_n$ is *consistent* iff

  $$P(SO_1, ..., SO_n, Dom) > 0$$

- An AND-refinement of a probabilistic obstacle is *minimal* if removing a subobstacle hinders the completeness of the refinement.

  Definition 3.28. (Minimal AND-refinement) A refinement of an obstacle O into subobstacles $SO_1, ..., SO_n$ is *minimal* iff

  $$P(O \mid \bigwedge_{j \neq i} SO_j, Dom) < 1 \qquad \text{for all } i \text{ s.t. } 1 \leq i \leq n$$

For an OR-refinement, the subobstacles must entail their parent:

Definition 3.29. (Strong entailment) As strong OR-Refinement of an obstacle O into subobstacles $SO_1, ..., SO_n$ must satisfy

$$P(O \mid SO_i) = 1 \qquad \text{for all } i \text{ s.t. } 1 \leq i \leq n$$

This condition can be relaxed:

Definition 3.30. (Weak entailment) A weak OR-Refinement of an obstacle O into subobstacles $SO_1, ..., SO_n$ must satisfy

$$P(O \mid SO_i) > 0 \qquad \text{for all } i \text{ s.t. } 1 \leq i \leq n$$

Ideally, an obstacle refinement shall be domain-complete; all obstacles should have been identified within the domain. For an OR-Refinement, the condition states that the parent obstacle may not be satisfied through other obstacles.

Definition 3.31. (Domain completeness) An OR-Refinement of an obstacle O into subobstacles $SO_1, ..., SO_n$ is *domain complete* iff

$$P(O \mid \neg SO_1, ..., \neg SO_n, Dom) = 0$$

3.3.3 Independence among probabilistic obstacles

In our example, the probability of mechanical failure of the water pump does not depend on, e.g., the probability of a leak in the spent fuel pool. Such assumption must be carefully checked by domain experts. In contrast, the satisfaction of a parent obstacle depends on the satisfaction of its children.

Definition 3.32. (Obstacle independence) Two obstacles are *dependent* if the set of behaviors that satisfy one of them impacts the satisfaction or non-satisfaction of the other. Two obstacles are *independent* if they are not dependent.

In terms of probabilities, O_1 and O_2 are independent iff

$$P(O_1 \mid O_2) = P(O_1 \mid \neg O_2) = P(O_1),$$
$$P(O_2 \mid O_1) = P(O_2 \mid \neg O_1) = P(O_2).$$

Note that in a OR-Refinement, two disjoint obstacles O_1, O_2 are dependent as the satisfaction of O_1 prevents the satisfaction of O_2, and vice-versa. Disjointness enforces a stronger condition

$$P(O_1 \mid O_2) = 0 \qquad \text{and} \qquad P(O_2 \mid O_1) = 0$$

Obstacles in an OR-Refinement should ideally be disjoint. Obstacles in an AND-Refinement should ideally be independent.

Two non-disjoint obstacle conditions CO_1 and CO_2 can be captured through three disjoint obstacles conditions: $\neg CO_1 \wedge CO_2$, $CO_1 \wedge \neg CO_2$, and $CO_1 \wedge CO_2$. Each of these can then have a different probability.

3.4 Formal specification of probabilistic goals and obstacles

To enable formal verification techniques, probabilistic goals with a required satisfaction rate may be formalized using the PCTL/PCTL* formalism recalled in Section 3.1.3. This section introduces specification patterns for probabilistic goals similar to the specification patterns available for non-probabilistic goals.

A probabilistic goal of form $C \rightarrow \Theta T$, where Θ denote a temporal operator such as $\Diamond$ or $\Box$, with a required satisfaction rate rsr may be formalized by the following PCTL* assertion

$$\Box\mathbb{P}_{\geq rsr}(C \rightarrow \Theta T).$$

As recall, the assertion $\Box\mathbb{P}_{\geq rsr}(C \rightarrow \Theta T)$ is satisfied by a behavior π if all states s along this behavior satisfy $Pr^s(C \rightarrow \Theta T) \geq rsr$.

To ease the specification of such probabilistic goals, we introduce the following shorthand notation similar to the operator 'leads to' introduced in [Han94]

$$C \Rightarrow_{\geq p} \Theta T \quad \equiv \quad \mathbb{P}_1[\Box\mathbb{P}_{\geq p}[C \rightarrow \Theta T]]$$

where $\mathbb{P}_1$ requires the probability of the enclosed assertion to equal 1. This operator can also be formalized in PCTL as

$$\mathbb{P}_1[\Box(C \rightarrow \mathbb{P}_{\geq p}[\Theta T])]$$

Formalizing probabilistic requirements is a tedious and error-prone task, even though it is required for formal and automated reasoning. Specifications patterns encode common formalizations to ease the specification of probabilistic goals. Here we provide some probabilistic specification patterns generalizing common specification patterns presented in [Dar95].

Probabilistic *Achieve* and *Cease*. An *Achieve* goal (respectively *Cease* goal) requires a target condition to be *true* (respectively *false*) at some point in the future when a current condition is satisfied. The time elapsed between the states satisfying the current and target conditions must be bounded for finite violability. In M-LTL, this is formalized as:

$$C \Rightarrow \Diamond_{\bowtie t} T \quad \textit{(Achieve)} \qquad \text{or} \qquad C \Rightarrow \Diamond_{\bowtie t} \neg T \quad \textit{(Cease)}$$

where $\bowtie$ is a comparison operator such as $>, <, \leq, \geq$, or $=$.

The probabilistic version for an *Achieve* goal (respectively *Cease* goal), states that a target condition is achieved (respectively not achieved) from a current condition with some probability above a specified threshold p. Such specification patterns are also sometimes referred as Probabilistic Response [Gru08]. In PCTL*, this may be formalized as:

$$C \Rightarrow_{\geq p} \Diamond_{\bowtie t} T \quad \textit{(Achieve)} \qquad \text{or} \qquad C \Rightarrow_{\geq p} \Diamond_{\bowtie t} \neg T \quad \textit{(Cease)}$$

For example, the goal Achieve [Alarm Raised When Low Water] with a required satisfaction rate of .95 corresponds to the probabilistic *Achieve* specification pattern:

$$WaterLevel \leq LOW \Rightarrow_{\geq .95} \Diamond_{<5\text{min}} AlarmRaised$$

Probabilistic *Maintain* and *Avoid*. A *Maintain* goal, (respectively an *Avoid* goal) states that a target condition always remains true (respectively false) given some condition on the current state. The time for which the target condition is true may be bounded or not. In LTL, this is formalized as:

$$C \Rightarrow \Box_{\bowtie t} T \quad \textit{(Maintain)} \qquad \text{or} \qquad C \Rightarrow \Box_{\bowtie t} \neg T \quad \textit{(Avoid)}$$

where $\bowtie$ is a comparison operator such as $>, <, \leq, \geq$, or $=$.

The probabilistic version for a *Maintain* goal states that the target condition must hold with at least some given probability threshold p. For a probabilistic *Avoid* goal, the target condition may not hold with at least a probability threshold p. This can be formalized as

$$C \Rightarrow_{\geq p} \Box_{\bowtie t} T \quad \textit{(Maintain)} \qquad \text{or} \qquad C \Rightarrow_{\geq p} \Box_{\bowtie t} \neg T \quad \textit{(Avoid)}$$

For example, the goal Maintain [Water Temperature Below 54] with a required degree of satisfaction of .95 corresponds to the probabilistic *Maintain* specification pattern:

$$True \Rightarrow_{\geq .99} (\Box_{\leq 30\, min} (WaterTemperature < 54)$$

3.5 Summary

This chapter introduced probabilistic goals and probabilistic obstacles together with a precise definition in terms of states and behaviors. This characterization helps reducing subjective estimates provided by domain experts. Correctness conditions on the goal/obstacle model structure were generalized to support both probabilistic goals and obstacles. The satisfaction arguments there were formulated in terms of contraints on probabilities. The chapter provided precise characterization of the independence for probabilistic goals and obstacles. It

showed how such characterization might be enforced by leveraging the refinement structure. The chapter also showed how the specification of probabilistic goals may be facilitated by use of specification patterns to enable formal analysis.

This chapter provides the basis for the assessment and control techniques of probabilistic obstacles as elaborated in the next two chapters. In addition, the proposed characterization in terms of states and behaviors enables the monitoring, at system runtime, of probabilistic obstacles for dynamic model adaptation, as Chapter 7 will show.

Chapter 4

Assessing the Likelihood and Criticality of Obstacles

In standard risk analysis, the identified risks are assessed in terms of their likelihood and the likelihood of their consequences. Determining these likely and critical risks is an important prerequisite for determining appropriate countermeasures. Risk prioritization is commonly performed by comparing risk likelihoods, the likelihood of risk consequences and the severity of those consequences.

In goal-oriented RE, risks are captured by obstacles and consequences are expressed as loss in satisfaction of the obstructed goals. This chapter explains how likely and critical obstacles are identified. It shows how the satisfaction rate of finer-grained probabilistic obstacles are estimated, how the satisfaction rate of higher-level probabilistic obstacles and higher-level probabilistic goals are computed from the estimated satisfaction rate of those finer-grained probabilistic obstacles, that is, leaves in obstacle refinement trees.

The satisfaction rate of a leaf obstacle is estimated by domain experts as a single-value probability that can be combined with other single-value probabilities for more accurate estimates. The satisfaction rate of a higher-level obstacle is obtained by up-propagation from the leaf obstacles. The satisfaction rate of these probabilistic obstacles determines likely obstacles.

The critical obstacles are those causing a major drop in satisfaction rate of high-level goals. The criticality of an obstacle is obtained by computing the satisfaction rate of the obstructed goals. The satisfaction rate of these is computed by up-propagation from the root obstacles to the leaf goals; and from the leaf goals to the high-level goals.

Two up-propagation techniques are proposed: a BDD-based propagation that provides fast satisfaction rate computation at increased pre-computation cost; and a pattern-based propagation that provides finer-grained estimates at increased specification cost.

Determining the likely and critical obstacles amounts to finding those likely obstacle combinations maximizing the drop in high-level goal satisfaction rates. As a result, the next *control* step might then focus on likely and critical obstacles to increase the completeness, accuracy and adequacy of the requirements.

The chapter is organized as follows. Section 4.1 details how leaf obstacles are estimated. Section 4.2 details how the satisfaction rate of high-level goals is computed from the estimated leaf obstacles. Section 4.3 details how comparing the satisfaction rate with the high-level goals to their respective required satisfaction rate allow likely and critical obstacles to be determined.

4.1 Estimating the satisfaction rate of leaf obstacles

The first phase is to collect the estimates for the satisfaction rate of leaf obstacles. Theses will be up-propagated next to compute the satisfaction rate of higher-level obstacles and then of obstructed goals in the goal model.

The satisfaction rate of the leaf obstacles might be available from historical data, measurements, specification sheets provided by contractors, etc. However, such data migth not be available for cost reasons, technical difficulties, system uniqueness or some practical limitation. The estimation of satisfaction rates therefore often relies on expert judgement.

To obtain more accurate estimates from domain experts, we proceed in two steps: single, fine-grained estimates are obtained from a diversity of domain experts (Section 4.1.1) and those estimates are combined (Section 4.1.2).

4.1.1 Acquiring single fine-grained estimates

We propose the following approach, inspired from risk analysis as seen in [OHa06; Otw92], for getting estimates for the satisfaction rate of leaf obstacles.

(1) Select the leaf obstacles to be estimated by domain experts. These are the ones for which no data are available.

(2) Identify, select, and train the domain experts. The selected experts should provide a diversity of judgements, be independent in their knowledge sources, and trained to make sure they understand the models and the techniques. How to select experts is discussed in [OHa06; Otw92; Vos08]. A a detailed discussion on the elicitation techniques, biais identification, and de-biasing techniques is available in [Coo91; Vos08].

(3) Elicit the satisfaction rate of the selected leaf obstacles from the selected experts. Techniques and pitfalls for such elicitation sessions are described in [OHa06; Vos08]. Note that the structure of the goal and obstacle models might be modified by the experts during this assessment step.

(4) Aggregate and analyze the results. The next section overviews techniques for combining multiple expert assessments.

(5) Document all elements from this process, in particular, what obstacles were estimated, how experts were selected and trained, and what raw estimates are provided by the experts. This enables future evaluation by independent experts.

4.1.2 Combining estimated satisfaction rates

The use of multiple sources or multiple experts is generally recognized to increase the accuracy of estimates [Coo91]. The problem is then to combine the satisfaction rates provided by multiple experts into a single satisfaction rate. The latter will be up-propagated next through obstacle and goal trees. Informal techniques are often used in practice to reach a consensus on a single value agreed by all experts [OHa06; Vos08]. More mathematical approaches are however recognized to produce more accurate results than informal ones [Bed01].

Among the mathematical approaches, we mainly distinguish two combination schemas: bayesian combinations and non-bayesian combinations.

Bayesian combination techniques rely on Bayes' theorem for combining probabilities. However, theses techniques require and produce a probability distribution. This discussion is therefore deferred to Section 6.2.1 where uncertainties in the estimates are introduced through distribution functions.

Non-Bayesian combination techniques do not require probability distributions. These include standard combinations such as taking the smallest or the largest satisfaction rate or averaging the satisfaction rates. These combination may be generalised by a *r-norm* [Bedo1; Coo91]. There we discuss r-norm-based combinations specialized to probabilistic obstacles, as inspired by [Bedo1].

The *r-norm satisfaction rate* combines the satisfaction rate of multiple domain experts by weighting domain experts with their relative importance and smoothing out variations.

Definition 4.1. (r-norm satisfaction rate) The *r-norm satisfaction rate*, denoted $P_r(A)$, for an assertion A is given by

$$P_r(A) = \left(\sum_{e \in Experts} w_e \left[P_e(A) \right]^r \right)^{1/r}$$

where w_e is the weight assigned to the expert e and $P_e(A)$ is the probability estimated by the expert e for the assertion A. Weights should sum to 1.

Picking the right weight for the experts is an important step; how to assign weights to experts is discussed in [Bedo1; Coo91; OHa06]. In the example below, we use equal weight for all experts. Section 6.2.1 shows how weights can be systematically determined when estimates are uncertain.

Depending on the value given to the parameter r, the norm corresponds to different combinations [Bedo1; Coo91]. For example, consider the obstacle Power Cabling Failure in Figure 4.1, estimated by two different experts e, e' as follows:

$$P_e(\text{Power Cabling Failure}) = .3 \quad \text{and} \quad P_{e'}(\text{Power Cabling Failure}) = .1$$

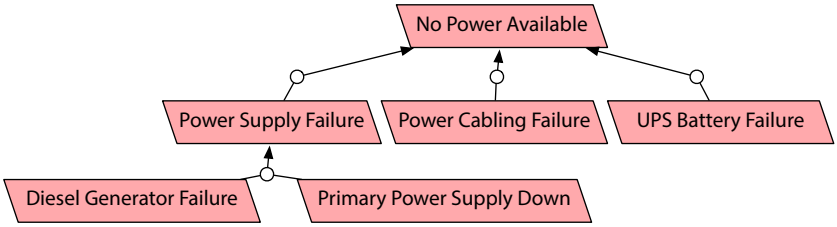


Figure 4.1 An obstacle refinement tree.

- The ∞-norm amounts to taking the largest satisfaction rate. The r-norm satisfaction rate is:

$$P_{\infty}(\text{Power Cabling Failure}) = .3$$

- The $-\infty$-norm amounts to taking the smallest satisfaction rate. The r-norm satisfaction rate is:

$$P_{-\infty}(\text{Power Cabling Failure}) = .1$$

- As $P_r(A) \to \prod P_e(A)^{w_e}$ when $r \to 0$, the 0-norm corresponds to a geometric mean [Coo91]. The r-norm satisfaction rate is:

$$P_0(\text{Power Cabling Failure}) = \sqrt{.3 \times .1} = 0.17$$

- The 1-norm corresponds to an arithmetic mean. The r-norm satisfaction rate is:

$$P_1(\text{Power Cabling Failure}) = \frac{.3 + .1}{2} = 0.20$$

As shown in the example above, the r-norm satisfaction rate depends on parameter r. Which *r-norm satisfaction rate* to chose depends on which properties for the combined estimate are desirable among the following ones.

- **Zero Preservation**. If all expert estimates that the satisfaction rate of an obstacle is zero, the r-norm satisfaction rate should be zero. This property is satisfied for all *r-norms*.

- **Marginalization**. The r-norm satisfaction rate should not depend on how the obstacle is OR-Refined. In other words, given an obstacle O OR-refined in two disjoint subobstacles SO_1, SO_2, the r-norm satisfaction rate should satisfy

$$P_r(O) = P_r(SO_1) + P_r(SO_2).$$

  For example, consider two experts estimating the subobstacles of the OR-Refinement in Figure 4.1. The first expert estimates .1 for Power Cabling Failure and .2 for UPS Battery Failure. The second expert estimates .2 for Power Cabling Failure and .1 for UPS Battery Failure. (For simplicity, we ignore the subobstacle Power Supply Failure.) As we will show in Section 4.2, both experts estimate that the satisfaction rate of the parent obstacle No Power Available is $.1 + .2 = .3$.

  Using a 0-*norm* for combining the satisfaction rate of the subobstacles yields $\sqrt{.2 \times .1} = .14$ for both subobstacles. With these combined estimates, the satisfaction rate of the parent obstacle is $.14 + .14 = .28$.

  With the 1-*norm*, however, the r-norm satisfaction rate of both subobstacle is $.5 \times (.2 + .1) = .15$. In that case, the satisfaction rate of the parent obstacle is .3, as the experts have independently estimated those quantities.

  The 1-*norm* is known to be the only r-*norm* to exhibit the marginalization property [Bedo1].

- **Independence Preservation**. The r-norm satisfaction rate should not depend on how the obstacle is AND-Refined. In other words, given an obstacle O AND-refined in two independent subobstacles SO_1, SO_2, the r-norm satisfaction rate should satisfy:

$$P_r(O) = P_r(SO_1) \times P_r(SO_2).$$

For example, consider the AND-refinement depicted in Figure 4.1. The first expert estimates .1 for the obstacle Diesel Generator Failure and .2 for Primary Power Supply Down. The second expert estimate .2 for Diesel Generator Failure and .1 for Primary Power Supply Down. Considering experts independently, they both estimate the satisfaction rate of the parent obstacle Power Supply Failure to be $.1 \times .2 = .02$.

Using 1-*norm* to combine their estimates, we obtain .15 for both subobstacles. The satisfaction rate of the parent obstacle is therefore $.15 \times .15 = .0225$, which differs from the estimate they provided independently.

Using 0-*norm*, the r-norm satisfaction rate of the subobstacle is $\sqrt{.2 \times .1}$. The satisfaction rate of the parent obstacle O is $(\sqrt{.2 \times .1})^2 = .02$.

The 0-*norm* has the independence preservation property unlike the 1-*norm* [Bedo1].

4.2 Computing satisfaction rates of higher-level obstacles and goals

The previous section showed how the satisfaction rate of leaf obstacles can be estimated from multiple domain experts. The second phase consists of determining the satisfaction rates of higher-level obstacles and then of high-level goals in the goal model from these estimated satisfaction rates. For obstacle assessment, the former satisfaction rates provide obstacle likelihoods whereas the latter satisfaction rates provide obstacle severities. This section shows two alternative propagation techniques for computing the satisfaction rate of higher-level obstacles and goals.

4.2.1 BDD-based computation of satisfaction rates

A first procedure for computing the satisfaction rate of goal in a goal model can be outlined as follows. (Details are provided in the following subsections.)

(1) For all root obstacles obstructing the goal, the set of AND-combinations of leaf obstacles entailing the root obstacle in the obstacle model is computed; this set is called the *entailment superset*. (Section 4.2.1.1).

(2) For the considered goal, the set of AND-combination of leaf obstacles obstructing this goal is then computed; this set is called the *obstruction superset*. (Section 4.2.1.2).

(3) Based on the estimated satisfaction rate of the leaf obstacles, the probability for the entailment and obstruction superset is computed. The satisfaction rate of an obstacle is computed from the probability for the corresponding entailment superset. The satisfaction rate of the goal is computed from the probability for the obstruction superset. (Section 4.2.1.3.)

An entailment/obstruction superset somewhat corresponds to the *cut set* of a fault tree [Bed01; Lam09]; Step 1 is similar. However, a cut set yields all combinations of leaf events causing the root event to occur, similarly to an entailment superset, whereas an obstruction superset yields all combinations of leaf obstacles causing the corresponding goal in the goal model to be obstructed. The propagation must therefore continue bottom-up through the goal model with specific propagation equations to assess the severity of obstruction consequences; Step 2 is not found in Fault Tree Analysis. Another difference is that the tree nodes here are formalizable goal/obstacle specifications, linked by entailment relationships among levels, rather than event labels.

4.2.1.1 Computing entailment sets

For a given obstacle O in the obstacle model, we need to compute all AND-combinations of leaf obstacles and domain properties/hypotheses that may entail O.

Definition 4.2. (Obstacle entailment set) An *entailment set* for an obstacle O is a set of obstacles O_i and domain properties/hypotheses DH_j such that this set entails the obstacle and is *consistent*, that is,

$$\{O_1, O_2, ..., DH_1, DH_2, ...\} \models O$$
$$\{O_1, O_2, ..., DH_1, DH_2, ...\} \not\models false$$

An entailment set OS for an obstacle O should be *minimal*, that is, all its elements are required for falsifying the goal:

$$\text{for all } O_i \text{ in } OS : OS \setminus O_i \not\models O$$
$$\text{for all } DH_j \text{ in } OS : OS \setminus DH_j \not\models O$$

An obstacle might have multiple alternative entailment sets.

Definition 4.3. (Entailment superset) The OR-combination of all alternative entailment sets for an obstacle O is called *entailment superset* for O. It is denoted by $ES(O)$.

The entailment sets in an entailment superset should ideally be disjoint; probability computation is simpler and the interpretation of unrelated causes is generally easier. Non-disjointness arises when entailment sets share common obstacles or domain hypotheses.

The entailment superset for an obstacle is computed by up-propagation through obstacle refinement tree from the leaf obstacles and domain properties/hypotheses in obstacle trees to their root obstacle.

Consider the obstacle AND/OR refinement tree anchored on a leaf goal LG in the goal model. To obtain the obstruction superset $OS(LG)$, we proceed by structural induction.

- For a leaf obstacle or a domain hypothesis LO:

$$ES(LO) = \{LO\}$$

- For an AND-refinement of O in subobstacles SO_1 and SO_2:

$$ES(O) = ES(SO_1) \times ES(SO_2),$$

  where $\times$ denotes the cartesian product over sets.

- For an OR-refinement of O in subobstacles SO_1 and SO_2:

$$ES(O) = ES(SO_1) \cup ES(SO_2).$$

(The generalization to refinements involving more subobstacles is straightforward.)

For example, the entailment superset for the obstacle Power Supply Failure in Figure 4.1 is {{Diesel Generator Failure, Primary Power Supply Down}}; The entailment superset for the obstacle No Power Available is {{Diesel Generator Failure, Primary Power Supply Down}, {Power Cabling Failure},{UPS Battery Failure}}. The entailment superset for the root obstacle Pump Motor Not On And Requested in Figure 4.2 contains the two extra entailment sets {Pump Mechanical Failure} and {Pump Electrical Failure}.

4.2.1.2 Computing obstruction sets

For a given goal G in the goal model, we need to compute all AND-combinations of leaf obstacles and domain properties/hypotheses that may obstruct G.

Definition 4.4. (Goal obstruction set) An *obstruction set* for a goal G is a set of obstacles O_i and domain properties/hypotheses DH_j such that this set obstructs the goal and is *consistent*, that is,

$$\{O_1, O_2, ..., DH_1, DH_2, ...\} \vDash \neg G$$
$$\{O_1, O_2, ..., DH_1, DH_2, ...\} \nvDash false$$

Similarily to obstacles, an obstruction set OS for a goal G should be *minimal*,

$$\text{for all } O_i \text{ in } OS : OS \setminus O_i \nvDash \neg G$$
$$\text{for all } DH_j \text{ in } OS : OS \setminus DH_j \nvDash \neg G$$

For example, the goal Achieve [Make Up Pump Motor On When Water Requested], shown in Figure 4.2, can be obstructed by the following four obstruction sets:

- {Pump Mechanical Failure},
- {Diesel Generator Failure, Primary Power Supply Down},
- {Power Cabling Failure},
- {UPS Battery Failure},
- {Pump Electrical Failure}.

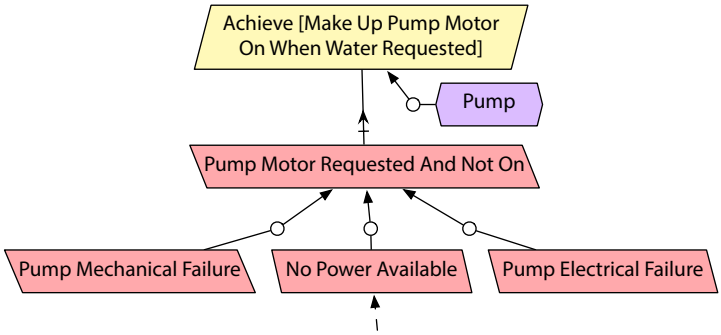


Figure 4.2 Obstacles for Achieve [Make Up Pump Motor On When Water Requested].

As the example shows, a goal might have multiple alternative obstruction sets.

Definition 4.5. (Obstruction superset) The OR-combination of all alternative obstruction sets for a goal G is called *obstruction superset* for G. It is denoted by $OS(G)$.

In the example, the obstruction superset for Achieve [Pump A Activated Until Measured Water Level Appropriate] is the set containing all obstruction sets above.

Like entailment sets, the obstruction sets in an obstruction superset should also ideally be disjoint. Non-disjointness arises when obstruction sets share common obstacles or domain hypotheses.

The obstruction superset for a goal is computed by up-propagation from the root obstacle to the corresponding obstructed leaf goals and finally from the leaf goals in goal trees to the considered higher-level goal.

The obstruction superset for a leaf goal LG obstructed by a root obstacle RO is given by its entailment superset:

$$OS(LG) = ES(RO)$$

In our example, the obstruction superset for the goal Achieve [Make Up Pump Motor On When Water Requested] is the entailment superset for the corresponding root obstacle Pump Motor Not On And Requested.

In the goal AND/OR refinement graph, the obstruction superset for a parent goal depends on the obstruction superset for its subgoals. For a parent goal PG AND-refined into subgoals SG_1 and SG_2 we have:

$$OS(PG) = OS(SG_1) \cup OS(SG_2)$$

(The generalization to AND-refinements with more subgoals is straightforward.)

We do not consider OR-refinement of goals here. OR-refinements of goals capture alternative systems; here we focus on a single system.

The obstruction sets in the superset thereby obtained are not necessarily disjoint. This arises from obstacle refinement trees sharing common obstacles, resulting in non-empty intersections of obstruction sets. Such dependencies must be taken into account when computing the probability of satisfaction of obstruction sets. This can be achieved automatically, see Section 4.2.1.3. Obstruction sets are minimal if goal and obstacle refinements are minimal.

Consider the goal refinement presented in Figure 4.3. The obstruction superset for the goal Achieve [Make Up Water Provided When Loss Of Cooling] may be computed from the obstruction supersets obtained for the subgoals Achieve [Make Up Water Requested When Loss Of Cooling] and Achieve [Make Up Water Provided When Requested]. Recursively, the obstruction superset for the goal Achieve [Make Up Water Requested When Loss Of Cooling] may be computed from the obstruction superset for the goals Achieve [Alarm Raised When Low Water] and Achieve [Make Up Water Requested When Alarm Raised].

4.2.1.3 Computing satisfaction rates from obstruction sets

The previous section showed how the entailment and obstruction superset can be obtained for a higher-level obstacle or goal. The satisfaction rate of a higher-level obstacle or goal is computed as follows:

(1) A binary-decision diagram (BDD) is built;

(2) Probability values are up-propagated through the BDD tree, from the leafs to the root node;

(3) The satisfaction rate of the higher-level obstacle or goal is computed from the probability value of the root node.

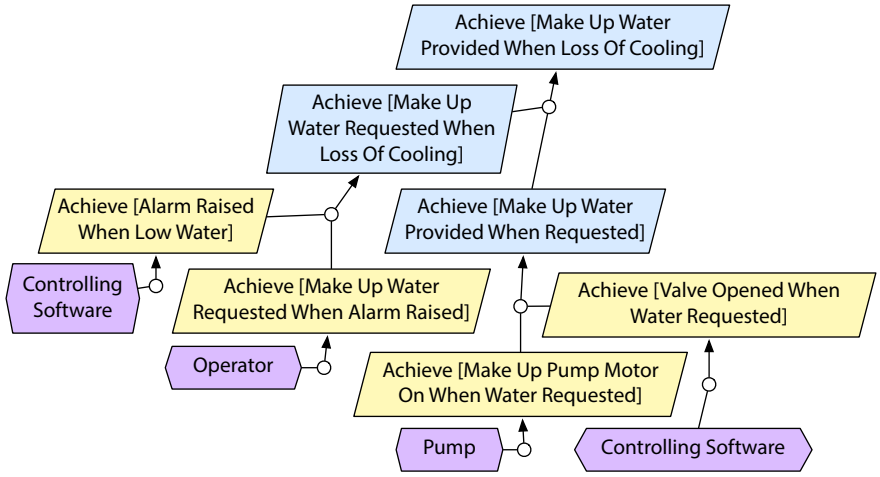


Figure 4.3 Refinement of Achieve [Make Up Water Provided When Loss Of Cooling].

Step 1: Building the BDD. To compute the probability of satisfaction of an entailment or obstruction superset next, a BDD is built that represents the corresponding Boolean formula where each leaf obstacle and domain hypothesis appears as a variable. This Boolean formula encodes the AND/OR-combination of leaf obstacles and domain hypotheses through the disjunction of the conjunction of elements in each obstruction set. As the ordered BDD is canonical, equivalent formulas result in the same BDD. This makes specific treatments of disjoint entailment or obstruction sets unnecessary; a superset with disjoint sets will result in the same BDD than an equivalent superset with non-disjoint sets.

In our example, consider the goal Achieve [Make Up Pump Motor On When Water Requested] (see Figure 4.1 and Figure 4.2). Its obstruction sets given before, correspond to the following Boolean formula:

$$\begin{aligned}&\text{Pump Mechanical Failure}\\ &\vee (\text{Diesel Generator Failure} \wedge \text{Primary Power Supply Down})\\ &\vee \text{Power Cabling Failure}\\ &\vee \text{UPS Battery Failure}\\ &\vee \text{Pump Electrical Failure}\end{aligned}$$

Efficient algorithms are available for building compact BDDs [Ebe05; Mei12]. Such BDDs enable fast computation of single-point probability values from single-point probability values for the variables [Bed01]. Figure 4.4 shows a BDD corresponding to the preceding formula. Each node represents a leaf obstacle. By following the solid edge if the corresponding leaf obstacle is satisfied or the dotted edge otherwise, we can determine whether the corresponding Boolean formula is *true* or *false*. The terminal nodes indicate whether the formula is satisfied (1) or not (0). For example, if Diesel Generator Failure and Primary Power Supply Down are both satisfied, the obstruction set is satisfied since the BDD path ends at node (1).

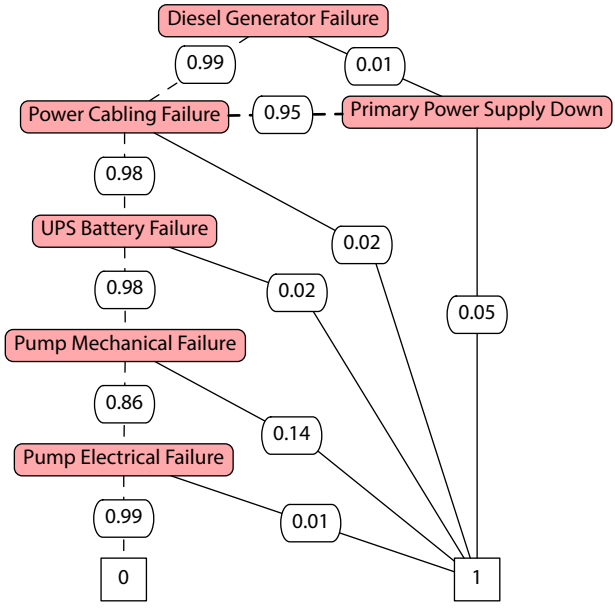


Figure 4.4 BDD for the obstruction set of Achieve [Make Up Pump Motor On When Water Requested].

Step 2: Propagating probability values. Every edge in the BDD has a probability label. The probability label for a solid edge corresponds to the estimated satisfaction rate of the leaf obstacle at its source; for a dotted edge, the probability label corresponds to 1 minus this probability.

The probability of a non-terminal node captures the probability that the formula corresponding to the subtree is satisfied. It is given by the product of the probability on the edge and the probability of the target node. For a terminal node, the probability is given by its value (0 or 1).

Obstacle	SatRate	1 - SatRate
Diesel Generator Failure	0.01	0.99
Primary Power Supply Down	0.05	0.95
Power Cabling Failure	0.02	0.98
UPS Battery Failure	0.02	0.98
Pump Mechanical Failure	0.14	0.86
Pump Electrical Failure	0.01	0.99

Table 4.1 Satisfaction rate for obstacles.

For example, let us consider the satisfaction rates summarized in Table 4.1. Based on these values, the probability of the node Pump Electrical Failure is given by:

$$P_{PumpElectricalFailure} = 0.99 \times 0 + 0.01 \times 1 = 0.010$$

The probability of the node Pump Mechanical Failure is:

$$P_{PumpMechanicalFailure} = 0.86 \times 0.010 + 0.14 \times 1 = 0.149$$

Similarily, we compute the probability for the nodes UPS Battery Failure, Power Cabling Failure and Primary Power Supply Down:

$$P_{UPSBatteryFailure} = 0.98 \times 0.149 + 0.02 \times 1 = 0.166$$
$$P_{PowerCablingFailure} = 0.98 \times 0.166 + 0.02 \times 1 = 0.182$$
$$P_{PrimaryPowerSupplyDown} = 0.95 \times 0.182 + 0.05 \times 1 = 0.223$$

We finally obtain the probability for the root node Diesel Generator Failure:

$$P_{DieselGeneratorFailure} = 0.99 \times 0.182 + 0.01 \times 0.223 = 0.183$$

Step 3: Computing satisfaction rate. The satisfaction rate of an obstacle O is given by the probability that its entailment superset is satisfied:

$$P(O) = Pr[ES(O)],$$

where $Pr[ES(O)]$ denotes the probability that the entailment superset $ES(O)$ is satisfied.

The satisfaction rate of a goal G is given by the probability that its obstruction superset is not satisfied:

$$P(G) = 1 - Pr[OS(G)],$$

where $Pr[OS(O)]$, respectively $Pr[OS(G)]$, denotes the probability that the obstruction superset $OS(O)$, respectively $OS(G)$, is satisfied.

In our example, we derive the probability of satisfaction for the goal Achieve [Make Up Pump Motor On When Water Requested] from the probability value for the root node of the BDD, that is,

$$P(\text{Achieve [Make Up Pump Motor On ...]}) = 1 - 0.183 = 81.7\%$$

4.2.2 Pattern-based computation of satisfaction rates

The BDD-based approach in the previous section does not support weak entailment and weak obstructions. As an alternative approach to propagation, we may compute the satisfaction rate of the obstacle or goal at each propagation step. Similarly to the approach presented in the previous section, this alternative procedure propagates the satisfaction rate from leaf obstacles to root obstacles, then from root obstacles to obstructed leaf goals, and finally from leaf goals to root goals. As opposed to the previous approach, obstruction supersets are not computed here.

Step 1: From leaf obstacles to root obstacles. For an obstacle O refined through two subobstacles SO_1, SO_2, the satisfaction rate of the parent obstacle is given by:

$$P(O) = P(SO_1) \times P(SO_2) \times P(O \mid SO_1, SO_2)$$

The term $P(O \mid SO_1, SO_2)$ captures weak entailment, as the combination of the subobstacles might not entail O (See Definition 3.30). This term has to be estimated by domain experts and annotate the refinement.

An obstacle OR-refined through two subobstacles is satisfied if one of them is satisfied. Therefore, an obstacle OR-refined through two subobstacles is *not* satisfied if all subobstacle are *not* satisfied. For a OR-Refinement of the obstacle O into two subobstacles SO_1, SO_2 the satisfaction rate of O is given by:

$$\begin{aligned} P(O) = 1 - \; & [1 - P(SO_1) \times P(O \mid SO_1)] \\ & \times [1 - P(SO_2) \times P(O \mid SO_2)] \,, \end{aligned}$$

where the terms $P(O \mid SO_i)$ capture weak entailment.

Step 2: From root obstacles to leaf goals. For an obstructed goal OG and the corresponding obstacle RO, the satisfaction rate is obtained using the following equation:

$$P(OG) = 1 - P(RO) \times P(\neg OG \mid RO),$$

where the term $P(\neg OG \mid RO)$ captures a partial obstruction.

Step 3: From leaf goals to top goals. In the most general case, the parent goal is satisfied if the two subgoals are satisfied, or if the first is satisfied and sufficient for satisfying the parent, or if the second is satisfied and sufficient for satisfying the parent. This leads to the following *general propagation equation for AND-refinements*:

$$\begin{aligned} P(G) = \; & P(SG_1, SG_2) \times P(G \mid SG_1, SG_2) \\ & + P(SG_1, \neg SG_2) \times P(G \mid SG_1, \neg SG_2) \\ & + P(SG_2, \neg SG_1) \times P(G \mid SG_2, \neg SG_1) \\ & + P(\neg SG_1, \neg SG_2) \times P(G \mid \neg SG_1, \neg SG_2) \end{aligned}$$

As we focus our attention on a single system, no alternative OR-refinements are to be considered. The satisfaction rate of a goal G given that none of the subgoals is satisfied is then equal to zero; thereby $P(G \mid \neg SG_1, \neg SG_2) = 0$. Moreover, if the refinement is complete, we have

$$P(G \mid SG_1, SG_2) = 1$$

The AND-propagation equation then reduces to:

$$\begin{aligned} P(G) = P(SG_1, SG_2) \\ + P(SG_1, \neg SG_2) \times P(G \mid SG_1, \neg SG_2) \\ + P(SG_2, \neg SG_1) \times P(G \mid SG_2, \neg SG_1) \end{aligned}$$

wher the terms $P(G \mid ...)$ capture that a single subgoal may entail the parent goal. Such probabilities are to be estimated by domain experts and annotate the refinement.

Depending on the type of refinement and goal, this propagation equation can be made further specific. Table 4.2 gives propagation equations for a sample of common refinement patterns known to be complete, consistent and minimal [Dar95].

Refinement Pattern	**Propagation Rule**
Milestone-driven	$P(G) = P(SG_1) \times P(SG_2)$
Case-driven	$P(G) = P(CS) \times P(SG_1) + (1–P(CS)) \times P(SG_2)$
Guard introduction	$P(G) = P(SG_1) \times P(SG_2) \times P(SG_3)$
Divide-and-conquer	$P(G) = P(SG_1) \times P(SG_2)$
Unmonitoribility-driven	$P(G) = P(SG_1) \times P(SG_2)$
Uncontrollability-driven	$P(G) = P(SG_1) \times P(SG_2)$

Table 4.2 Propagation equations for common goal refinement patterns.

For example, for a *milestone-driven* refinement [Dar95], the satisfaction of a single milestone-based subgoal is not sufficient for satisfying the parent goal. The propagation equation therefore reduces to:

$$P(G) = P(SG_1) \times P(SG_2)$$

However, for a *case-driven* refinement [Dar95], the parent goal is satisfied when one of the subgoals is satisfied. The probability of satisfying the case condition CS, denoted $P(CS)$, equals $P(G \mid SG_1, \neg SG_2)$. Therefore, assuming two disjoint cases, the propagation equation becomes:

$$P(G) = P(CS) \times P(SG_1) + (1 - P(CS)) \times P(SG_2)$$

The specific simplification of the general propagation equation thus depends on the goal refinement pattern being used. This information is available in the annotation of the refinement node [Lam09].

4.2.3 Discussion

This section compares the approaches proposed in the two previous sections for computing the satisfaction rate of a high-level goal in the goal model.

The propagation procedure using obstruction supersets supports non-disjoint goals and obstacles. The pattern-based propagation procedure does not. In the pattern-based propagation, the computation of satisfaction rate of a goal or an obstacle only depends on the refinees whereas in the BDD-based propagation the obstruction supersets also capture the refinement structure.

For example, consider the goal model fragment in Figure 4.5. The satisfaction rate of the obstacle No Power Available is 0.040. With the pattern-based propagation procedure, the satisfaction rate of both subgoals is 0.960. The satisfaction rate of the parent goal Achieve [Make Up Water Provided When Requested] is computed using the specialized propagation equation for a *divide-and-conquer* refinement [Dar95], leading to $0.96 \times 0.96 = 0.921$. This however counts the obstacles twice. In contrast, the obstruction superset for the parent goal is {{No Power Available}}. The probability to satisfy the obstruction superset is 0.04. The satisfaction rate of the goal is therefore 0.96.

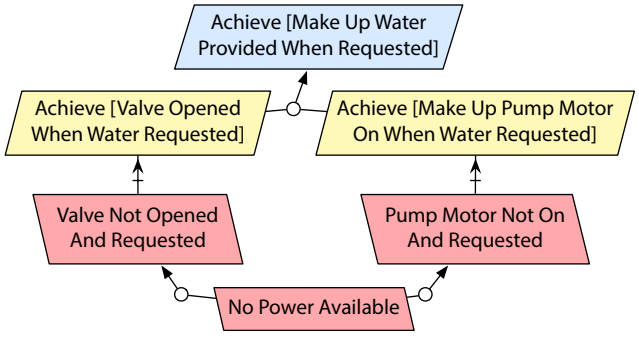


Figure 4.5 Goal/obstacle model fragment for Achieve [Make Up Water Provided When Requested].

The pattern-based propagation procedure appears to be finer-grained as it supports weak entailments and weak obstructions. These need however to be elicited from domain experts. Using more specific propagation equation leads to more precise estimates, in particular when the refinement follows the *by-case* refinement pattern [Dar95].

For example, consider the (hypothetical) refinement in Figure 4.6. Given disjoint obstruction sets, the satisfaction rate of the goal Maintain [Water Level Above LOW] will equal the product of the satisfaction rate of the subgoals. However, the refinement with only one subgoal satisfies the partial entailment condition. In other words, the probability that Maintain [Water Level Above LOW] is satisfied is not 0 when only Achieve [Water Level Above LOW When Alarm Raised] is satisfied. Using the specific propagation equation gives a satisfaction rate for the parent goal that will take the weak entailment into account, resulting in an higher satisfaction rate.

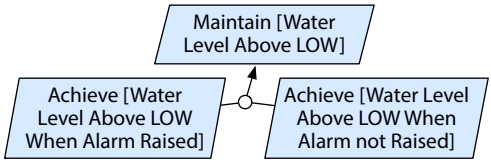


Figure 4.6 By-case refinement for Maintain [Water Level Above LOW].

Note that the pattern-based procedure might return erroneous results when obstacles are shared by multiple goals. For instance, if an obstacle is shared by two leaf goals with a common ancestor, the pattern-based propagation will account for the obstacle twice for that common ancestor; the BDD-based propagation will account for the obstalce only once. When no obstacle is shared, the results yielded by both procedures are equal.

The main difference between the two propagation procedures is about *when* most of the computation occurs. With the BDD-based propagation technique, most of the computation cost occurs at the construction of the obstruction supersets for the considered goal. The computation of the probability given an obstruction superset is very cheap in comparison. With the pattern-based approach, the cost cannot be split in two smaller procedures. In particular,

- As it will be seen in Chapter 6, the BDD-based approach also enables fast computation of probability distributions. To obtain probability distributions, we compute many obstruction superset probabilities on the same obstruction superset, taking advantage of the separation in two steps.

- As it will be seen in Chapter 7, the BDD-based procedure also enables the runtime monitoring of satisfaction rates as the obstruction superset is not computed at runtime.

On a randomly generated model with 10.000 goals and 10.000 obstacles (with 1000 obstruction links), the pattern-based computation takes about 13 seconds, whereas the BDD-based computation takes about 18 seconds. However, computing the probability of the obstruction superset takes only 5 milliseconds. Each experiment was performed 80 times and the observed variance was very small (below 60 ms for the two propagations, and below a tenth of a millisecond for the probability computation). All benchmarks were performed using *BenchmarkDotNet* [BDNET] on an Apple MacBook Pro with a 3 GHz Intel Core i7, 16 GB of 1600 MHz DDR3 Ram, and an Apple SSD drive.

In conclusion, the BDD-based procedure appears superior to the pattern-based procedure as it is more efficient, thanks to the pre-computation of the BDD and to the simpler equations; it is more general, as it does not depend on specific refinement patterns; and it requires less input from domain experts, as weak entailment and weak obstructions are not considered. The pattern-based procedure should be used only when a single computation is required, no obstacles is shared between goals and weak entailment or weak obstructions are found in the model.

4.3 Identifying most likely and critical obstacles

This section shows how likely and critical risks are pointed out based on the estimated satisfaction rates for leaf obstacles (see Section 4.1) and the satisfaction rate of the obstructed high-level goals, as computed by one of the two propagation procedures (see Section 4.2). These likely and critical obstacles should be resolved in the next phase of obstacle control.

The criticality of an obstacle is accordingly determined as the loss in satisfaction rate of some important high-level goals from the goal model. In our running example, the obstacle UPS Battery Failure causes, alone, a loss of 0.011 for the high-level goal Achieve [Make Up Water Provided When Loss Of Cooling].

An obstacle is said to be *critical* for a goal if it causes the satisfaction rate of this goal to fall below the required satisfaction rate.

Definition 4.6. (Critical obstacle) A probabilistic obstacle O is *critical* for a goal G if obstacle O results in a goal satisfaction rate such that $P(G) < RSR(G)$, i.e. $SV(G) > 0$. (As a recall, $SV(G) = RSR(G) - P(G)$, see Definition 3.11.)

An obstacle is said *acceptable* for a goal if it results in a satisfaction rate for this goal higher than its required threshold.

Definition 4.7. (Acceptable obstacle) A probabilistic obstacle O is *acceptable* for a goal G if obstacle O does not result in a goal satisfaction rate such that $P(G) < RSR(G)$.

To evaluate obstacle criticality, we may proceed in two ways:

- **Global impact analysis:** the satisfaction rate of *all* leaf obstacles are together propagated bottom-up in the goal graph to see how much the resulting satisfaction rate of higher-level goals deviates from their required satisfaction rate.

- **Local impact analysis:** the consequence of a *single* leaf obstacle is evaluated by up-propagation of the satisfaction rate of this leaf obstacles, all other leaf obstacles being assigned a probability of 0 (meaning that they are all assumed to never happen).

In-between, we can evaluate the consequences of a *combination* of leaf obstacles. By iterating on the size of the combinations, we move from local impact analysis to global impact analysis. A set of obstacle is *critical* if the obstacles together causes the violation severity to drop below 0.

For example, global impact analysis reveals that the root goal Achieve [Make Up Water Provided When Loss Of Cooling] is satisfied in 80.31% of cases when all obstacles are considered. Local impact analysis reveals that the obstacle UPS

Battery Failure causes the satisfaction rate of the root goal Achieve [Make Up Water Provided When Loss Of Cooling] to fall from 1 to .989. With a required satisfaction rate of .99, the obstacle UPS Battery Failure is critical for Achieve [Make Up Water Provided When Loss Of Cooling].

The most critical obstacle for a goal is the obstacle that maximize the violation severity.

Definition 4.8. (Most critical obstacle) A probabilistic obstacle O is a *most critical obstacle* for a goal G if obstacle O is critical and no other obstacle results in a higher violation severity $SV(G)$.

A set of obstacle is *a most critical set of obstacles* if no other set of obstacles of the same size causes a higher violation severity.

Obstacle assessment can be presented using a *violation diagram*. A violation diagram somewhat correspond to a risk matrix [Ayy14]; A risk matrix groups risk likelihood and criticality in discrete categories. In a violation diagram, the risks are represented in a two-dimensional plot showing the violation severity for a high-level goal and probability of occurrence of a leaf obstacle combination.

In a violation diagram, the x-axis represents the probability of leaf obstacle combination occurrence; it is the product of the estimated satisfaction rate of the leaf obstacles in the combination. The y-axis captures the violation severity for the goal considering the leaf obstacles in the combination.

Figure 4.7 shows a violation diagram for the goal Achieve [Make Up Water Provided When Loss Of Cooling] with the leaf obstacles from the obstacle model; here, the combinations contains a single leaf obstacle. We can see that most of the obstacles taken alone results in a violation severity equal to 0; these are therefore not critical for the system. Three obstacles are however critical, namely Pump Mechanical Failure, UPS Battery Failure, and Power Cabling Failure. These critical obstacles should be resolved in the next phase.

Criticality assessment should be an iterative process. Once most likely and critical obstacle combinations of size i are resolved, a new violation diagram should be produced with obstacle combinations of size $i + 1$. The process ends when no obstacle combination is found critical and likely.

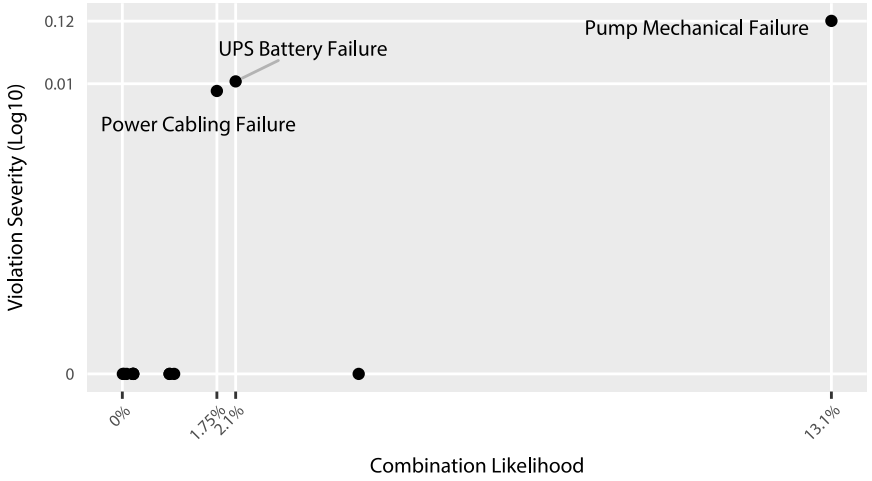


Figure 4.7 Violation Diagram for Achieve [Make Up Water Provided When Loss Of Cooling].

Figure 4.8 shows a violation diagram for the goal Achieve [Make Up Water Provided When Loss Of Cooling] considering pairs of obstacles, i.e. combinations of size 2. Among the 91 pairs of leaf obstacles, 39 are critical; there are 36 pairs that contains a critical obstacle identified with the previous violation diagram in Figure 4.7. Therefore, there are 3 pairs of leaf obstacles that are critical and do not contain an obstacle critical alone. Those are (Pump Electrical Failure and Alarm Not Raised And Low Water), (Pump Electrical Failure, Valve Mechanical Failure), and (Pump Electrical Failure, Valve Electrical Failure). In our experience, we observed that most of the likely and critical obstacle combinations are caused only by a subset of these.

There is a multi-criteria optimization problem here as we are looking for minimal sets of leaf obstacles that maximize the severity of goal violations. To solve the optimization problem, we can naïvely generate all possible leaf obstacle combinations. The violation severity $SV(G)$ is then computed for the root goal G. Most critical combinations are identified by sorting the leaf obstacle combinations by violation severity.

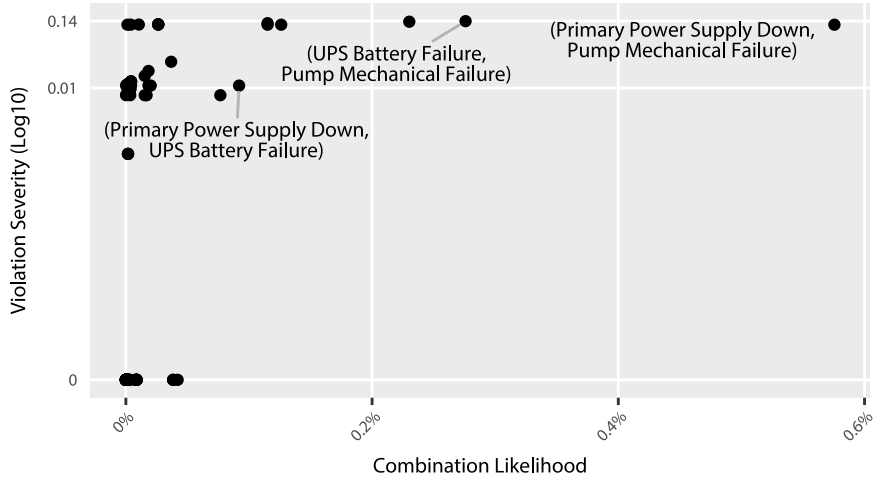


Figure 4.8 Violation Diagram for Achieve [Make Up Water Provided When Loss Of Cooling] with pair of obstacles.

Note that a 'most critical obstacle combination' is *Pareto efficient* for the size of the combination; that is no other combination cause a higher violation severity. The set of leaf obstacle combinations that maximizes the severity defines a *Pareto front*; it is the set containing the most critical obstacle combinations. Efficient algorithms for generating *Pareto fronts* are available [Bor01; Kun75]. These algorithms and the specific structure of our problem suggests that our generation of leaf obstacle combinations and their ranking by severity could thereby be optimized in order to scale up for larger systems.

The techniques presented in this chapter consider a single high-level goal only. In practice, several high-level goals might be competing. In such situations, the satisfaction rate of these high-level goals are to be combined using a normalized weighted sum in order to apply the techniques as if there was only a single aggregated high-level goal. The weights in this aggregation correspond to the relative priority of those high-level goals.

4.4 Summary

This chapter showed how likely and critical risks are pointed out to focus the resolution step on important obstacle combinations. The satisfaction rates of leaf obstacles are first estimated by domain experts, possibly multiple ones to be combined in order to produce more accurate estimates. The leaf estimates are then up-propagated through the obstacle and goal models to compute the satisfaction rate of important high-level goals. The chapter presented two propagation techniques: a BDD-based approach and a pattern-based one. The BDD-based approach is more efficient, more general and requires less input from domain experts; the Pattern-based approach provides finer-grained estimates. The computed satisfaction rate of the high-level goals is then compared with their respective required satisfaction rates. Obstacles causing the computed satisfaction rate to fall below the required satisfaction rate of these high-level goals are the critical obstacles to be resolved in the next phase of risk control. Violation diagrams captures likely and critical obstacle combinations.

The techniques presented in this chapter provides a basis for determining most appropriate countermeasures, as seen in the next chapter.

Chapter 5

Controlling Likely and Critical Obstacles

The identified and assessed obstacles need be resolved. Likely and critical ones have to be cost-effectively controlled through appropriate countermeasures yielding new goals to be integrated in the goal model. Alternative countermeasures are identified first; appropriate ones are selected next. As the countermeasures are integrated in the original goal model, the software requirements will be more complete in covering unexpected through possible situations.

This chapter details what most appropriate countermeasure goals are, how cost-effective selection of countermeasures goals may be determined, and how selected countermeasure goals are integrated in the original model.

An appropriate countermeasure selection ensures that the satisfaction rate of high-level goals becomes higher than their required satisfaction rate while maintaining a correct model. A *valid* countermeasure guarantee progress—once integrated in the model, the satisfaction rate of some high-level goals has to increase. The chapter introduces integration schemas guaranteing that only the required parts of the model are modified. As the model is being modified, the propagation of changes ensures that the model remains correct after integration. As a result, a correct and more complete model is obtained.

The chapter is organized as follows. Section 5.1 details what a valid countermeasure is. Section 5.2 discusses how the need for a new cycle of obstacle analysis on the countermeasure goals is determined. Section 5.3 details how countermeasures are assessed to be most appropriate and selected. Section 5.4 describes how countermeasures are integrated into the goal model.

5.1 Valid countermeasures

Likely and critical obstacles should be controlled through appropriate countermeasures [Lam09]. The latter are identified using heuristics such as *Avoid obstacle*, *Reduce obstacle likelihood*, *Mitigate obstacle*, *Weaken goal*, *Substitute goal*, *Restore goal*, or *Substitute agent* [Lam00]; some may be automatically produced using the theory revision techniques [Alr16]. As mentioned in Section 2.2.4, countermeasure goals are specified by LTL assertions that may capture timing conditions on their application.

A *valid* countermeasure should ensure that its integration increases the satisfaction rate of some high-level goal.

Definition 5.1. (Countermeasure integration) The *integration of a countermeasure goal* CG to a leaf obstacle LO in goal model M, denoted $Int_{CG,LO}(M)$, maps M to a new model M' such that $CG \in M'$.

Integrating a countermeasure may requires to de-idealize goals; those need be validated by domain experts.

Definition 5.2. (De-idealized goal) A goal G' is a *de-idealized* version of goal G if $G \models G'$.

The application of the integration operator should increase the probability of satisfaction of some root goal at least, that is,

Definition 5.3. (Progress) An integration $Int_{CG,LO}(M)$ guarantee *progress* if and only if there is at least one root goal RG' in M' corresponding to RG in M such that

$$P_{M'}(RG') > P_M(RG)$$

where $P_M(G)$ refers to the satisfaction rate of G in model M.

This progress property is guaranteed by valid countermeasures; Precisely, a *valid countermeasure* CG for a leaf obstacle LO satisfies one of the following conditions:

$$\{LO', Dom'\} \not\models \neg OG' \qquad \text{(non-obstruction)}$$
$$\{LO', Dom'\} \models false \qquad \text{(domain-inconsistency)}$$

where, given that $M' = Int_{LO,CG}(M)$,

- OG' is the obstructed goal in the model M' corresponding to OG,
- LO' is the leaf obstacle in the model M' corresponding to LO,
- Dom' is the set of domain properties in the model M'.

The first condition may be relaxed in order to remove the obstruction on a higher-level goal AG rather than on the obstructed leaf goal LO, as in the *Strong mitigation* resolution tactic [Lamoo]:

$$\{LO', Dom'\} \not\models \neg AG' \qquad \text{(ancestor non-obstruction)}$$

where AG' is the ancestor in M' corresponding to AG, such that AG is an ancestor of the obstructed leaf goal OG.

These conditions might be generalized for probabilistic goals and obstacles:

$$P(\neg OG' \mid LO', Dom') < 1 \qquad \text{(non-obstruction)}$$
$$P(\neg AG' \mid LO', Dom') < 1 \qquad \text{(ancestor non-obstruction)}$$
$$P(LO', Dom') = 0 \qquad \text{(domain-inconsistency)}$$

The two conditions hereabove may appear weak for probabilistic goals. For example, the first condition might be satisfied even if the satisfaction rate of the obstructed goal does not increase. Those two conditions may be strengthened as follows:

$$P(\neg OG' \mid LO', Dom') < P(\neg OG \mid LO, Dom) \qquad \text{(strong non-obstruction)}$$
$$P(\neg AG' \mid LO', Dom') < P(\neg AG \mid LO, Dom) \qquad \text{(strong ancestor non-obstruction)}$$

The *domain-inconsistency* might however be relaxed to capture partial reduction:

$$P(LO', Dom') < P(LO, Dom) \qquad \text{(weak domain-inconsistency)}$$

Definition 5.4. (Valid countermeasure) A *valid countermeasure* CG for a probabilistic leaf obstacle LO satisfies the *strong ancestor non-obstruction*.

Note that if a countermeasure goal satisfies the *weak domain-inconsistency*, it satisfies *strong non-obstruction*; an decrease in the satisfaction rate of an obstacle results in an increase of the satisfaction rate of the obstructed goal. If a countermeasure goal satisfied the *strong non-obstruction*, it trivially satisfies the *weak ancestor non-obstruction*.

A *valid countermeasure* is a sufficient condition for guaranteeing an increase in the satisfaction rate of some root goal.

Proposition 5.1. (Valid countermeasures guarantee progress) The integration of a valid countermeasure guarantee progress.

Proof. A valid countermeasure is such that its integration causes an increase in the satisfaction rate of the obstructed goal (*strong non-obstruction*), an increase in the satisfaction rate of an ancestor of the obstructed goal (*strong ancestor non-obstruction*), or a decrease in the satisfaction rate of the leaf obstacle (*weak domain inconsistency*). Therefore, the satisfaction rate of some root goal increases:

- In a correct goal refinement, an increase in the satisfaction rate of a children goal results in an increase in the satisfaction rate of the parent goal. Otherwise, the goal refinement is not minimal.

- An decrease in the satisfaction rate of a root obstacle results in an increase in the satisfaction rate of the corresponding obstructed leaf goal. Otherwise, the refinement of the root obstacle is not domain-complete.

- In a correct obstacle refinement, a decrease in the satisfaction rate of a children obstacle results in a decrease in the satisfaction rate of the obstacle goal. Otherwise, the obstacle refinement is not minimal. □

Obstacle resolution tactics such as *Avoid obstacle*, *Reduce obstacle likelihood*, *Mitigate obstacle*, *Weaken goal*, *Substitute goal*, *Restore goal*, or *Substitute agent* [Lamoo; Lamo9] can be shown to produce valid countermeasures modulo propagations of their effect through the refinement trees in which they are involved (such propagation will be explained in Section 5.4).

For example, the countermeasure goal Achieve [Redundant Pump Motor On When Primary Pump Failure] to the obstacle Pump Electrical Failure, shown in Figure 5.1, is valid. The obstacle Pump Electrical Failure no longer obstruct Achieve [Make Up Water Provided When Requested]; the *strong ancestor non-obstruction* condition is satisfied. This countermeasure together with Achieve [Valve Opened When Water Requested] guarantee the satisfaction of the parent goal Achieve [Make Up Water Provided When Requested]. In this example, the parent goal is not de-idealized.

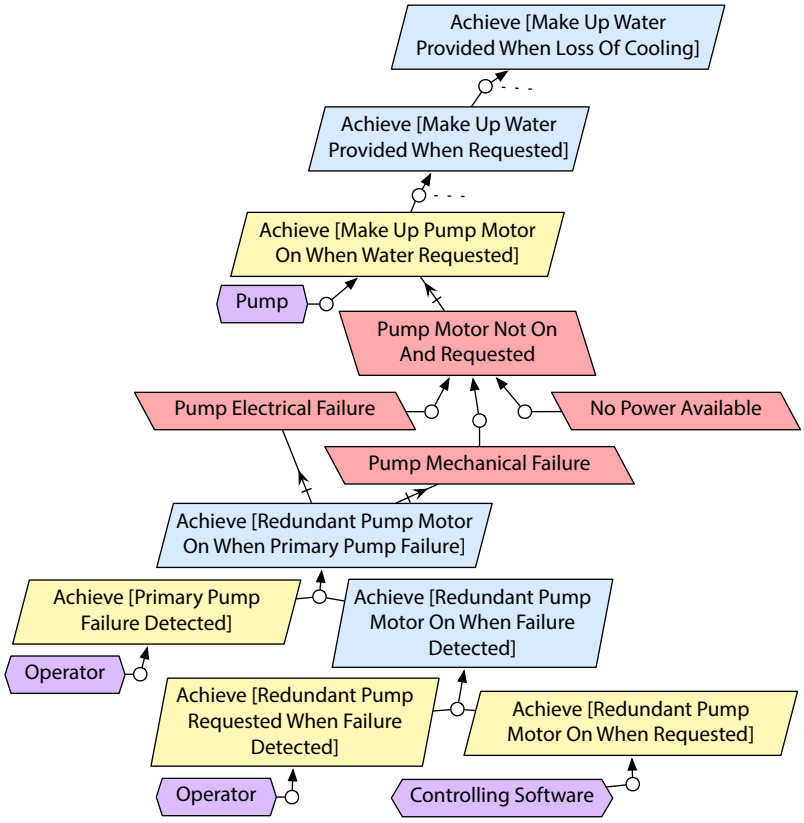


Figure 5.1 A countermeasure goal to the obstacle Pump Electrical Failure.

Multiple candidate ancestors might be considered for the *strong ancestor non-obstruction* condition. In our example, Achieve [Make Up Water Provided When Requested] and Achieve [Make Up Water Provided When Loss Of Cooling] are potential candidates with respect to the countermeasure goal Achieve [Redundant Pump Motor On When Primary Pump Failure].

The nearest candidate to this goal in the goal graph appears preferable to ensure more local model changes—that is, in our example the goal Achieve [Make Up Water Provided When Requested]. The lower ancestor the more local the change. The ancestor of the anchor goal may need be de-idealized, but all the descendants of the anchor goal are more likely to be de-idealized by the propagation, as seen in Section 5.4.1. The lower the anchor goal, the fewer goals may need modification.

Definition 5.5. (Anchor goal) The *anchor goal* of a countermeasure goal CG is the lowest ancestor goal AG meeting the *strong ancestor non-obstruction* condition.

As discussed in Section 5.4, the anchor goal of a coutermeasure goal is the goal through which the countermeasure goal is integrated.

5.2 Iterating obstacle analysis

As Figure 5.1 shows, a countermeasure goal should itself be refined down to leaf goals assignable to single agents. These leaf goals might be obstructed by new obstacles; a new cycle of obstacle analysis may be therefore needed.

To determine whether a new obstacle analysis cycle is required, the maximal change in satisfaction rate of the considered high-level goals once the countermeasure goal is integrated into the goal model should be taken into account.

- At best, the satisfaction rate of the countermeasure goal is 1 (no possible obstruction of it). By up-propagation, this satisfaction rate leads to a satisfaction rate sr_1 for the high-level goal.
- At worst, the satisfaction rate is 0. This leads to another satisfaction rate sr_0 for the high-level goal.

The countermeasure's *maximal impact* is then obtained by taking the difference:

$$sr_1 - sr_0.$$

Based on this, the analyst may decide at RE time whether a new cycle is needed or not.

For example, consider the countermeasure goal Achieve [Redundant Pump Motor On When Primary Pump Failure]. If the satisfaction rate of this countermeasure is 0, the satisfaction rate of the high-level goal Achieve [Make Up Water Provided When Loss Of Cooling] is 81.74%. If the satisfaction rate of this countermeasure is 1, the satisfaction rate of the high-level goal is 94.94%. The maximal impact on the high-level goal is thus 13.20%. As the impact is felt important, it may be worth spending time in studying obstacles to this countermeasure goal at RE time to better estimate the impact of its deployment.

5.3 Selecting valid countermeasures to obstacle

Thanks to the *progress* property introduced before, a valid countermeasure integrated into the goal model results in an increase of the satisfaction rate of some high-level goal. This quantifies the impact of a countermeasure; it is the key for selecting the most appropriate countermeasures; the latter are maximizing such impact.

Section 5.3.1 describes how the impact of alternative countermeasures is assessed. Section 5.3.2 clarifies what most appropriate countermeasures are to be selected and deployed when an obstacle resolution is required. Section 5.3.3 explains how such selection is systematically determined.

5.3.1 Assessing countermeasures impact

The integration of a countermeasure goal into the goal model impacts the satisfaction rate of a high-level goal. As Section 5.4 will show, depending on the specific obstacle resolution strategy and associated countermeasure integration schema being chosen, the obstructed goal is either removed from the goal model or kept. In the former case, the obstructed goal is replaced by the countermeasure goal; in the latter case, a new refinement is introduced involving both

the obstructed goal and the countermeasure goal. The satisfaction rate of the high-level goal is computed as described next to evaluate the impact of the countermeasure.

- *Assessing countermeasure impact when the obstructed goal is replaced.* In this case, when the satisfaction rate of the high-level goal is computed as described in Section 4.2, the propagation procedure computing the satisfaction rate of a high-level goal need to use the satisfaction rate of the countermeasure goal in place of the satisfaction rate of the replaced goal.

  For example, the goal Achieve [Make Up Water Automatically Requested When Alarm Raised] was generated using the *Agent Substitution* strategy [Lamoo]. When integrated, it replaces the goal Achieve [Make Up Water Requested When Alarm Raised]. The computation of the satisfaction rate of the parent goal Achieve [Make Up Water Requested When Loss Of Cooling] uses the satisfaction rate of this countermeasure goal rather than the satisfaction rate of the replaced goal. When integrated, the satisfaction rate of the top goal increases from 81.74% to 81.77%.

- *Assessing countermeasure impact when the obstructed goal is kept.* In this case, the anchor goal is refined into two subgoals; one subgoal is the obstructed goal conjoined with the negation of the obstacle condition; the other subgoal is the countermeasure goal. The satisfaction rate of the anchor goal is obtained using the satisfaction rates of those two subgoals, as described Section 4.2.

  For example, integrating the countermeasure goal Achieve [Redundant Pump Motor On When Primary Pump Failure] causes the anchor goal to be refined through two subgoals: Achieve [Make Up Water Provided When Requested And No Pump Failure] and Achieve [Redundant Pump Motor On When Primary Pump Failure]. The satisfaction rate of the root goal increases from 81.74% to 94.94%: The down-propagation of changes in the goal graph reduces the formal specification of the obstacles Pump Electrical Failure and Pump Mechanical Failure to *false*; The satisfaction rate of the latter is therefore 0.

Such computation is very easy when specific notations for goal exception and goal replacement are used, see Section 5.4 below. The resolutions are added and removed to evaluate the impact of the countermeasure.

This however requires the BDD to be built for each combination, which slows down the process. Improvements can be envisioned as one could build a single BDD covering all possible combinations using extra variables to encode which resolution is selected; Such BDD could then be computed once at RE-time.

5.3.2 What are most appropriate countermeasures?

The selection among alternative countermeasure that were identified should increase the satisfaction rate of high-level goals beyond their Required Satisfaction Rate (RSR) at reasonable cost.

Definition 5.6. (Safe Selection) A *safe selection* of countermeasures is a set of countermeasures that, once integrated, guarantees that the satisfaction rate of the high-level goals considered is greater than their respective RSR.

A selection of countermeasures must be consistent; conflicting countermeasures may not be selected together:

$$\{CM_i, CM_j, Dom\} \models false \text{ for } i \neq j \qquad \text{(selection-consistency)}$$

Countermeasures goal are to be refined until their leaf subgoals are assignable to single agents. They might share common subgoals and/or have conflicting subgoals. Such positive and negative interactions among countermeasure goals are not further considered in the thesis; their handling deserves future work.

Countermeasures come with a cost. The latter are elicited with regard to the countermeasure's contributions to soft goals in the goal model (See Section 2.1.6) such as Maximize[Operator Safety], Minimize[Power Consumption] or Maximize[Speed Computation]. Without loss of generality, only single costs are considered here. Multiple costs can be combined by a weighted sum where weights correspond to the priorities of the considered soft goals.

Definition 5.7. (Countermeasure selection resolution cost) The *resolution cost* of a selection is the sum of the costs associated with the selected countermeasures.

Definition 5.8. (Cost-optimal selection) A selection is *cost-optimal* if no other selection increases the probability of satisfaction of the high-level goal without increasing the resolution cost.

The selection of most appropriate countermeasures is defined as the one having the lowest resolution cost such that the satisfaction rate of the high-level goals is maximized while being safe, consistent, and cost-optimal.

5.3.3 Selecting most appropriate countermeasures

The selection of most appropriate countermeasures thus amounts to solving two optimization problems:

(1) Finding the minimal cost for guaranteeing the RSR of the considered high-level goals.

We may iteratively generate all possible selections and keep the selection that minimizes cost while guaranteeing that the RSRs are met. The complexity of this naïve approach is $O(2^n)$ where is n the number of countermeasures.

(2) Finding the selections that maximize the satisfaction rate of the high-level goals given this cost.

We may then generate all possible selections, compute their cost, and keep the selection with a minimal cost and the largest satisfaction rate of the high-level goals. The complexity of this naïve computation is $O(2^n)$ for n countermeasures.

In our example, the minimal cost for guaranteeing the RSR of 95% for the goal Achieve [Make Up Water Provided When Loss Of Cooling] is 2. With 6 possible resolutions, there are 64 possible selections. Among them 24 are safe selections that guarantee the RSR with costs ranging from 2 to 6 (we used unitary

costs for the countermeasure goals). A single selection of two countermeasure goals maximizes the satisfaction rate of Achieve [Make Up Water Provided When Loss Of Cooling], namely, Achieve [Cooling System Repaired] and Achieve [Power Supplied When Primary Power Supply Failure]. On this running example, the best satisfaction rate that can be achieve is 97.18%.

When no safe countermeasure selection is found, most appropriate countermeasures should be selected that maximize the satisfaction rate of the high-level goals.

The above procedure for selecting most appropriate countermeasures is based on exhaustive search. It should thus be improved. Obstacle-driven runtime adaptation, as considered in Chapter 7, requires timely selection; a naïve approach based on exhaustive search is likely to be too slow in practice (See Section 9.3.3). The above procedure could be improved by exploring the following directions:

- The problem of finding a cost-optimal selection shares similarities with the NP-hard Knapsack optimization problem [Cor09]. The latter is concerned with filling a bag with valued items without exceeding a maximal weight while maximizing value. The problem here differs in that the value and weight of items may not simply sum. Improvements of our naïve algorithm are expected as a pseudo-polynomial algorithm exists for the Knapsack problem [Cor09].

- Other techniques such as [Men03] might also improve selections. Incremental control learning is a technique that present, at each iteration, a set of key variables (in our case, countermeasure goals) that have the most impact on the system. This technique has already been applied to select risk mitigations in the DDP framework, showing promising results. (See Chapter 10 for a detailed discussion about DDP.)

- The Cross-Entropy method is an optimization technique for solving combinatorial problems [DeB05; Jeg12]. Applied to our problem, the method can be broken in two steps: *(1)* an initial selection is generated; *(2)* the parameters of the random generation are updated according the score of

the generated selection. The steps are run a large number of times. Such technique has been successfully applied for selecting software adaptations at runtime [Mor17a].

- The problem of selecting most appropriate countermeasures shares similarities with the problem of selecting 'best' alternative designs expose similar characteristics. State-of-the-art genetic algorithms might therefore improve our procedure for selecting most appropriate countermeasures. The key idea is to evolve a given population, here a set of countermeasure selections, towards a better one. Each iteration yields a new generation based on the previous one, the fitness of the individuals, and possible combinations of individuals. Genetic algorithms were used for selecting 'best' designs in [Hea11].

5.4 Integrating selected countermeasures in the goal model

The selected countermeasures should be integrated in the original goal model in order to progress towards a more complete goal model. For a given obstacle, integrating a countermeasure goal to an obstacle in a goal model means:

(a) Connecting this goal to a parent node in the model;

(b) Adding other sibling subgoals in the refinement, if necessary;

(c) Propagating the corresponding changes along refinement trees in which the countermeasure goal is involved;

(d) Refining this goal if necessary.

In addition to the progress property introduced in Section 5.1, the integration $Int_{CG,LO}(M)$ of a countermeasure CG to a leaf obstacle LO in goal model M should satisfy the *minimal change* and *correctness preservation* properties.

Definition 5.9. (Minimal change) An integration $Int_{CG,LO}$ guarantee *minimal changes* if it preserve prescribed behaviors in M that are not affected by LO. That is, for any goal G in M such that $\{LO, Dom\} \not\models \neg G$ and corresponding goal G' in M', we should have:

$$G' \models G.$$

Definition 5.10. (Correctness preservation) An integration $Int_{CG,LO}$ guarantee *correctness preservation* if the correctness of goal refinements in M is preserved in M'. That is, if all refinements in M are complete, consistent, and minimal then those in M' are complete, consistent, and minimal as well.

Section 5.4.1 shows how the minimal change property can be guaranteed by countermeasure integration schemas. Section 5.4.2 details how correctness is preserved through the integration. In these two sections the countermeasure goals are introduced in the original goal model by modification of the refinement structure. Section 5.4.3 proposes a notational mechanism for improving the documentation of obstacle handling seen as exception handling in goal models.

5.4.1 Integration schemas for ensuring minimal changes

Two alternative integration schemas may be used for ensuring that the *minimal change* property is met, that is, nothing more than necessary is modified. Which schema to apply depends on the obstacle resolution tactic being selected [Lam00; Lam09].

- The first schema removes the obstructed goal; it should be applied when the *Substitute goal* or *Weaken goal* tactic is used for resolving the obstacle.
- The second integration schema keeps the obstructed goal in the model; it should be applied when the *Avoid obstacle*, *Reduce obstacle likelihood*, or *Mitigate obstacle* tactic is used.

Removing the obstructed goal. Figure 5.2 shows a first integration schema expressed as a model rewriting rule. On the left, the countermeasure goal CG is not integrated whereas the latter is integrated on the right. In this first schema, the refinement of anchor goal AG, containing at least one obstructed goal, is replaced by a new refinement. The latter contains the countermeasure goal CG to leaf obstacle LO together with all non-obstructed children, that is G_3. (An anchor's obstructed children are those being directly or indirectly obstructed by LO.) The anchor AG may need to be de-idealized; in this case, AG' replaces AG in the modified goal model. The de-idealization might requires de-idealization of other connected goals such as G_1 and PG.

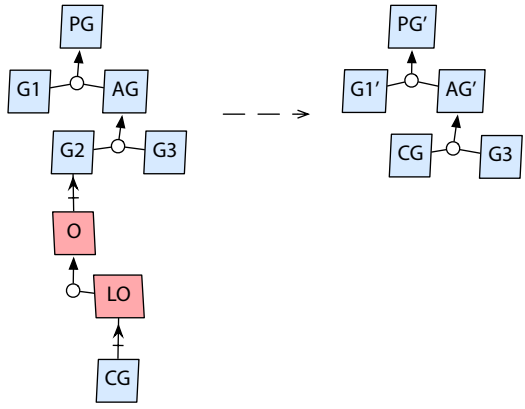


Figure 5.2 Integration schema with obstructed goal being removed.

This first integration schema has a precondition for use, namely, the countermeasure goal CG and the non-obstructed children must be sufficient for satisfying the anchor goal:

$$\{CG, Children(AG) \setminus ObstructedChildren(AG, LO)\} \models AG'$$

where $ObstructedChildren(AG, LO)$ denotes the children of AG obstructed by the leaf obstacle LO, and $Children(AG)$ denotes the children of the anchor goal AG. Otherwise, the new refinement would not be complete.

For example, the goal Achieve [Make Up Water Requested When Alarm Raised] assigned to the Controlling Software agent is a countermeasure produced through the *agent substitution* tactic [Lamoo; Lamo9]. Its anchor goal is Achieve [Make Up Water Requested When Loss Of Cooling]. The following refinement is complete, consistent, and minimal for this anchor goal:

Achieve [Make Up Water Requested When Loss Of Cooling]
→ Achieve [Make Up Water Requested When Alarm Raised]
→ Achieve [Alarm Raised When Low Water]

We may therefore replace the old refinement by this one.

It is easy to see that this first integration schema meets our minimal change property; G_3 remains untouched and anchor goal AG and its connected goals such as G_1 might be de-idealized through change propagation (see Section 5.4.2). All these goals thus satisfy $G' \models G$.

Keeping the obstructed goal. Figure 5.3 shows a second integration schema. In this schema, a new refinement is introduced; it includes a de-idealized version of the obstructed anchor, that is $\neg LO \rightarrow AG$, and the countermeasure goal. The obstructed anchor is de-idealized for removing the obstruction. The negation of the leaf obstacle is added to form a *by-case* AND-Refinement [Lamoo]. The de-idealization might requires de-idealization of other connected goals such as G_1 and PG.

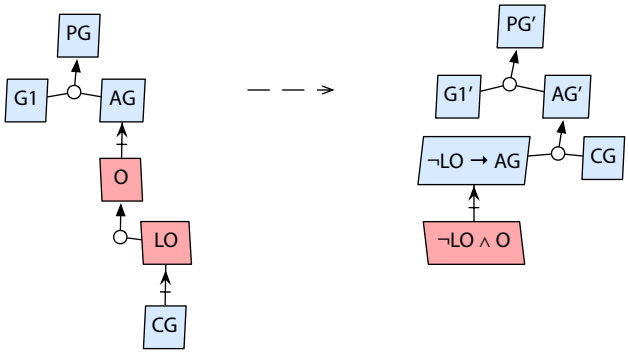


Figure 5.3 Integration schema with obstructed goal being kept.

This second integration schema has a precondition for use as well, namely, the countermeasure goal must be sufficient for satisfying the anchor goal when the obstacle occurs:

$$\{LO, CG, Dom\} \models AG'$$

For example, the countermeasure goal Achieve [Cooling System Repaired] entails its anchor Achieve [Make Up Water Provided When Loss Of Cooling] when the obstacle Valve Mechanical Failure occurs. This second integration schema then produces the following refinement:

Achieve [Make Up Water Provided When Loss Of Cooling]
→ Achieve [Make Up Water Provided When Loss Of Cooling
And No Valve Mechanical Failure]
→ Achieve [Cooling System Repaired]

Note that the anchor goal is not de-idealized in this example. The new goal Achieve [Make Up Water Provided When Loss Of Cooling And No Valve Mechanical Failure] is a de-idealization of the obstructed subgoal. The negation of the obstacle is added to the goal antecedent:

$$LossOfCooling \wedge \neg MakeUpValveFailure \Rightarrow \Diamond_{\leq 55h} MakeUpWaterProvided$$

This second integration schema can be seen to meet our minimal change property as well; the reasoning is similar to the first schema. Only siblings of the anchor goal and its ancestor may need to be de-idealized. The other goals were obstructed by the resolved obstacle and are therefore not concerned by the minimal change property.

5.4.2 Change propagation for preserving refinement correctness

Goal de-idealization and countermeasure integration may require corresponding changes to be propagated along refinement trees in which the countermeasure goal is involved. Such propagations are intended to ensure our third property on integrations, that is, the resulting model must remain *complete*, *consistent*, and *minimal* as defined in Section 3.2.2. This section discusses how de-idealizations and change propagation can be performed.

De-idealization by strengthening the goal's antecedent. A first way of de-idealizing a goal of form $C \Rightarrow \Theta T$ is to add an adequate conjunct to its antecedent:

$$AddConjunct(C \Rightarrow \Theta T, \Theta EC) = [C \wedge \Theta EC] \Rightarrow \Theta T$$

For example, the goal Achieve [Make Up Pump Motor On When Water Requested] may be de-idealized so as to exclude pump failure from pump actuation:

$$WaterRequested \wedge \neg PumpFailure \Rightarrow \Diamond_{\leq 5m} PumpMotorOn$$

De-idealization by weakening the goal's consequent. A second way of de-idealizing the goal $C \Rightarrow \Theta T$ is to add an adequate disjunct to its consequent:

$$AddDisjunct(C \Rightarrow \Theta T, \Theta EC) = C \Rightarrow [\Theta T \vee \Theta EC]$$

For example, the goal Achieve [Make Up Pump Motor On When Water Requested] might be de-idealized so as to require the pump or the redundant pump to be actuated:

$$WaterRequested \Rightarrow \Diamond_{\leq 5m} PumpMotorOn \vee \Diamond_{\leq 10m} RedundantPumpMotorOn$$

Change propagation. When a goal is de-idealized, the change must be propagated along the refinement trees in which this goal is involved—both up and down such trees. When applying the *AddConjunct* or *AddDisjunct* operators the corresponding syntaxic changes are applied recursively to the other goals up and down refinement links.

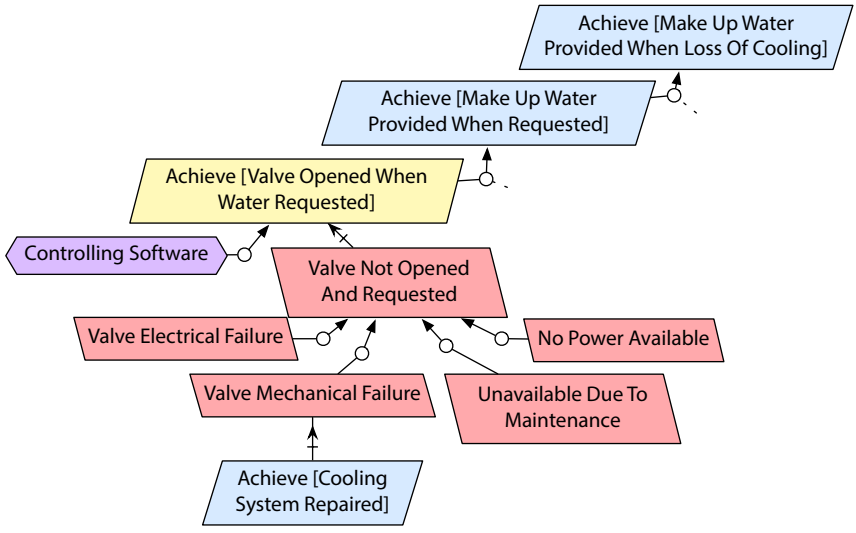


Figure 5.4 A countermeasure goal to the obstacle Valve Mechanical Failure.

For example, the integration of the countermeasure goal Achieve [Cooling System Repaired] for the obstacle Valve Mechanical Failure, as shown in Figure 5.4, requires change propagation to the descendants of the goal Achieve [Make Up Water Provided When Loss Of Cooling]. First, this goal is modified as previously shown:

$$LossOfCooling \wedge \neg MakeUpValveFailure \Rightarrow \Diamond_{\leq 55h} MakeUpWaterProvided$$

Next, the change is down-propagated, leading to an application of *AddConjunct* to the goal Achieve [Make Up Water Provided When Requested]:

$$MakeUpWaterRequested \wedge \neg MakeUpValveFailure \\ \Rightarrow \Diamond_{\leq 24h} MakeUpWaterProvided$$

The next refinement instantiates the *divide-and-conquer* refinement pattern [Lamoo]. The *AddConjunct* operator is applied to the goal Achieve [Valve Opened When Water Requested]. (This operator could have been applied to the goal Achieve [Make Up Pump Motor On When Water Requested] but the latter was assumed irrelevant.) We obtain:

$$MakeUpWaterRequested \wedge \neg MakeUpValveFailure \\ \Rightarrow \Diamond_{\leq 15m} MakeUpValveOpen$$

Since this goal is a leaf goal, the propagation ends.

Change propagation in the general case proceeds as follows, as in [Lam98b].

- When *AddConjunct* is applied to a goal for de-idealization, the specification pattern followed by the formal specification of the goal is identified and used to extract the antecedent.
- The de-idealized antecedent is obtained by adding the conjunction. The original antecedent is then replaced by the de-idealized one.
- The process is applied recursively to the subgoals (resp. parents) until leaf goals (resp. root goals) are reached.

The procedure is similar when applying *AddDisjunct*.

Such change propagation produces formal specifications that often need be simplified. Moreover, other mechanisms are required for change propagation to goals not matching a known specification pattern. Even though pattern-based propagation can be performed semi-automatically, interaction with the analyst is required when no specification pattern can be matched. The general problem of automatic change propagation through arbitrary refinement structures remains open.

5.4.3 Obstacle resolution as exception handling

The integration of countermeasure goals as such through new, explicit refinements in the original model raises several issues:

- The goal graph might undergo significant changes every time a new obstacle is identified.
- Normal situations are mixed with exceptional ones; it may prove hard to distinguish the former from the latter without domain expertise.
- Goal specifications become increasingly more complex.
- As new countermeasures are introduced, the ordered nesting of exceptional cases along refinements may lead to a combinatorial blow-up of special cases.

This section shows how dedicated constructs may help addressing these issues.

Extending the goal specification language. Dedicated constructs are proposed for encapsulating the required modifications while documenting each exceptional case separately.

Except. A first construct links a countermeasure goal to its anchor goal:

Goal AG
  FormalSpec $C \Rightarrow \Theta T$
  Except O **then** CG

where AG denotes the anchor goal $C \Rightarrow \Theta T$ of a countermeasure goal CG to an obstacle O. Semantically, this implicit specification is fully equivalent to the refinement in Figure 5.5. This construct may be used under the following precondition:

$$\{O, CG, Dom\} \models AG$$

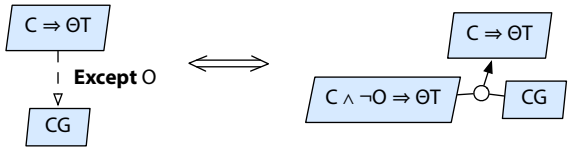


Figure 5.5 Semantic equivalent of *Except*.

For example, consider the goal Achieve [Make Up Water Provided When Loss Of Cooling]. Under the exceptional condition of a mechanical failure of the make-up valve, the goal is guaranteed through repairing the cooling system. We may therefore specify the following:

Goal Achieve [Make Up Water Provided When Loss Of Cooling]
Except Valve Mechanical Failure **then** Achieve [Cooling System Repaired]

This specification is logically equivalent to the refinement illustrating the second integration schema in Section 5.4.

Multiple *Except* annotations may be attached to a single goal to cope with different obstacles; the latter may therefore be introduced incrementally.

Compared with the complexity of an equivalent explicit specification, the complexity of an implicit goal specification with multiple *Except* annotations remains linear in the number of exceptions. The specification of the original goal remains unchanged. Moreover, multiple annotations sharing the same countermeasure goal may be factored out to simplify the model.

Provided. A second construct specifies an extra conjunct on the antecedent of an original goal G to produce a countermeasure:

Goal AG
FormalSpec $C \Rightarrow \Theta T$
Provided EC

where EC denotes an extra conjunct to be added to the goal antecedent for resolving the considered obstacle. Semantically, this implicit specification is equivalent to:

Goal AG
FormalSpec $C \wedge EC \Rightarrow \Theta T$

The *Provided* construct is typically used for de-idealizing goals. For example, the de-idealization of the goal Achieve [Make Up Pump Motor On When Water Requested] may be specified by highlighting the normal situation as follows:

Goal Achieve [Make Up Pump Motor On When Water Requested]
Provided $\neg PumpMechanicalFailure$

It often proves convenient to write *ProvidedNot* EC instead of *Provided* EC.

RelaxedTo. Symmetrically to *Provided*, this construct specifies an extra disjunct on the consequent of an original goal G to produce a countermeasure:

Goal AG
FormalSpec $C \Rightarrow \Theta T$
RelaxedTo ED

where ED denotes an extra disjunct to be added to the goal consequent for resolving the considered obstacle. Semantically, this goal is equivalent to:

Goal AG
FormalSpec $C \Rightarrow \Theta T \vee ED$

This construct is useful for de-idealizing goals as well. For example, another de-idealization of the same goal Achieve [Make Up Pump Motor On When Water Requested] might be specified by highlighting the normal situation as follows:

Goal Achieve [Make Up Pump Motor On When Water Requested]
RelaxedTo $RedundantPumpMotorOn$

Multiple *Provided* and *RelaxedTo* annotations may be attached to a single goal to introduce multiple countermeasures.

Replaces. This construct appears useful for tracing previous versions of a goal:

Goal AG
Replaces AG'

Such traceability helps readers understand the rationale behind the final goal.

Model refactoring for goal exceptions. In practice, the analyst should decide at some point whether a countermeasure goal refers to an exceptional situation or to a normal one to be considered in the original goal model. Such a decision might depend on various factors such as the satisfaction rate of the resolved obstacle, the criticality of the obstructed goal, domain-specific culture, stakeholders wishes, and so forth. To make the decision flexible and easily reversible, model refactoring operators are proposed for attaching or detaching the annotations introduced in the previous section to/from a goal model. Three operators are available for transforming an annotated model portion into a standard one.

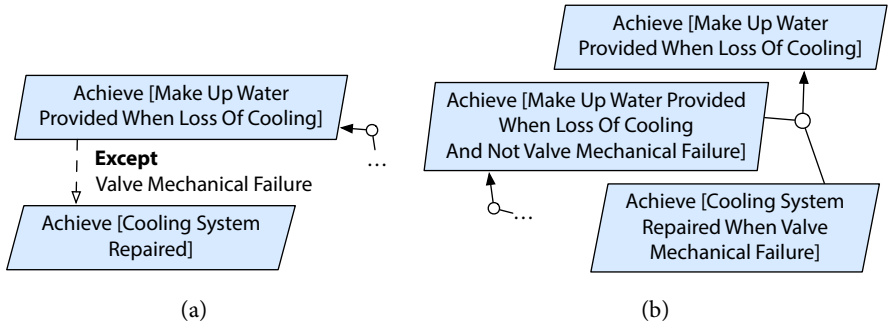


Figure 5.6 Implicit and explicit countermeasure integration.

Detach-Except, applied to an annotated goal, produces a new model where the goal is no longer annotated with a specific ***Except*** clause. The operator introduces a new refinement with two children: the countermeasure goal and a de-idealization of the original goal (see Figure 5.5 from left to right). The children of the original goal are then children of the de-idealized goal. Back to an earlier example, the operator takes the model fragment in Figure 5.6a to produce the model fragment in Figure 5.6b.

Detach-Provided, applied to an annotated goal, produces a new model where a specific *Provided* annotation is 'compiled' into its equivalent formal specification. For example, after application of this operator the goal Achieve [Make Up Pump Motor On When Water Requested] is specified without its *Provided* annotation as follows:

Goal Achieve [Make Up Pump Motor On When Water Requested]
FormalSpec $WaterRequested \wedge \neg PumpMechanicalFailure \Rightarrow \Diamond_{\leq 5m} PumpMotorOn$

Detach-RelaxedTo, applied to an annotated goal, produces a new model where a specific *RelaxedTo* annotation is removed. Back to an earlier example, the application of this operator to the goal Achieve [Make Up Pump Motor On When Water Requested] yields the following goal specification:

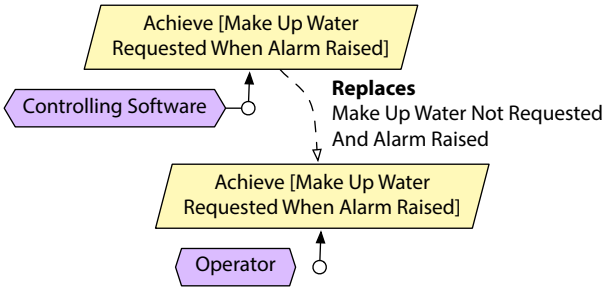
Goal Achieve [Make Up Pump Motor On When Water Requested]
FormalSpec $WaterRequested \Rightarrow [\Diamond_{\leq 5m} PumpMotorOn \vee \Diamond_{\leq 10m} RedundantPumpMotorOn]$

Similarily, three operators are available for transforming a standard model portion into an annotated one—namely, *Attach-Except*, *Attach-Provided* and *Attach-RelaxedTo*. These operators are the reverse of the *Detach* ones.

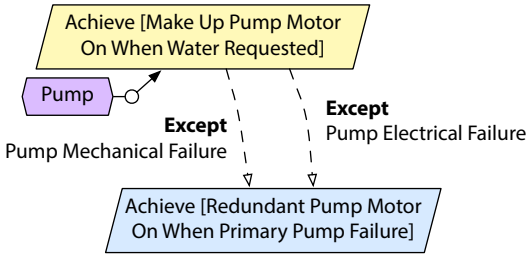
Exception diagrams. Textual goal specifications with *Except* and *Replaces* annotations may be graphically represented in an exception diagram. Figure 5.7 shows two portions of such a diagram for the goals Achieve [Make Up Water Requested When Alarm Raised] and Achieve [Make Up Pump Motor On When Water Requested]. This diagram captures that

(a) This goal replaces the similar goal assigned to the Operator agent ;

(b) When the obstacle Pump Mechanical Failure or Pump Electrical Failure occurs the countermeasure goal Achieve [Redundant Pump Motor On When Primary Pump Failure] will guarantee this goal.



(a) Exception diagram with a *replace* annotation.



(b) Exception diagram with two *except* annotations.

Figure 5.7 Exception diagrams.

5.5 Summary

This chapter presented techniques for controlling likely and critical obstacles. The latter are resolved through valid countermeasures in order to produce more complete requirements. Countermeasures are selected to maximize the satisfaction rate of high-level goals while minimizing the resolution cost. Our integration operator guarantees three properties, namely, *progress* towards a more complete goal model, *minimal changes*, and *correctness preservation*. A valid countermeasure goals guarantees *progress*; once the countermeasure is integrated, the satisfaction rate of some high-level goal(s) increases. Such increase quantifies the impact of a countermeasure and drives the selection of appropriate countermeasures. The chapter proposed two integration schemas, to be selected according the specific resolution tactic being used, to guarante *minimal*

change to the original model. Propagating the changes through guarantees the correctness of the new model. However, such integrations intermixes ideal and exceptional cases. To mitigate this, extensions to the goal specification language were proposed to specify exceptions and de-idealization to goals together with refactoring operators. Exceptions may then be visualized in exception diagrams.

The techniques presented so far always assume a single-value for goal satisfaction rates. The next chapter introduce uncertainty about the satisfaction rates and details how such uncertainty may impact the assessment of most critical obstacles.

Chapter 6

Handling Uncertainty in Satisfaction Rates

Requirements engineers faces uncertainties in a variety of ways regarding the system-to-be [Cheo9c; Esf13; Let14; Per14; Wel10]. In particular, the extent to which the requirements will be satisfied is uncertain.

This type of uncertainty is commonly called *physical uncertainty*; It refers to system phenomena [OHa06; Per14; Vos08]. For example, the chance of a sensor breaking down might be captured by a probability. This domain-level form of uncertainty can be reduced by changing the system under consideration—e.g., by introduction of appropriate countermeasures as seen in the previous chapter.

However, the extent to which those probabilities are accurate is uncertain as well. This uncertainty is commonly called *knowledge uncertainty*; it refers to the assessment of physical uncertainty by experts [OHa06; Per14; Vos08]. Our imperfect knowledge of what the exact probability values are may lead us to estimate them with some uncertainty margin. Such meta-level form of uncertainty can be reduced through further studies, consultation of more experts, increased experience, run-time monitoring of relevant events or data, and so forth.

Few risk-based requirement engineering frameworks so far support such knowledge uncertainty; the techniques for risk assessment and control are therefore less accurate and less applicable.

This chapter addresses this problem of knowledge uncertainty in probability values. It explains how knowledge uncertainty can be captured, how uncertainty margins for leaf obstacles impact uncertainty margins for high-level goals, and how such uncertainty margins can be reduced. The techniques in the previous chapters for identifying likely and critical risks and for selecting most appropriate countermeasures are extended to cope with uncertainties in probability estimates.

Our framework is extended to support knowledge uncertainty by means of probability distributions. Such distributions allow experts to specify their knowledge about satisfaction rates of leaf obstacles. A repeated application of the single-value propagation procedure in Section 4.2.1, using sampled satisfaction rates for the leaf obstacles, provides satisfaction rates for higher-level goals together with uncertainty margins. Two metrics, the violation uncertainty and the uncertainty spread, summarize the uncertainty about satisfaction rates. These metrics provide finer-grained assessment for critical obstacles. Selecting most appropriate countermeasures in presence of uncertainty guarantees that higher-level goals are satisfied up to some uncertainty threshold.

The chapter is organized as follows. Section 6.1 describes how knowledge uncertainty is captured through probability distributions. Section 6.2 shows how leaf obstacles are assessed when considering knowledge uncertainty, how the uncertainty propagates through the obstacle and goal models, and how obstacles with critical uncertainty margins are identified. Section 6.3 details how countermeasures are selected in presence of knowledge uncertainty.

6.1 Capturing knowledge uncertainty

Probability distributions are commonly used in risk analysis for representing the uncertainty about single-point values of random variables [Vos08].

A *probability distribution* is a function providing the probabilities of occurence of possible outcomes. The domain considered here is the set of possible probability values, that is, the interval $[0, 1]$. The probability distribution captures the probability for each possible single-point value in that domain. This section extends probabilistic goals and obstacles to capture the uncertainty about satisfaction rates.

Section 6.1.1 presents how uncertainty is captured in satisfaction rates. Section 6.1.2 provides two metrics for measuring critical uncertainties. Section 6.1.3 shows how probability distributions can be estimated by domain experts.

6.1.1 Uncertainty about satisfaction rates

Let A denote a probabilistic assertion specifying a goal, an assumption (prescriptive or descriptive), or an obstacle in the goal/obstacle models.

Definition 6.1. (Satisfaction uncertainty) The *satisfaction uncertainty* for assertion A, denoted by su_A, is defined as a probability density function over its single-point probabilities of satisfaction.

For example, Figure 6.1 shows the probability distribution capturing the satisfaction uncertainty su_G for the goal Achieve [Make Up Water Provided When Loss Of Cooling] appearing in Figure 4.3. As seen in Figure 6.1, the probability of satisfaction for this goal lies between 71.3% and 84.1%, with a more likely value around 80%. This multi-value estimate does not completely meet the goal's required satisfaction rate (RSR), prescribed to be 80% as depicted by the red line on the right. Section 6.2.2 below describes how such distribution can be computed.

Figure 6.1 Satisfaction uncertainty for the goal Achieve [Make Up Water Provided When Loss Of Cooling].

Uncertainty about goal satisfaction arises from satisfaction uncertainties about obstacles, domain hypotheses, and domain assumptions. For example, the certainty about the extent to which the goal Achieve [Make Up Water Provided When Loss Of Cooling] is satisfied depends on our certainty about the extent to which, among others, the obstacles Pump Electrical Failure or No Power Available are satisfied. Figure 6.2 illustrates the satisfaction uncertainty for these obstacles.

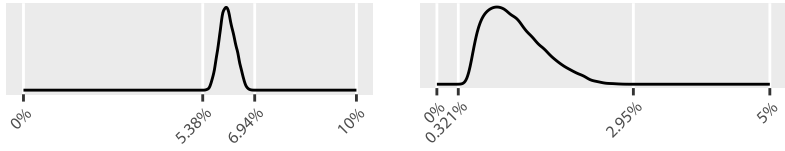


Figure 6.2 Satisfaction uncertainty for obstacles.

6.1.2 Uncertainty metrics for goal satisfaction

The probability of observing a satisfaction rate of at least p for goal G, denoted $SU_G(p)$, is obtained as follows [Wac07]:

$$SU_G(p) = \int_p^1 su_G(x)$$

This probability corresponds to the area delimited by the su_G curve and value p. In Figure 6.1, the chance of the goal Achieve [Make Up Water Provided When Loss Of Cooling] being satisfied in at least 80% of cases is accordingly given by:

$$SU_{\text{Achieve[MakeUpWaterProvidedWhenLossOfCooling]}}(0.8) = 0.7313$$

It corresponds to the red area in Figure 6.1. As shown in Section 6.2.2, su_G can be computed by up-propagation from leaf obstacles to their root in obstacle refinement trees, and then by up-propagation through the goal model. Given su_G, it is fairly easy to compute $SU_G(p)$.

Two metrics may be defined to capture *(a)* our degree of certainty that the goal's RSR will be met; and *(b)* the spread of satisfaction uncertainty below this RSR.

Goal violation uncertainty. The violation uncertainty for a goal is the probability of being below its required satisfaction rate. This metric corresponds to *goal failure risk* in [Let14].

Definition 6.2. (Violation uncertainty) The *violation uncertainty* for a probabilistic goal G, denoted by $VU(G)$, is the proportion of satisfaction uncertainty falling below the goal's RSR.

It is obtained as follows:

$$VU(G) = SU_G(RSR(G)) = \int_0^{RSR(G)} su_G(x)$$

For discrete values, the violation unceratinty is computed as the ratio between the number of values below RSR and the total number of values.

Graphically, this violation uncertainty corresponds to the surface below the satisfaction uncertainty curve, i.e. the probability density function, up to the goal's RSR. In our example, the violation uncertainty of Achieve [Make Up Water Provided When Loss Of Cooling] is .7313. This means that it is 73.13% certain that the goal will not meet its RSR of 80%. In Figure 6.1, most of the curve is on the left of the goal's RSR.

Uncertainty spread. Violation uncertainties only capture how much uncertainty lies below the required satisfaction rates. The same surface might however take many shapes with more or less spread. In practice, such spread may help making decisions. It might appear less problematic to have the uncertainty closer to the RSR than equally spread down to zero. If the uncertainty is close to the goal's RSR, a small change to this RSR might drastically change the violation uncertainty score.

The *uncertainty spread* for a goal measures the spread of uncertainty below the goal's RSR.

Definition 6.3. (Uncertainty spread) The *uncertainty spread* for a goal G, denoted by $US(G)$, is the semi-standard deviation of its probability distribution with respect to this RSR.

The semi-standard deviation of the distribution is used here for measuring spread. It differs from standard deviation as only values below a threshold are taken into account [Roh11]—here, those below the RSR. The uncertainty spread is obtained as follows.

$$US(G)^2 = \int_0^{RSR(G)} (x - RSR(G))^2 \cdot su_G(x)$$

For discrete values, this quantity can be computed as follows:

$$US(G) = \sqrt{\frac{1}{k}\sum_{x_i}(x_i - RSR(G))^2},$$

where x_i are those discrete values of G's satisfaction uncertainty which fall below $RSR(G)$, and where k denotes the number of such values.

Uncertainty spread values need be interpreted according to the shape of the curve. Whatever the shape, however, Chebyshev's inequality states that at least 75% of the data are at most at 2 spread values from $RSR(G)$ [Waco7]. If the goal's satisfaction uncertainty fits a specific probability distribution, tighter bounds can be obtained.

Back to our example, the goal Achieve [Make Up Water Provided When Loss Of Cooling] has a mean of 78.63% and an uncertainty spread of 0.0269. According to Chebyshev's inequality, this means that we have at least 75% of the satisfaction uncertainty between $(78.63-2\times2.69) = 73.25\%$ and $(78.63+2\times2.69) = 84.01\%$.

Other metrics, such as the expected satisfaction rate or metrics similar to the ones introduced in [Let14], could capture other facets of knowledge uncertainty. Which metrics appear more adequate remains an open question.

6.1.3 Eliciting distributions for leaf obstacle satisfaction rates

To facilitate the elicitation of distribution functions for leaf obstacles from domain experts, discrete points may be used that can be fitted to specific probability distributions—such as Beta, PERT, Triangular, etc. [Voso8].

A *quantile* is a single probability value attached to a cumulative probability. It indicates the cumulative probability to observe a single value of probability of satisfaction [Bedo1]. The .5 quantile, called *mode*, is the most likely probability of satisfaction.

To further support such elicitation from domain experts, databases providing estimates for quantiles can be used; they are available in a wide range of domains—e.g., aerospace, health, bank, nuclear, chemical, gas, water pollution, and so forth [Cooo8]. For example, our estimates for the running example and

the case-studies were based on reliability databases and published historical data in various industries [Ayy14; Cen92; Mil92]. Reliable techniques are also available for obtaining accurate single values or distribution estimates from trained experts [OHa06].

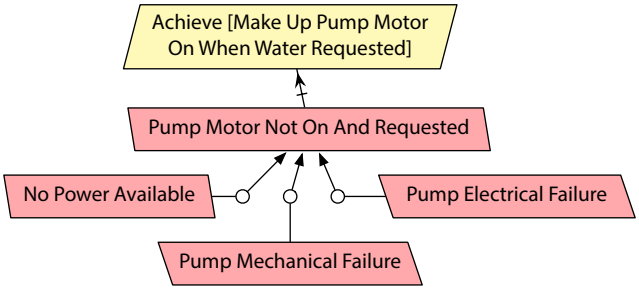


Figure 6.3 An obstacle tree fragment for Achieve [Make Up Pump Motor On When Water Requested].

Consider the obstacle Pump Mechanical Failure appearing in Figure 6.3, stating that the pump fails for more than 5 minutes:

$$\Diamond(WaterRequested \wedge \Box_{\leq 5m} PumpMechanicalFailure)$$

Experts would be asked to estimate the worst-case probability of observing a behavior where water is requested and the pump mechanically fails for more than 5 minutes. An expert might estimate that there are:

- at least 10% chances of observing at least 10 behaviors where the pump fails out of 100 equiprobable ones;
- at least 50% chances of observing at least 15 such behaviors;
- and at least 90% chances to observe at least 30 of them.

The .10, .5 and .9 quantiles are .1, .15, and .3 for the probability of satisfaction of this leaf obstacle, respectively. Figure 6.4 shows the satisfaction uncertainty for this obstacle.

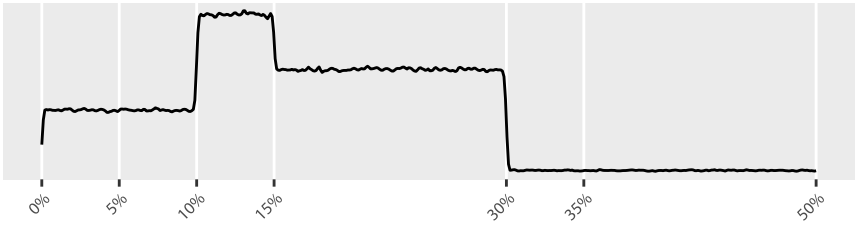


Figure 6.4 Satisfaction uncertainty for Pump Mechanical Failure.

A probability distribution might then be fitted to the estimates in order to interpolate probability values between the provided estimates. Which distribution fits best may depend on the domain. For example, we might fit uniform distributions between the estimated quantiles, as seen in Figure 6.4.

6.2 Obstacle assessment with knowledge uncertainty

As seen in Chapter 4, the identification of likely and critical obstacles is important for the next resolution step. This section shows how such identification is extended to support knowledge uncertainty about satisfaction rates. The satisfaction rate of some obstacles might be highly uncertain; these obstacles should have their uncertainty margins reduced. Our general approach for achieving this consists of the following steps:

- The accuracy of estimates for the satisfaction rate of leaf obstacles is increased by combining those elicited from multiple experts and/or other data sources (Section 6.2.1);
- Goal satisfaction rates and their uncertainty margin are computed from those more accurate estimates (Section 6.2.2);
- Leaf obstacles are prioritized according to their impact on the satisfaction of top-level goals using the metrics previously introduced (Section 6.2.3).

6.2.1 Eliciting more accurate estimates for leaf obstacles

The first step of our approach consists of eliciting estimated satisfaction rates for the leaf obstacles in the obstacle AND/OR refinement trees built during the obstacle identification phase. To reduce uncertainty margins for high-level goals, such estimates should be as adequate and accurate as possible.

As discussed in Chapter 4, the use of multiple sources or multiple experts is generally recognized to increase the accuracy of estimates and more mathematical approaches are recognized to produce accurate results [Cle99; Coo91]. Among available approaches, some are aimed at characterizing expert judgements by comparative assessment between their estimates and known quantities. The derived characteristics are then used for combining estimates from multiple experts.

In our proposed probabilistic framework, leaf obstacles may be annotated with estimated quantiles from multiple experts, e.g.,

Obstacle Pump Mechanical Failure
Definition The pump is not pumping water due to a mechanical failure.
Probability [Expert1] quantiles (10%,15%,30%)
Probability [Expert2] quantiles (20%,25%,35%)

However, experts are not all equal. Some might systematically over- or under-estimate probabilities of satisfaction; provide very large or very narrow estimates; provide estimate ranges that are accurate or not; and so forth. To get more accurate estimates, we may compare those provided by multiple experts with known values in order to characterize each expert. This is known as *calibration* [Bed01].

Definition 6.4. (Calibration variable) A *calibration variable* is a quantity whose exact single-point value is known.

In our context, a calibration variable should be closely related to a leaf obstacle—typically, a leaf obstacle whose satisfaction rate is known, an agent/resource failure whose failure rate is known from reliability databases [Akh01], and so forth.

In our running example, three calibration variables might be identified: Push Button Broken, Temperature Sensor Broken and Fire Sprinkler Broken. Table 6.1 summarizes estimates and known values for two of them. The X% columns show the quantiles estimated by the corresponding domain expert for the calibration variable. Our estimates were based on available reliability databases [Ayy14; Cen92; Mil92].

Variable	Expert	10%	50%	90%	Known Value
Push Button Broken	Expert 1	3.81%	4.28%	4.76%	4.37%
	Expert 2	3.96%	4.45%	4.94%	
Temperature Sensor Broken	Expert 1	2.31%	2.59%	2.88%	4.37%
	Expert 2	4.18%	4.70%	5.22%	
Fire Sprinkler Broken	Expert 1	0.01%	0.02%	0.03%	0.02%
	Expert 2	0.02%	0.022%	0.024%	

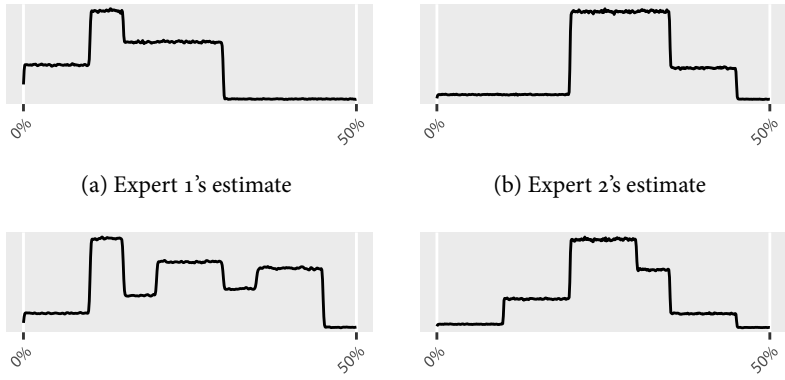
Table 6.1 Expert estimates for calibration variables.

Two techniques are available for characterizing multiple experts and combining their estimated quantiles. We instantiate them to our context in order to combine multiple quantiles on a leaf obstacle into a single satisfaction uncertainty.

- *Cooke's technique* [Coo91] characterizes each expert through a single weight. This weight combines a calibration score and an information score. The *calibration score* measures how close the expert's estimate is to the known value of the calibration variable; the closer the probability estimated for this value, the higher the score. The *information score* measures how precise the estimates are; the narrower the estimates, the higher the score. The satisfaction uncertainty combining the quantiles from multiple experts is obtained as a weighted sum of these quantiles. In our example, *Expert 1* gets an information score of 2.45, a calibration score of 0.76, and a weight of 0.59; *Expert 2* gets an information score of 2.75, a calibration score of 1, and a weight of 0.40. Figures 6.5(a) and 6.5(b) show the quantile function for both experts; Figure 6.5(c) shows the resulting satisfaction uncertainty.

- *Mendel-Sheridan's technique* combines the quantiles of the experts by use of a Bayesian calibrator/estimator [Men89]. It first computes a minimally informative distribution for the expert's characteristics. This a priori distribution considers each expert to be unbiased; it does not weight any probability more than others. This a priori distribution is then updated with respect to the expert's quantiles and the known value of calibration variables. The resulting distribution is used to combine the quantiles of the experts into a satisfaction uncertainty for our leaf obstacle. Figure 6.5(d) shows the resulting satisfaction uncertainty using this technique.

Which technique performs best remains an open question; it may depend on the application domain, the experts, and the number of calibration variables [Coo91].



(a) Expert 1's estimate

(b) Expert 2's estimate

(c) Combined estimates using Cooke's technique

(d) Combined estimates using Mendel-Sheridan's technique

Figure 6.5 Satisfaction of Pump Mechanical Failure.

6.2.2 Computing goal satisfaction rates and their uncertainty

Chapter 4 described a procedure for computing single-point probability values for goal satisfaction from single-point probability values for satisfaction of obstructing leaf obstacles. To build a full probability distribution for a top goal in the goal model, we need to sample the satisfaction uncertainty for those leaf obstacles. This procedure is known as Monte Carlo simulation.

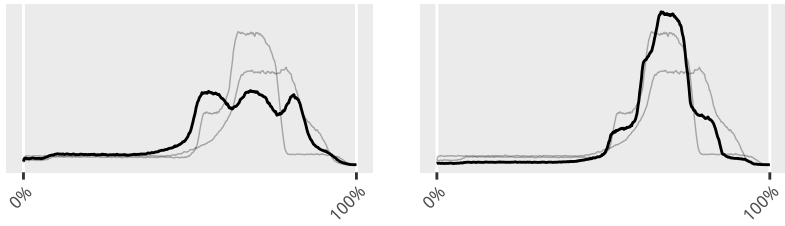
To sample a satisfaction uncertainty, a random number between 0 and 1 is uniformly picked. The inverse of the cumulative distribution function associated with the satisfaction uncertainty is then used to get a corresponding probability of satisfaction [Waco7]. More likely probabilities of satisfaction are thereby picked more often than less likely ones.

The probability of satisfaction of a top goal is then computed for a given sample. The sampling is repeated a large number of times to obtain a set of probabilities of satisfaction for this top goal. The obtained set of probabilities can then be aggregated into a distribution by using their frequency; this is known as *Kernel Density Estimation* [Frio1]. For example, by repetitive sampling for the leaf obstacles, the following probabilities of satisfaction for the top goal Achieve [Make Up Water Provided When Loss Of Cooling], in Figure 4.3, might be obtained:

$$0.762, 0.690, 0.686, 0.675, 0.736, 0.908, 0.794, 0.555, 0.678...$$

Using Kernel Density Estimation techniques, we can build the distribution shown in Figure 6.6. Figure 6.6(a) shows, in black, the satisfaction rate when expert estimates are combined using Cook's technique. Figure 6.6(b) shows, in black, the satisfaction rate using the Mendel-Sheridan's technique. The two curves in gray corresponds to the satisfaction rates obtained when only one of the two expert is used.

The number of samplings required depends on various factors such as the satisfaction uncertainties for the leaf obstacles, the numerical precision to achieve, the time available to solve the problem, and so forth. Note that the metrics introduced in the previous section can be computed from the sampled satisfaction rates, without infering the satisfaction uncertainty curve.



(a) Combined estimates using Cooke's technique

(b) Combined estimates using Mendel-Sheridan's technique

Figure 6.6 Satisfaction of Achieve [Make Up Pump Motor On When Water Requested].

6.2.3 Finding critical obstacles

The third step in our approach for obstacle assessment in presence of knowledge uncertainty about satisfaction rates consists of prioritizing obstacles according to the metrics introduced in Section 6.1.2, that is, the violation uncertainty and uncertainty spread for top-level goals.

Definition 6.5. (Critical obstacle in presence of uncertainty) An obstacle O is *critical* for a goal G if it results in a positive violation uncertainty; that is $VU(G) > 0$.

As in Section 4.3, we may proceed iteratively on the size of the combinations.

- The impact of each single leaf obstacle on the goal model is first assessed individually. For a given top goal, the procedure in the previous section is applied for each leaf obstacle with the satisfaction probability of all other leaf goals being set to 0. The violation uncertainty and the uncertainty spread for the considered top goal are then computed according to their definition in Section 6.1.2.

The result may be represented on a scatter plot, called *violation diagram with uncertainty*, to highlight the most critical obstacles to a considered high-level goal. The x-axis reports the uncertainty spread and the y-axis reports the violation uncertainty of the obstacle combinations for the considered goal.

- When all likely and critical individual obstacles are resolved, the violation uncertainty and uncertainty spread may be computed again with pairs of leaf obstacles in order to build a new violation diagram;
- Then with triples, and so forth.

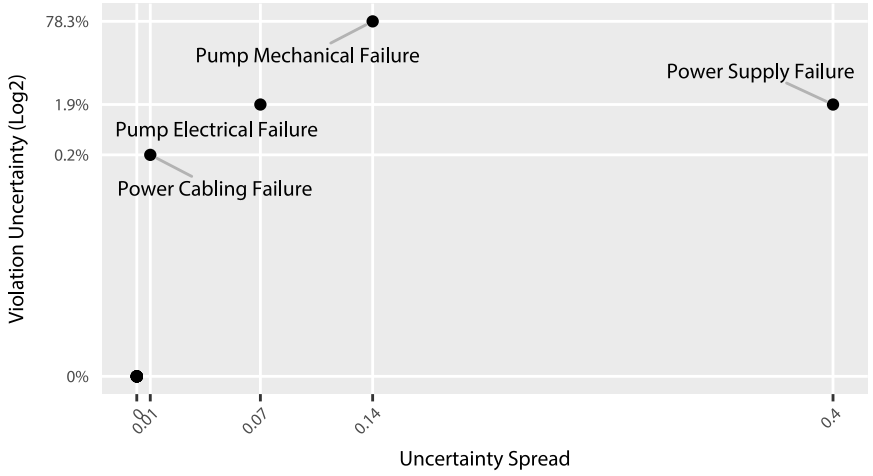


Figure 6.7 Violation diagram with uncertainty for the goal Achieve [Make Up Water Provided When Loss Of Cooling].

Our running example contains 12 leaf obstacles. The violation uncertainty and uncertainty spread for the top goal Achieve [Make Up Water Provided When Loss Of Cooling] resulting from the 12 obstacles are 90.70% and 0.1818, respectively. Figure 6.7 shows the violation diagram with uncertainty for leaf obstacles taken alone. Among the 12 leaf obstacles, there are 4 critical obstacles:

- The obstacle Pump Mechanical Failure is clearly a critical one. The violation of the top goal caused by the obstacle is mostly certain (with 78.3%).
- The top goal violation caused by Pump Electrical Failure is uncertain; however the uncertainty spread is very low. This indicates that the uncertainty is probably close to the goal's RSR. This obstacle might thus not be that critical as a slight change in the goal's RSR might drastically change the violation uncertainty.
- The top goal violation caused by Power Supply Failure is also uncertain. Its uncertainty spread appears high; the uncertainty is probably not close to the goal's RSR.
- The leaf obstacle Power Cabling Failure is not causing the highest violation uncertainty or a high uncertainty spread.
- It appears certain that the remaining 8 obstacles do not individually prevent the root goal from reaching its RSR.

As discussed in Section 4.3, our experience indicates that most of the critical pairs of obstacles contains an obstacle critical alone. This suggests the need for iterating on the size of obstacle combinations. Figure 6.8 shows the violation diagram with uncertainty for the goal Achieve [Make Up Water Provided When Loss Of Cooling] considering pair of leaf obstacles.

6.3 Selecting countermeasures with knowledge uncertainty

The previous section showed how likely and critical obstacles can be highlighted in the presence of knowledge uncertainty about satisfaction rates. When countermeasures to these are identified, most appropriate ones should be selected. Selecting most appropriate countermeasures in presence of knowledge uncertainty requires a relaxation of safe selections as defined in Section 5.3.2. In the presence of uncertainty, a safe selection ensures that the violation uncertainty is bounded.

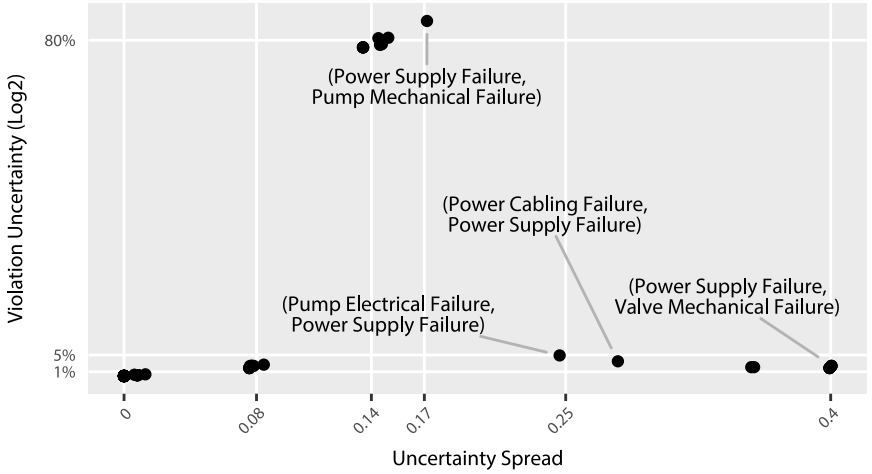


Figure 6.8 Violation diagram with uncertainty for the goal Achieve [Make Up Water Provided When Loss Of Cooling].

Definition 6.6. (Safe Selection with Uncertainty) Given a violation uncertainty threshold T, a *safe selection* of countermeasures is a set of countermeasures that, once integrated, guarantees that the violation uncertainty of the high-level goals is lower than T.

Not all safe selections are equal. The most appropriate ones need to

- Minimize the high-level goal's violation uncertainty—that is, they need to maximize the fraction of uncertainty above the goal's required satisfaction rate.
- Minimize the uncertainty spread—the closer the uncertainty to the goal's RSR, the better.

According this relaxed definition, selecting the most appropriate countermeasures amount to solving the following optimization problem:

(1) Find the minimal cost for guaranteeing the violation uncertainty threshold for the high-level goals,

(2) Find the selections that minimize the violation uncertainty and the uncertainty spread for the high-level goals given this cost.

Back to our running example, consider the high-level goal Achieve [Make Up Water Provided When Loss Of Cooling]. The minimal cost for guaranteing a violation uncertainty below 5% ranges from 1 to 6. There are 63 possibles selections; among these, 48 are safe. The minimal cost is 1; we used unitary costs for the countermeasures. The countermeasure selection that minimizes the violation uncertainty only contains the countermeasure goal Achieve [Cooling System Repaired]. The resulting violation uncertainty and uncertainty spread is 2.9% and 0.3773 respectively. This should be contrasted with their counterparts 89.8% and $.1719$, respectively, without the countermeasure goal.

6.4 Summary

The quantitative technique presented in this chapter allows analysts to cope with knowledge uncertainty about satisfaction rates of system goals and their obstructing obstacles. The probabilistic framework presented in the previous chapters was extended to reason explicitly about uncertain estimates. The impact of estimation uncertainty is measured on high-level goals through two metrics: goal violation uncertainty and uncertainty spread. The satisfaction rates and their uncertainties for high-level goals are computed by up-propagation through obstacle and goal refinement trees, from leaf obstacles whose satisfaction rates and uncertainty need be estimated. As a result, more critical leaf obstacles can be highlighted for resolution and most appropriate countermeasures can be selected. To reduce uncertainty margins, our approach allows estimates from multiple sources to be combined.

Beyond frameworks for goal-oriented RE, other frameworks for RE and risk analysis might benefit from our techniques. In particular, Fault Tree Analysis (FTA) might integrate similar metrics for problematic uncertainties together with means for reasoning about risk consequences and combinations of multiple expert estimates. Other software engineering areas are faced with the problem of handling uncertainties about estimated quantities [Fenoo]. The application of similar techniques appears worth considering in other contexts beyond RE as well.

The next chapter shows how the assessment and control of critical obstacles can be performed at system runtime to support obstacle-driven runtime model adaptation.

Chapter 7

Handling Obstacles At System Runtime

Software systems are deployed in changing environments. The identification, assessment, and control of obstacles through countermeasures is based on environment assumptions and obstacle satisfaction rates that are determined at RE time. These influencing factors may however turn to be different at system runtime [Fic95]. Some assumptions might no longer hold; new characteristics might emerge; experts' estimates at RE time for obstacle assessment might prove inaccurate at system runtime; other estimates might not be available at RE time; and so forth. For better fit to the system's goals under changing or originally unknown conditions, it might thus be better to defer decisions on selecting most appropriate countermeasures to system runtime [Fea98].

This chapter discusses how obstacles and goals may drive the runtime adaptation of a software system. In particular, it introduces how probabilistic obstacles can be monitored at runtime and how the monitored satisfaction rates can drive the dynamic selection of more appropriate countermeasures in view of changing conditions.

The satisfaction rate of an obstacle is defined as the lowest state probability of satisfying its specification. Such precise characterization in terms of states and behaviors enables the monitoring of state probabilities. The satisfaction of a specification is monitored at runtime using LTL-monitoring techniques; its satisfaction rate is estimated by counting positive and negative occurrences of the formula. The monitored satisfaction rate of leaf obstacles is then up-propagated to high-level goals in the goal model (as seen in Chapter 4). If the satisfaction rate of a high-level goal does not exceed its required satisfaction rate, an adaptation of the software system is required. The currently deployed countermeasures are reconsidered; more appropriate ones are selected (as in Chapter 5) among

those identified at RE time, in view of runtime information about changing conditions; and the newly selected countermeasures are deployed in the running system. The software adaptation process thus can be driven by the satisfaction of probabilistic goals. This provides traceable criteria for deciding when an adaptation is required and what adaptations should be performed to guarantee the satisfaction of the considered probabilistic goals. We focus here on handling the uncertainty about which best countermeasures to deploy; the countermeasures candidate for deployment are assumed to be known.

The chapter is structured as follows. Section 7.1 presents the overall approach for dynamically switching to most appropriate countermeasure at runtime. Section 7.2 introduces a specific running example of runtime adaptation. Section 7.3 details how probabilistic obstacles can be monitored at runtime. Section 7.4 details how probabilistic obstacles may drive the runtime adaptation process. Section 7.5 explains how the running software can be adapted from the derived goal model adaptations. Section 7.6 discusses how knowledge uncertainty may affect obstacle monitoring and model adaptation.

7.1 Overview of the approach

The objective in this chapter is to let the system dynamically switch to more appropriate countermeasures to leaf obstacles in view of evolving environment conditions and obstacle satisfaction rates. The satisfaction rate of leaf obstacles was estimated at RE time; it can now be observed at system runtime. A set of alternative countermeasure goals to those leaf obstacles is assumed to be known; the countermeasures are identified and specified at RE time.

A countermeasure to an obstacle should dynamically replace the current one when, unlike the latter, it makes the satisfaction rate of high-level goals exceed their required satisfaction rate. The satisfaction rate of high-level goals is obtained from the monitored satisfaction rate of leaf obstacles by up-propagation through the goal/obstacle model, using the technique presented in Section 4.2.1. The monitored satisfaction rate of a leaf obstacle is obtained by counting the observed behaviors.

More specifically, our approach comprises 6 steps detailed in the next sections. The relation among steps is shown in Figure 7.1.

(1) **Build monitors for leaf obstacles.** The probabilistic leaf obstacles in obstacle refinement trees need be monitored at system runtime. Our approach extends the monitoring technique introduced in [Bau11] for non-probabilistic LTL assertions, in order to monitor probabilistic LTL assertions at system runtime. This step builds, at RE time, LTL_3 monitors for the leaf obstacles. We limit ourselves here to propositional logic. The list of predicates to observe at runtime is thereby provided. (See Section 7.3.1)

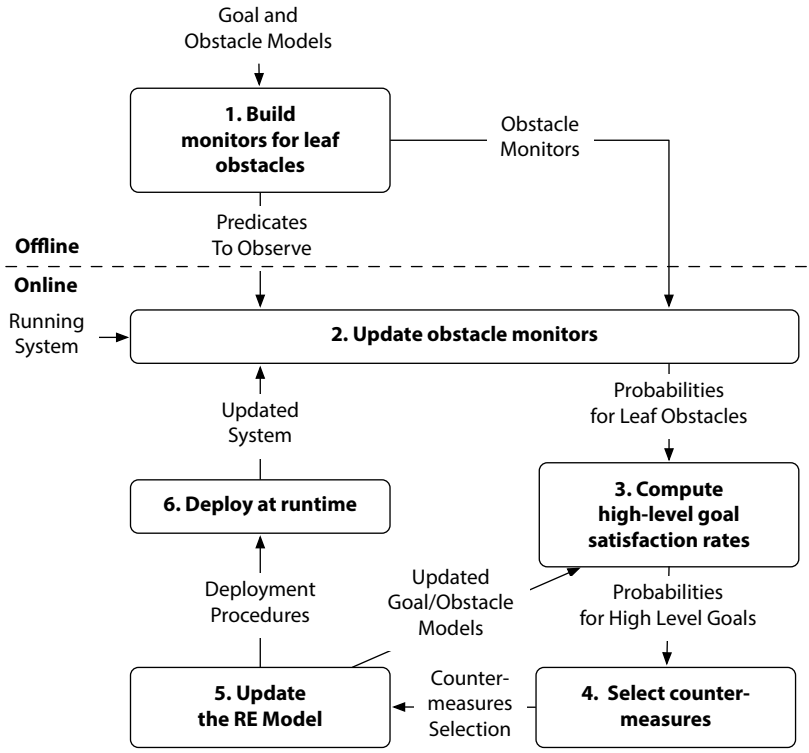


Figure 7.1 Overview of the approach.

(2) **Update obstacle monitors.** At runtime, the states of the monitored system are observed at a regular pace. (This pace may be chosen to fit a specific domain.) At every observation, a new virtual monitor for each leaf obstacle is started while existing monitors are updated. (See Section 7.3.2.)

(3) **Compute high-level goal satisfaction rates.** The monitored satisfaction rate of leaf obstacles is up-propagated through obstacle/goal refinement trees up to high-level goals. (As explained in Chapter 4.2.1, however performed here at runtime.)

(4) **Select countermeasures.** The comparison between the monitored satisfaction rates obtained for those goals and their required satisfaction rate (RSR) determines whether the current countermeasures to the monitored obstacles are still appropriate. If a monitored goal satisfaction rate falls below the goal's RSR, alternative more appropriate countermeasures are selected among those available. (As explained in Section 5.3)

(5) **Update the RE model.** The goal/obstacle model is updated accordingly by integrating the new current countermeasures and updating the propagation in Step 4. (As explained in Section 5.4)

(6) **Deploy at runtime.** The software is automatically adapted according to the selected countermeasures in the new model. (See Section 7.5)

7.2 Running example for runtime adaptation

The running example for this section was inspired from a flood detection system [Hugo6a; Hugo6b]. This running example is commonly used to illustrates or evaluates other runtime adaptation techniques [Beno8; Golo8; Saw10; Wel11].

> *'Hydrologists have traditionally approached the problem of modeling and predicting floods by deploying sensing and logging equipment in areas susceptible to flooding. River profile data from these sensors, such as depth and rate of flow, is then either transmitted off-site or collected manually. This data is then used as the input to complex spatial flood prediction models, which*

typically run on high-performance computers and Grid technologies. This data is also used as the basis for much simpler single-point models that can run on simpler devices.'—[Hug06a]

The example system here monitors the speed and depth of the river. If the levels are criticals (approximated here as '*above a given threshold*'), the locals are warned as soon as possible. The speed can be measured either using an ultrasound sensor or a camera-based sensor. Figure 7.2 shows a fragment of the goal model; Figure 7.3 shows a fragment of a corresponding obstacle model.

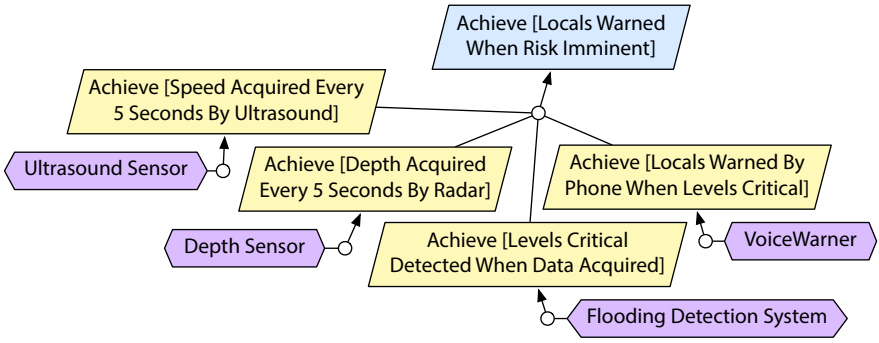


Figure 7.2 Goal model fragment for a Flood Detection System.

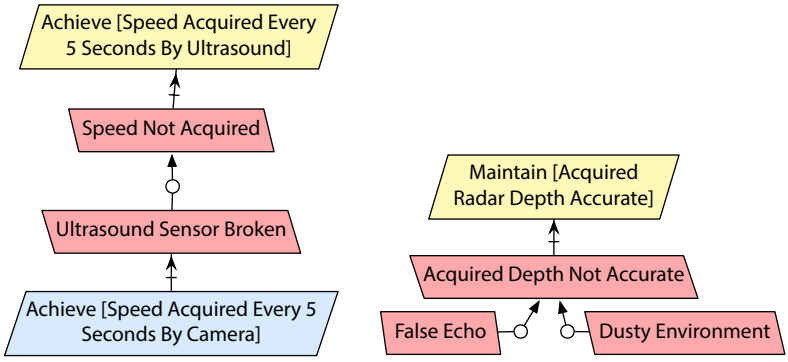


Figure 7.3 Obstacle model fragment for a Flood Detection System.

7.3 Monitoring probabilistic obstacles

At RE time, domain experts estimate the satisfaction rates of the leaf obstacles in obstacle refinement trees based on their knowledge of the system or their experience with similar systems as discussed in Section 4.1. These estimates might prove inaccurate at system runtime—they might be too rough or environment properties or assumptions might have changed in the meantime. Moreover, some variables might be hard to estimate at RE time—prior data might not be available or might be too costly to acquire; too many parameters might be involved; etc. Monitoring the actual satisfaction rate of leaf obstacles at system runtime helps filling this gap. Former estimates may be made more accurate; missing data may be made available.

7.3.1 Background: Monitoring non-probabilistic LTL assertions

This section introduces the necessary background for monitoring propositional non-probabilistic LTL assertions.

Monitoring the runtime satisfaction of an LTL formula relies on the finite trace observed so far, and on an automata-based expression of formula satisfaction. The approach presented in [Bau11] is chosen as it reports LTL formula satisfaction or violation as early as possible.

LTL_3, the LTL considered in [Bau11], uses *true*, *false*, and *inconclusive* as truth values. A finite trace is labeled as *true* if any continuation of it satisfies the formula; *false* if any continuation falsifies the formula; and *inconclusive* otherwise. A LTL_3 formula monitor is a finite state machine (FSM) that reads finite traces and outputs the corresponding truth value. As suggested by Figure 7.4, this FSM is built from the formula and its negation [Bau11].

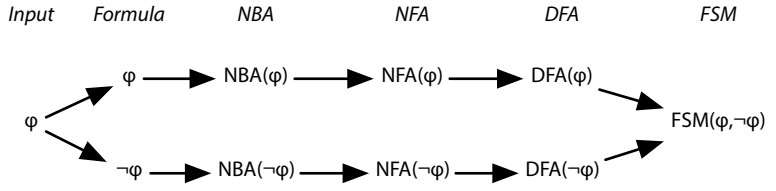


Figure 7.4 Steps for computing monitor FSM from input formula ϕ.

- The associated non-deterministic Büchi automaton (NBA) is generated using a standard algorithm [Baio8].

- A non-deterministic finite automaton (NFA) is then generated by performing an emptiness check for each NBA state. A state s is labeled with *true* if a continuation satisfying the formula exists (that is, if the language corresponding to the NFA with initial state s is not empty); it is *false* otherwise.

- The NFA is determinized to produce a deterministic finite automaton (DFA) using powersets construction [Baio8].

The monitor FSM results from the product of both DFAs. Let (s_1, s_2) denote a state in this product, where s_1 is the DFA state corresponding to the formula ϕ, and s_2 the DFA state corresponding to its negation $\neg\phi$. This monitor FSM state is labeled as:

- *true* if s_1 is labeled as *true* and s_2 as *false* (no continuation exists such that the formula is falsified);

- *false* if s_1 is labeled as *false* and s_2 as *true* (no continuation exists such that the formula is satisfied);

- *inconclusive* otherwise (a continuation exists such that the formula is satisfied, and a continuation exists such that the formula is not satisfied.)

At runtime, the current state of the monitor FSM is updated according to the truth value of the observed predicates and state assertions on the FSM transitions. More details about the approach can be found in [Bau11].

7.3.2 Monitoring-based estimation of satisfaction rates

As seen in Section 3.3.1, the satisfaction rate of an obstacle of form $\Diamond(C \wedge \Theta OC)$ is the upper bound among the state probabilities of $C \wedge \Theta OC$. We may at runtime count the number of observed behaviors satisfying and not satisfying the formula $C \wedge \Theta OC$; for a state s, the ratio between the number of observed behaviors rooted in s satisfying the formula and the number of observed behaviors

rooted in s may be used as estimates for the corresponding state probability. The automata-based monitoring procedure for LTL_3 determines at runtime whether $C \wedge \Theta OC$ is satisfied for a behavior rooted in s.

Note that we are monitoring probabilistic assertions here, leveraging a procedure for monitoring non-probabilistic assertions. We are not limited by specific patterns such as *Achieve* or *Maintain*; any valid combination of LTL operators is supported.

Definition 7.1. (Monitored Satisfaction Rate) The monitored satisfaction rate of an obstacle or a goal is its actual satisfaction rate as observed in the running system.

The process for obtaining the monitored satisfaction rate can be summarized as follows:

(a) Observed states are collected;

(b) (Optional step) Those observed states are filtered;

(c) Monitors are created and updated according to the observed states;

(d) Monitors labeled with T (true) and F (false) are counted; The ratio between the number of T- and F-monitors provides the monitored satisfaction rate.

***(a)* Observing states.** Consider the obstacle Dusty Environment shown in Figure 7.3. Figure 7.5 shows the process of monitoring the satisfaction rate of that obstacle, formalized as $\Diamond(\Box_{\leq 2s} dustyEnvironment)$, during 8 observations. The squares represent states of the observed system. Three states A, B, C are observed; A and B satisfy the predicate $dustyEnvironment$ while C does not. The circles show the label of the current state of the LTL_3 monitors.

As the semantics of our language is synchronous [Leto8], observations are made at a regular pace. If observations are performed every second, the Metric Linear Temporal Logic (M-LTL) formula $C \wedge \Theta OC$ inside the $\Diamond$-operator can be transformed into an LTL conjunction. Back to our example, if observations are performed every second, we observe:

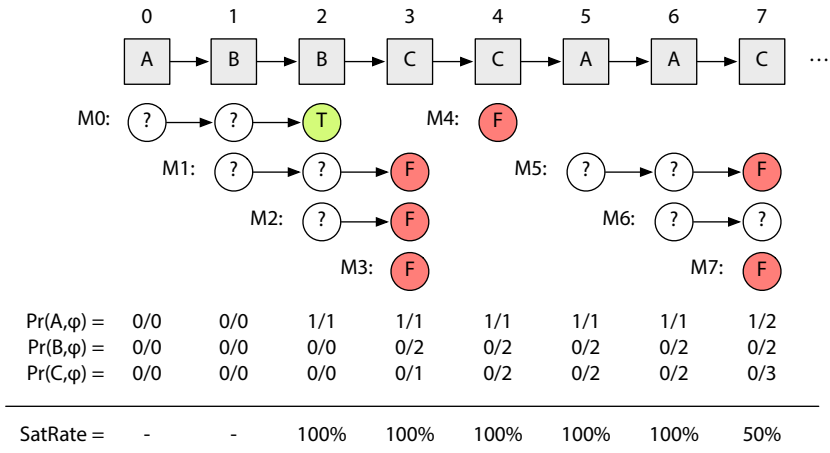


Figure 7.5 Monitoring the probabilistic obstacle Dusty Environment.

$$\phi : dustyEnvironment \wedge \bigcirc dustyEnvironment \wedge \bigcirc \bigcirc dustyEnvironment$$

To observe states, the monitoring infrastructure may monitor the propositional atoms appearing in the obstacle specifications. The global state of the system, as defined in Section 2.1.1, may thereby serve as a unique identifier for the observed state. Depending on the mentioned propositional atoms, the monitored states might thereby be different, see Section 9.1.

***(b)* Filtering observed states.** As time goes by, the number of monitors will increase and the monitored state probability will then vary less and less. This problem is acknowledged in other approaches such as [Epi09]. Indeed, at the begining, 10 new positive monitors might strongly impact the state probability, e.g.,

$$\text{from } {}^{5}\!/_{10} = .5 \text{ to } {}^{15}\!/_{20} = .75$$

whereas when 1000 observations have been made, the state probability would only change

$$\text{from } {}^{5}\!/_{1000} = .005 \text{ to } {}^{15}\!/_{1010} = 0.014.$$

This is not a problem if the 'actual' satisfaction rate does not vary a lot. On the contrary, if the 'actual' satisfaction rate is changing over time, this will smooth out variations.

To circumvent that effet, the process for obtaining the monitored satisfaction rate may be modified to forget old information [Cal11]. This filtering step keeps only recent monitors, for example the last 100 monitors.

The appropriate number of monitors to be kept needs to be determined adequatly depending on the domain, how fast the states are observed, and the desired innaccuracy margins (see Section 7.3.3). Such appropriate number might also be automatically set to an optimal aging factor [Cal14].

***(c)* Creating and updating monitors.** For a monitored leaf obstacle, at each observation of the running system, an LTL_3 monitor is started to check whether the behavior from the current state satisfies ϕ. As seen below, virtual copies are used in practice to avoid creating new monitors at runtime.

Figure 7.6 shows the corresponding LTL_3 monitor. The top left monitor state labeled with ? is the initial state. Each transition is labeled with a state formula. The label on states are: T for *true*, F for *false*, ? for *inconclusive*.

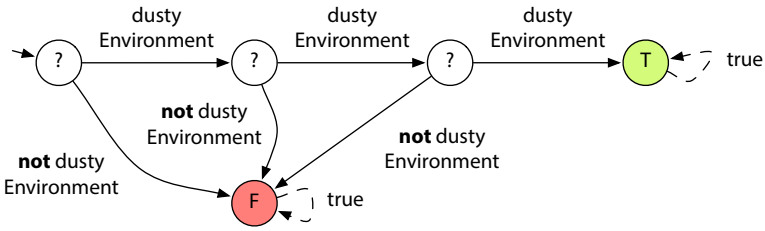


Figure 7.6 LTL_3 monitor for Dusty Environment.

Let us have a closer look at the example in Figure 7.5.

- At observation 0, we start a monitor M_0 to check whether the future system behavior satisfies ϕ.
- At observation 1, the monitor M_0 is still inconclusive; a new monitor M_1 is started.

- At observation 2, the current state of the monitor M_0 is updated to T (true). For state A, one observed behavior so far satisfies ϕ (see '1/1' at the bottom of Figure 7.5). A new monitor M_2 is started.

- At observation 3, we get the label F (false) for the current state of the monitors started at observations 1 to 3. For state B, two behaviors were observed to not satisfy ϕ (see '0/2' at the bottom of Figure 7.5). For state C, one behavior is seen to not satisfy ϕ ('0/1').

- At observation 7, one observed behavior from state A satisfies ϕ among the two observed ones (the third one is still inconclusive). The two observed behaviors from B violate the formula. The three observed behaviors from C also violate the formula.

In practice, creating a new LTL_3 monitor at each observation is clearly unrealistic as the time complexity is $\mathcal{O}(2^{2^n})$ where n is the size of the formula [Bau11]. To avoid creating multiple instances of the same monitor, one LTL_3 monitor is built at RE time; the monitors being started at runtime are virtual copies of the former. A *virtual copy* only contains a pointer to the current state of the LTL_3 monitor. The complexity of starting a "new" virtual monitor there becomes $\mathcal{O}(1)$.

The complexity of updating all monitors is $\mathcal{O}(\mathrm{n})$ where n is the number of virtual monitors. This number depends on the number of observations. The worst-case situation corresponds to a system where all observed states are unique and all monitors remain inconclusive forever. Such system is fairly unlikely. Our running example and the evaluation case study in Chapter 9 suggest that monitors have a short life which reduces the cost of updating monitors.

Other monitoring techniques such as [Gia01; Hav04; Tha05; dAm05] might be used to determine the satisfaction of the formula $C \wedge \Theta OC$. LTL_3 technique, however, reports both violation and satisfaction of the formula as early as possible. Its three-value semantics distinguishes cases where a formula is satisfied, not satisfied, or none applies. Techniques such as [dAm05] amalgamate the last two cases. Other monitoring techniques, such as [Bas11], support richer logics

such as metric temporal logic and first-order logic. These techniques might be used in place of LTL_3; this might improve the applicability of the technique to industrial settings.

***(d)* Obtaining monitored satisfaction rates.** In this setting, the *monitored state probability* of state s is the ratio between the number of monitors started in s whose current state is labeled as T (true), and the number of monitors started in s whose current state is labeled as T (true) or F (false).

In our example, the monitored state probability of state A is not available for the two first observations; it is equal to 1 for the five next observations, and to .5 for the last one. The satisfaction rate of our obstacle is the upper bound of these state probabilities. It changes from 'not available' to 100% at observation 2, then decreases from 100% to 50% at observation 7 (see the bottom of Figure 7.5).

Figure 7.7 shows the evolution of the satisfaction rate of the obstacle Dusty Environment in our simulated environment. The black line shows the computed satisfaction rate. The graph was produced by monitoring the system with the tool described in Chapter 8.

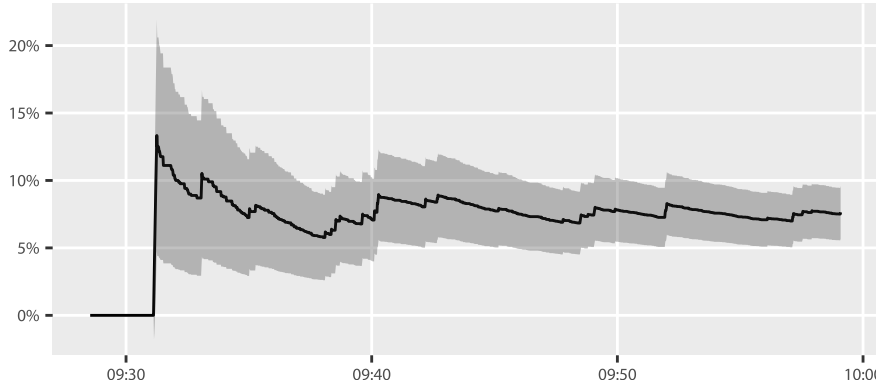


Figure 7.7 Monitored satisfaction rate of Dusty Environment.

To efficiently obtain the satisfaction rate of leaf obstacles, a list of monitors is kept in memory for each observed state; the number of behaviors satisfying the formula $C \wedge \Theta OC$ and the number of behaviors violating it are kept in registers.

Once a monitor reaches a monitor state labeled as *T* (true) or *F* (false), it can be removed from the list and the corresponding register is updated. Computing the state probability is then reduced to arithmetic operations on these registers.

7.3.3 Bounding inaccuracy margins

To mitigate the risk of unnecessary system adaptations, 'enough' observations should be made before deciding whether or not an adaptation is required. Otherwise, decisions might be based on non-statistically significant data, possibly leading to adaptations that deteriorate the system instead of improving it.

For example, assume 4 behaviors—2 positive and 2 negative ones. We might be tempted to estimate the state probability to 0.5. However, it is very likely that the real probability is not .5. It can be shown that there is a 95% of chance for the 'exact' value to lie between .01 and .99, which is not very informative.

To address this problem, we may compute the number of observations required to achieve a specified level of accuracy. Every observation triggers a new virtual monitor which increases the number of available data.

For example, let us require that the monitored satisfaction rate of a leaf obstacle should differ from the exact value by at most 1% with 95% certainty, that is, 95% of the values lies within $\pm 1.96\sigma$ of the exact value, where σ is the standard deviation. This interval holds if the variation around the exact satisfaction rate has a normal distribution [Wac07]. In our example, 1.96σ must be equal to .01. As the value of σ is not known, the following unbiased estimator provides an estimate [Wac07]:

$$\hat{\sigma} = \sqrt{\frac{p \times (1-p)}{n}},$$

where p is the estimated satisfaction rate for the considered leaf obstacle. Replacing σ by $\hat{\sigma}$ in our example results in the following expression:

$$1.96\hat{\sigma} = 1.96\sqrt{\frac{p \times (1-p)}{n}} = 0.01.$$

This expression can be solved to compute the number n of observations that must be made to guarantee a satisfactory level of accuracy. If the estimated satisfaction rate of the obstacle Dusty Environment is 20%, we have:

$$n = \left(\frac{1.96}{0.01}\right)^2 \times .8 \times .2 \approx 6146$$

Our accuracy bound thus requires 6146 observations of the state with the lowest state probability. With one observation every second, it requires at least 1h 42m 26s. Note that this is a lower bound; as discussed in Section 3.3.1, the satisfaction rate is the lowest state probability; to reach that uncertainty, the system has to spend the required timespan in this state with the lowest state probability. This indicates how fast our technique may compute a statistically accurate monitored satisfaction rate. (Note that such limitation applies to any monitoring technique with the same amount of observed data and the same statistical guarantees.)

In the opposite direction, we may compute the uncertainty margins given a timespan of observation. Assume that we make an observation every second for 30 minutes; we therefore have 1800 observations. The estimated standard deviation is:

$$\hat{\sigma} = \sqrt{\frac{p \times (1-p)}{n}} = \sqrt{\frac{.8 \times .2}{1800}} = 0.009428...$$

The 'exact' monitored satisfaction rate lies between 78% and 81% with 95% of certainty; this is already a narrow interval. Such information might be used at runtime to decide whether it is reasonable to adapt the running software.

In our example, we used the estimated satisfaction rate $p = 0.2$. If no estimated satisfaction rate p is available, it is common to use $.5$; this value maximizes the number n of observations. This guarantees the accuracy for any satisfaction rate.

Figure 7.7 shows in dark gray the evolution of the 95% confidence interval as more observations are made. We can see that the confidence interval is large at the begining where few observations are available, and narrow around the satisfaction rate when more observations are available.

7.4 Obstacle-driven system adaptation

This chapter proposes to let probabilistic goals and obstacles drive the runtime adaptation process. A system adaptation is required at runtime when the current configuration of countermeasures does not guarantee the required satisfaction rate (RSR) of the system's high-level goals. This provides a criteria for deciding when and why an adaptation is required. It also documents how the software system should adapt.

The *actual* satisfaction rate of these goals must therefore be determined from the *monitored* satisfaction rates of leaf obstacles. When this actual satisfaction rate falls below the goal's RSR, alternative countermeasures maximizing its satisfaction rate of these goals should replace the corresponding ones in the current configuration.

- The up-propagation procedure described in Section 4.2.1 determine the actual satisfaction rate of high-level goals from the monitored satisfaction rate of leaf obstacles.
- The countermeasure selection procedure described in Section 5.3 is then used to select most appropriate countermeasures.
- Countermeasures are then integrated in the ideal model and deployed in the running system—Section 7.5 will describe next.

If no selection guarantees the goal's required satisfaction rate, the selection that maximizes the satisfaction rate of the high-level goals is picked. The monitoring system might also prompt the user for interactively defining new countermeasures.

Back to our example, Figure 7.8 shows the satisfaction rate of the goal Achieve [Locals Warned When Risk Imminent], computed from the monitored satisfaction rates of the leaf obstacles. (See Figure 7.2 for the goal model.) Figure 7.9 shows the monitored satisfaction rates for our obstacles. Every 5 minutes, the optimization process computes a most appropriate countermeasure selection. At 08:26 and 09:21, an adaptation is required as the goal's satisfaction rate falls below its required satisfaction rate.

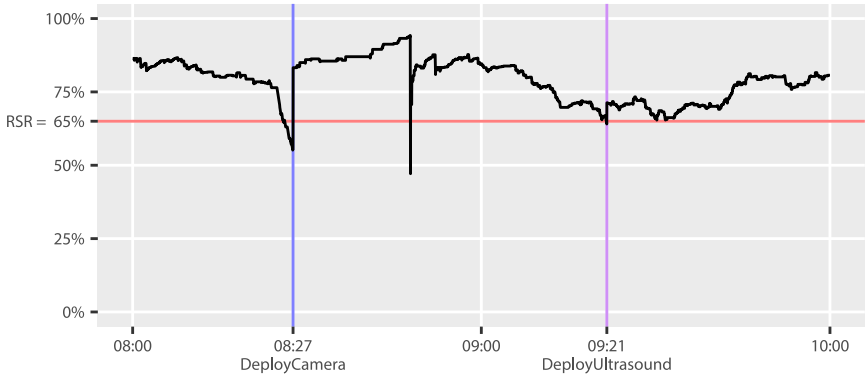


Figure 7.8 Monitored satisfaction rate of the goal Achieve [Locals Warned When Risk Imminent].

The peak at 08:47, seen in Figure 7.8 and in Figure 7.9(Noisy Image), illustrate the problem with statistically innaccurate satisfaction rates discussed in Section 7.3.3. Here, it did not trigger an adaptation as the violation was very short (it lasts 2 seconds).

7.5 Runtime deployment of most appropriate countermeasures

When more appropriate countermeasures are selected and integrated in the goal model, the running software system should be adapted to match the updated goal model.

The software component responsible for adapting the running system is named *Adaptor* in the following discussion. This component keeps track of a current selection of countermeasures. When new most appropriate countermeasures are computed, it identifies the countermeasures to be *(i) added*—as not found in the current selection but in the selection of the most appropriate ones; and *(ii) removed*—as no longer in the selection of the most appropriate ones.

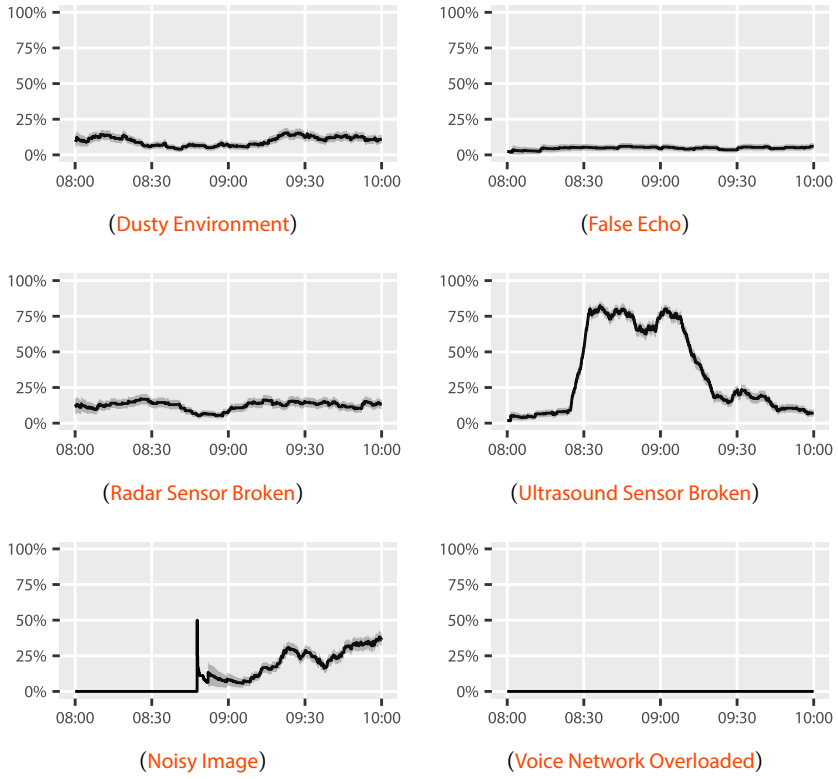


Figure 7.9 Monitored satisfaction rate of the leaf obstacles.

To adapt the software system, the *Adaptor* calls the activation and deactivation procedures associated with the countermeasures to be added or removed. The *activation* procedure is the code-related procedure responsible for the deployment of the corresponding countermeasure in the running system. The *deactivation* procedure is responsible for its removal. These procedures are used to:

- add, remove or replace a running component;
- update configuration parameters;
- activate hardware components;
- and so forth.

The countermeasure goals are decorated with these procedures.

For example, let us consider that the monitored satisfaction rate of the leaf obstacle Ultrasound Sensor Broken increases at 08:26, as shown in Figure 7.9. This increase causes a decrease in the monitored satisfaction rate of the high-level goal Achieve [Locals Warned When Risk Imminent] below its RSR (as seen in Figure 7.8). This decrease causes the countermeasure Achieve [Speed Acquired Every 5 Sec By Camera] to be selected as most appropriate countermeasure. The activation procedure for Achieve [Speed Acquired Every 5 Sec By Camera], namely, *DeployCamera*, is called and replaces the software component acquiring the speed by the camera-related component. As Figure 7.8 shows, the satisfaction rate of the high-level goal increases above its RSR after software adaptation. At 09:21, the satisfaction rate of Achieve [Locals Warned When Risk Imminent] falls again below the required satisfaction rate. This fall is due to an increase in the satisfaction rate of the obstacle Noisy Image for the selected countermeasure goal. This cause the countermeasure goal Achieve [Speed Acquired Every 5 Sec By Camera] to be removed from the selection, causing the deactivation procedure *DeployUltrasound* to be activated. The satisfaction rate then increases back.

The problem of runtime software reconfiguration is known to be complex [Ghe12; Nah16]. This problem is by no means addressed here; the sketched approach is hiding the reconfiguration process behind activation/deactivation procedures.

7.6 Updating uncertain probability values at runtime

Managing uncertainty at runtime is a critical challenge [Ben10]. Chapter 6 discussed how knowledge uncertainty can be integrated with obstacle assessment and control.

Such knowledge uncertainty may be reduced at runtime using the monitored information. However, if the observed values vary a lot over time, the knowledge uncertainty might increase, as 'actual' value is indeed uncertain.

To integrate knowledge uncertainty at runtime, the current monitored satisfaction rate should no longer be expressed as a single-value but rather as a multi-value satisfaction rate.

The process for obtaining the monitored satisfaction rate is updated as follows.

(a) Observed states are collected;

(b) (Optional step) Those observed states are filtered;

(c) Monitors are created and updated according to the observed states;

(d) Monitors labeled with T (true) and F (false) are counted; the ratio between the number of T- and F-monitors provides the monitored satisfaction rate.

(e)* A *Beta* probability distribution is fitted to the number of monitors.

(f)* The *Beta* distribution updates the current monitored satisfaction rate using Bayesian inference. The updated satisfaction rate then becomes the new satisfaction rate.

Step *(e*)* is straightforward as the parameters of the *Beta* distribution are, α, the number of positive monitors and, β, the number of negative monitors [Voso8].

The computation in step *(f*)* relies on Bayesian Inference; it is a statistical computation in which the Bayes' theorem is used to update a probability distribution as more information becomes available [Waco7]. The updated satisfaction rate $P(O \mid M)$, where M represents what has been last observed, is given by:

$$P(O \mid M) = \frac{P(M \mid O) \times P(O)}{P(M)}$$

where $P(M \mid O)$ is the likelihood of observing the monitored satisfaction rate, $P(O)$ is the current monitored satisfaction rate, and $P(M)$ is a scaling factor. The Beta distribution obtained at step *(e*)* captures the likelihood $P(M \mid O)$.

Figure 7.10 shows how the satisfaction rate of the obstacle Pump Mechanical Failure is updated with the observed satisfaction rate. The lighest line shows the satisfaction rate provided by the experts, as seen in Chapter 6. The darkest line shows the satisfaction rate updated after 6 observations. The other shades show the intermediate steps. For this example, we assumed that the monitored

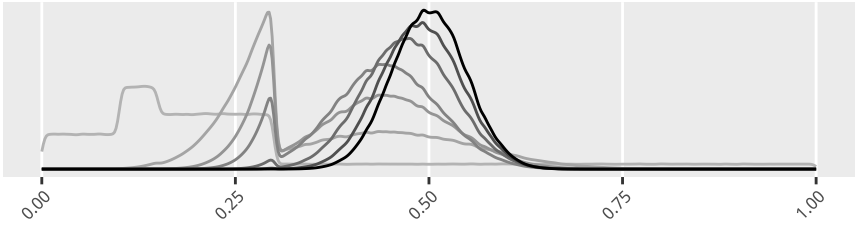


Figure 7.10 Monitored satisfaction rate of the leaf obstacles with knowledge uncertainty.

satisfaction rate was .45 for the first three observations and .55 for the three last. We can see that the more observations are made, the more accurate the satisfaction rate is (the curve gets narrower.)

7.7 Summary

Software systems should ideally self-adapt to changing conditions and assumptions in order to keep their goals satisfied. This chapter detailed an obstacle-driven runtime adaptation approach aimed at increasing the actual satisfaction rate of probabilistic system goals. Leaf obstacles are monitored at runtime to let the system dynamically switch to more appropriate countermeasure goals that increase the satisfaction rate of the system's high-level goals under new conditions. The approach guarantees that the required satisfaction rate of high-level goals remains satisfied when obstacle satisfaction rates are changing.

This requires obstacle satisfaction rates to be monitored at system runtime. The proposed monitoring technique extends the LTL_3 approach in [Bau11] to support the monitoring of probabilistic assertions. The monitors are built at RE time; at runtime, virtual copies of LTL_3 monitors keep track of whether observed behavior satisfy obstacle specifications. State probabilities for each observed state can thereby be estimated; their upper bound provides the monitored obstacle satisfaction rates. The latter are propagated through the obstacle/goal model up to the system's high-level goals. The monitored satisfaction rates thereby obtained for these goals are compared with their required satisfaction rate; when the former fall below the latter, more appropriate countermeasures replace the current ones to remain above the required threshold.

The next chapter describes the tools supporting the various techniques presented in the thesis.

Chapter 8

Tool support

The techniques presented in the previous chapters involve a significant amount of computation. For example, computing the satisfaction rate of high-level goals together with their uncertainty margins typically requires a large number of propagations to achieve reasonable precision. Manual propagation or the use of standard spreadsheets appears not feasible in practice. Other examples include the computation of most appropriate countermeasures, which requires computing the satisfaction rate of every countermeasure combination.

Dedicated tools are thereby needed to apply our techniques in a realistic setting. This chapter discusses the toolset, called KAOSTools, supporting our techniques. These tools were used in the presented running examples troughout the preceding chapters and evaluated on larger models in Chapter 9.

Unlike [Dar97], the proposed toolset is freely available together with its source code enabling other researchers to easily reuse or extend the tool suite. The tool suite accepts a textual declaration of goal and obstacle models that can be easily shared and integrated with existing versionning tools. The proposed tool suite was designed to run on remote servers through command-line tools.

This chapter is organized as follows. Section 8.1 describes the declaration language allowing analysts to provide a textual declaration of the goal and obstacle models. Section 8.2 describes the architecture of the KAOSTools toolset. Section 8.3 details the command-line tools available for applying the techniques.

All tools described in this chapter are written in C# and run on Windows, Linux, and MacOS. The source code is freely available under MIT Licence [Cai17b; Cai17d; Cai17f]. Packaged versions are also available in NuGet [NuGet].

8.1 KAOSTools: The declaration language

The techniques presented in the previous chapters rely on a goal model and its corresponding obstacle model. This section presents the declaration language, as a textual format, used for input to the tools.

In this language, element declaration start with the keyword **declare** and end with the keyword **end**. All elements have a unique identifier used to reference the element. Attributes precisely declare each element. They start with a keyword, such as **name** or **definition**, and end with a value, such as a string or an integer. The accepted values depend on the attribute.

The following subsections describe the attributes for each element, such as goals and obstacles, and provide examples of such declarations.

Goals, soft goals, domain properties, and domain hypotheses

To declare a goal, the keyword **declare** must be followed by the keyword **goal**. Code 8.1 provides an example of declaring the goal Achieve [Make Up Water Provided When Loss Of Cooling]. The unique identifier **make_up_water_provided** immediately follows the keyword **goal** between square brackets. Comments start with a dash (**#**) and end at the end of the line. Table 8.1 describes the attributes for declaring goals.

```
declare goal [ make_up_water_provided ]
  name "Achieve [Make Up Water Provided When Loss Of Cooling]"
  definition
    "Make up water shall be provided when loss of cooling occurs."
  rsr .99 # or rsr 99%
  refinedby response_to_loss_of_cooling, pump_on_when_activated
end
```

Code 8.1 Declaration of the goal Achieve [Make Up Water Provided When Loss Of Cooling].

Attribute	Used for	Accepted value
`name`	Name	String
`definition`	Definition	String
`refinedby`	Goal AND-Refinement	List of identifiers
`rsr`	Required Satisfaction Rate	Number or percentage
`obstructedby`	Goal obstruction	Identifier
`assignedto`	Goal assignment	Identifier
`formalspec`	Formal specification	Formal specification

Table 8.1 Attributes for goal declarations

Goal refinements are declared using the keyword **`refinedby`**. The list of identifiers refers to other goals, domain properties or domain hypotheses in the corresponding AND-Refinement. If an identifier is not explicitly defined, a goal with corresponding identifier is added to the model. For example, Code 8.1 specifies a single AND-Refinement with two subgoals, namely, **`response_to_loss_of_cooling`** and **`pump_on_when_activated`**.

Multiple **`refinedby`** attributes are used to declare alternative refinements. Goal refinements can be decorated with their refinement pattern using the name of the pattern between square brackets, as seen in Code 8.2. In addition, a partial entailment can be declared for the subgoals; the probability to satisfy the parent goal alone is declared between brackets after the subgoal identifier. Code 8.2 declare that subgoal **`amb_mobilized_when_amb_allocated_on_road`** satisfy its parent goal with a probability .8.

```
refinedby [case]
  amb_mobilized_when_amb_allocated_on_road [.8],
  amb_mobilized_when_amb_allocated_at_station [.2]
```

Code 8.2 Declaration of the refinement pattern and partial entailment.

Similarly to goals, soft goals can be declared using the keywork **`softgoal`** with attributes **`name`** and **`definition`**. Domain properties and domain hypotheses are declared using the keyword **`domprop`** and **`domhyp`**, respectively. The available attributes are summarized in Table 8.2.

Attribute	Used for	Accepted value
`name`	Name	String
`definition`	Definition	String
`formalspec`	Formal specification	Formal specification

Table 8.2 Attributes for declarations of domain properties and hypotheses

Obstacles

Obstacles are declared using the keyword `obstacle` followed by their unique identifier between square brackets. Code 8.3 shows the declaration for the non-leaf obstacle Valve Not Opened And Requested. Table 8.3 shows the attributes available for declaring obstacles.

Attribute	Used for	Accepted value
`name`	Name	String
`definition`	Definition	String
`refinedby`	Obstacle AND-Refinement	List of identifiers
`resolvedby`	Obstacle resolution	Identifier
`formalspec`	Formal specification	Formal specification
`esr`	Estimated Satisfaction Rate	Number, probability distribution or percentage

Table 8.3 Attributes for obstacle declaration

Obstacle refinements. The declaration of obstacle refinements is similar to the declaration of goal refinements. The identifiers must refer to obstacles, domain properties or domain hypotheses. If an identifier is not explicitly defined, a new obstacle with corresponding identifier is added to the model. Code 8.3 declares four alternative refinements each with a single obstacle.

Goal obstructions. Obstructions to goals are declared using the keyword `obstructedby`. The value of this attribute is a single identifier referring to an obstacle—if not explicitly defined, a new obstacle will be added. Code 8.4 shows how the obstruction of Achieve [Make Up Pump Motor On When Water Requested] is declared.

```
declare obstacle [ valve_not_open ]
  name "Valve Not Opened And Requested"
  refinedby valve_failure
  refinedby no_power_available
  refinedby unavailable_due_to_maintenance
  refinedby valve_electronic_failure
end
```

Code 8.3 Declaration of the obstacles Valve Not Opened And Requested.

```
declare goal [ pump_motor_on ]
  ...
  obstructedby motor_not_on
end
```

Code 8.4 Declaration of the goal Achieve [Make Up Pump Motor On When Water Requested].

Obstacle satisfaction rates. Leaf obstacles are estimated by experts. The declaration language supports both single-value and multi-value satisfaction rates. Code 8.5 shows the declaration of the obstacle Valve Mechanical Failure with a single-value satisfaction rate of .002, whereas Code 8.6 shows the declaration of the obstacle Pump Mechanical Failure with a multi-value satisfaction rate. The supported multi-value satisfaction rates are summarized in Table 8.4.

```
declare obstacle [ valve_failure ]
  name "Valve Mechanical Failure"
  resolvedby [restoration:make_up_water_provided]
             cooling_system_repaired
  esr 0.002
end
```

Code 8.5 Declaration of the obstacle Valve Mechanical Failure.

As seen in Chapter 6, a quantile is a single probability value attached to a cumulative probability. To declare the cumulative probabilities, the model attribute `@experts.quantiles` must be provided. For example, Code 8.6

```
@experts.quantiles "(10%, 50%, 90%)"
declare obstacle [ manual_opening_valve_failure ]
  ...
  esr quantile[0.001,0.002,0.003]
end
```

Code 8.6 Declaration of the obstacle Valve Not Opened When Valve Control Not Unavailable.

Value	Used for	Example
`beta`	Beta Distribution	`beta[2,4]`. A beta distribution with $\alpha = 2$ and $\beta = 4$. See [Vos08] for details on parameters α and β.
`uniform`	Uniform Distribution	`uniform[.1,.3]`. A uniform distribution between .1 and .3.
`triangular`	Triangular Distribution	`triangular[.1,.4,.6]`. A triangular distribution with a .1 min-value, .6 max-value and a .4 mode.
`pert`	PERT Distribution	`pert[.1,.4,.6]`. A PERT distribution with a .1 min-value, .6 max-value and a .4 mode. See [Vos08] for details about PERT distributions.
`quantiles`	Quantiles	`quantiles[.1,.4,.6]`. A list of quantiles.

Table 8.4 Multi-value satisfaction rates

specify that 10% of the values are below 0.001, 50% are below 0.002 and 90% of the values are below 0.003. The model attribute `@experts.quantiles` need only be declared once for the model.

Obstacle resolutions. Obstacles resolutions are declared using the keyword `resolvedby` followed by a goal identifier—if not explicitely defined, a new goal will be added. Code 8.5 shows the declaration of the resolution of obstacle Valve Mechanical Failure by the goal `cooling_system_repaired`.

Resolution tactics can be declared between square brackets (see Code 8.5), followed by the anchor goal (`make_up_water_provided` in the example). Table 8.5 summarizes the supported resolution tactics.

Resolution tactic	Used for
`substitution`	Goal substitution
`prevention`	Obstacle prevention
`obstacle_reduction`	Obstacle reduction
`restoration`	Goal restoration
`weakening`	Weakening
`weak_mitigation`	Weak mitigation
`strong_mitigation`	Strong mitigation

Table 8.5 Resolution tactics

Agents

Agents are declared using the keyword **`agent`** followed by their unique identifier between square brackets. Table 8.6 summarizes the attributes for agents. Code 8.7 provides an example of agent declaration.

```
declare agent [ operator ]
  name "Operator"
end
```

Code 8.7 Declaration of the agent Controlling Software.

Agent assignments. An agent assignment to a goal is declared using the keyword **`assignedto`**. Code 8.8 shows how the Operator agent is assigned to the goal Achieve [Valve Opened When Unavailable Due To Maintenance]. If not explicitly defined, an agent with corresponding identifier is added to the model.

```
declare goal [ manual_opening_of_valve ]
  name "Achieve [Valve Opened When Unavailable Due To Maintenance]"
  obstructedby manual_opening_valve_failure
  assignedto operator
end
```

Code 8.8 Declaration of the goal Achieve [Valve Opened When Unavailable Due To Maintenance].

Attribute	Used for	Accepted value
`name`	Name	String
`definition`	Definition	String
`type`	Whether the agent is a *software* or a *environment* agent.	`software` or `environment`

Table 8.6 Attributes for agent declaration

Predicates and formal declarations

Formal specifications of goals and obstacles can be provided using the keyword `formalspec`. The language supports first-order, Linear Temporal Logic assertions. A complete BNF grammar is available in Appendix B. Code 8.9 provides an example of a formal specification—here, the LTL formula

$$\Diamond(dieselGeneratorRequested \\ \wedge \Box_{\geq 5min} \neg(\exists g : DieselGenerator \cdot generatorStarted(g)))$$

```
declare obstacle [ diesel_generator_not_started ]
  name "Diesel Generator Not Started"
  formalspec
    sooner-or-later ( dieselGeneratorRequested() and
      always, for more than 5 minutes,
      not (exists g: DieselGenerator . generatorStarted(g))
    )
  probability 0.002
end
```

Code 8.9 Formal specification of the obstacle Diesel Generator Not Started.

The identifier `generatorStarted` refers to a predicate with a single argument. Predicates can be explicitly declared using the keyword `predicate` followed by a unique identifier. Code 8.10 shows the declaration of the predicate GeneratorStarted. Table 8.7 provides the attributes for declaring predicates.

```
declare predicate [ generatorStarted ]
  name "GeneratorStarted"
  argument g: dieselGenerator
  formalspec g.Started
end
```

Code 8.10 Declaration of the predicate ValveOpened.

As seen in Code 8.10, predicates may have arguments. In the example, the formal specification states that the predicate is true if the attribute `Started` is true for the instance of `DieselGenerator` provided as an argument. When referring to a predicate in a formal specification, the actual arguments must match the order and the respective entity types declared in the argument list.

Entities, associations and types

Formal specifications refer to entities and associations to express their truth value. All referenced entities, entity attributes, associations, association attributes and types from the object model associated with the considered goal model are automatically inferred from the formal specification of goals, domain properties, domain hypotheses, obstacles, and predicates. This appears particularly useful for checking whether formal specifications refer to the appropriate elements and to avoid similar names (e.g. `dieselGenerator` and `diesel_generator`).

The analyst might enrich the object model by explicitly defining elements. Tables 8.8, 8.9 and 8.10 provide the attributes for declaring entities, associations and given types.

Entity and association attributes. An entity or association attribute is composed of an attribute identifier and a given type. A given type is a base type, such as *Boolean* or *String*, or a domain-specific type such as *MotorStatus*. The latter are declared using the `type` keyword. Code 8.11 shows an example of attribute declaration.

Attribute	Used for	Accepted value
`name`	Name	String
`definition`	Definition	String
`formalspec`	Formal specification	Formal specification
`argument`	Formal arguments	An identifier followed by a colon (`:`) and an identifier

Table 8.7 Attributes for predicate declaration

Attribute	Used for	Accepted value
`name`	Name	String
`definition`	Definition	String
`type`	Entity type	`software`, `environment` or `shared`
`isa`	Inheritance	Identifier
`attribute`	Attribute	An identifier optionally followed by a colon (`:`) and an identifier

Table 8.8 Attributes for entity declaration

Attribute	Used for	Accepted value
`name`	Name	String
`definition`	Definition	String
`attribute`	Attribute	An identifier optionally followed by a colon (`:`) and an identifier
`link`	Linked entity	An identifier

Table 8.9 Attributes for association declaration

Attribute	Used for	Accepted value
`name`	Name	String
`definition`	Definition	String

Table 8.10 Attributes for given type declaration

```
declare entity [ dieselGenerator ]
  name "DieselGenerator"
  attribute Started: bool
end
```

Code 8.11 Declaration of the entity DieselGenerator.

```
declare association [ generator_valves ]
  name "GeneratorValves"
  link [1] dieselGenerator
  link [1,N] electroValve
end
```

Code 8.12 Declaration of the association GeneratorValves.

Linking associations and entities. Entities are connected to other entities through associations. The attribute **`link`** declares the association connecting entities, optionally with corresponding multiplicities [Lam09]. Code 8.12 provides an example with an association linking the entities DieselGenerator and ElectroValve. In the association, there is at most one diesel generator, but one or more corresponding valve(s).

Experts and calibration variables

To support the techniques presented in Chapter 6, the declaration language was extended to explicitly declare experts and calibration variables. Experts are declared using the keyword **`expert`** followed by their unique identifier. Calibration variables are declared using the keyword **`calibration`** followed by their unique identifier. Code 8.13 declares two experts and one calibration variable. Tables 8.11 and 8.12 describes the attributes availables.

For a calibration variable, the reference value is declared using the **`esr`** attribute without declaring an expert. The estimates by experts are declared using the **`esr`** attribute with the corresponding expert between square brackets. When declaring the estimated satisfaction rate by an expert in an obstacle, the expert is also declared in square brackets (see Code 8.14).

```
declare expert [ expert1 ] name "Expert 1" end
declare expert [ expert2 ] name "Expert 2" end

declare calibration [ calibration1 ]
  name "Push Button Broken"
  esr 4.37%
  esr[expert1] quantile[3.812%, 4.285%, 4.759%]
  esr[expert2] quantile[3.961%, 4.453%, 4.945%]
end
```

Code 8.13 Declaration of the association GeneratorValves.

Attribute	Used for	Accepted value
`name`	Name	String

Table 8.11 Attributes for expert declaration

Additional features

In addition to the constructs presented before, it is possible to define custom attributes, split the model into different files, and declare model managment attributes such as the title of the model, the version, and the authors. Utilities are also available to help analysts writing declarations.

Custom attributes. All elements can have extra attributes not declared by the grammar. These custom attributes must start with $. Such extensions proved usefull to support specific tools without changing the grammar. For example, Code 8.15 shows how activation and deactivation procedures are declared—our runtime adaptation tool did not require changes to the declaration language.

Splitting the model into multiple files. It is easier to handle a large model by splitting it into multiple files. Submodels can be imported using `import` followed by the path to the submodel. Adding attributes to an element already declared facilitates the splitting into multiple files. The element is 'declared' using the keyword `override`, in place of `declare`, to add an attribute. This proved useful to split goals in a specific file and obstacles in another one while declaring the obstructions in the file with the obstacle declarations. Code 8.16 shows an example of such usage.

Attribute	Used for	Accepted value
`name`	Name	String
`definition`	Definition	String
`esr`	Estimated Satisfaction Rate	Number, probability distribution or percentage

Table 8.12 Attributes for declaration of calibration variables.

```
declare obstacle [ power_cable_failure ]
  name "Power Cabling Failure"
  esr[expert1] quantile[0.015,0.01752,0.02]
  esr[expert2] quantile[0.001,0.002,0.003]
end
```

Code 8.14 Declaration of the association GeneratorValves.

Model attributes. Code 8.17 shows how the title, the authors and the version of the considered goal model can be declared.

Utilities. A Checker tool is available for checking the correct syntax and providing statistics for the model. For example, this tool may produce the following:

```
$ mono Checker.exe model.kaos
Goals: 20
Root goals: 1
Leaf goals: 14
Goal refinements: 6

Obstacles: 16
Root obstacles: 6
Leaf obstacles: 12
Obstacle refinements: 11

Resolutions: 10

Generated goal exceptions: 9 (distributed over 3 goals)
Generated provided assumption: 23 (distributed over 6 goals)
```

```
declare goal [ speed_acquired_by_camera ]
  name "Achieve [SpeedAcquiredEvery5SecondsByCamera]"
  refinedby images_acquired,
            tracers_identified,
            speed_acquired_from_tracers
  $ondeploy "DeployCamera"
  $onwithold "DeployUltrasound"
end
```

Code 8.15 Declaration of the association GeneratorValves.

```
# In the file 'goal.kaos'
declare goal [ manual_opening_of_valve ]
  name "Achieve [Valve Opened When Unavailable Due To Maintenance]"
  assignedto operator
end
...
import "obstacle.kaos"

# In the file 'obstacle.kaos'
override goal [ manual_opening_of_valve ]
  obstructedby manual_opening_valve_failure
end
declare obstacle [ manual_opening_valve_failure ]
  name "Valve Not Opened When Valve Control Not Unavailable"
  probability 0.0001
end
...
```

Code 8.16 Splitting model declaration accross multiple files.

The tool OmnigraffleExport generates goal and obstacle diagrams corresponding to the graphical syntax of the model provided in textual format. The analyst can edit the generated diagrams using OmniGraffle [Ols10]. This tool was used to produce all goal and obstacle diagrams presented in the thesis. Last, a bundle is available for the TextMate text editor [Gra07] to provide syntax highlighting.

```
@title "Spent Fuel Pool"
@author "Antoine Cailliau, Axel van Lamsweerde"
@version "0.1"
```

Code 8.17 Model attributes.

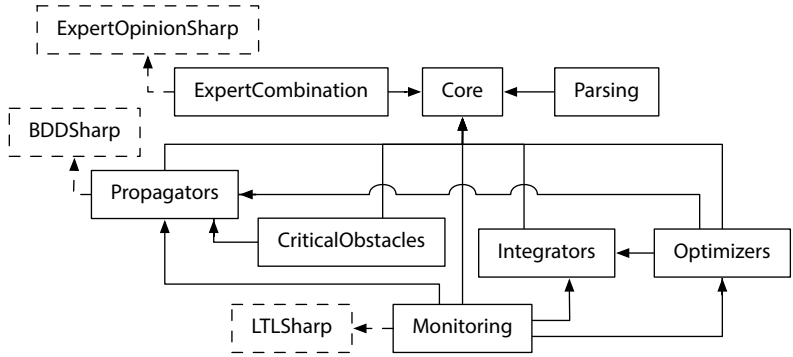


Figure 8.1 KAOSTools Architecture.

8.2 KAOSTools: Architecture

Figure 8.1 shows the architecture of the KAOSTools tool suite. Boxes refer to libraries, dashed boxes to external libraries. The latter were developed to support KAOSTools and might be used outside the framework. Arrows indicate *USE* links [Med10]. The architectural modules are briefly described below.

Core. This package contains all classes representing the probabilistic goals, domain properties, obstacles, and so forth. It mainly supports the techniques in Chapter 3 together with other libraries through *helpers* for selecting, modifying, and creating KAOS model elements.

Parsing. This package contains all the classes related to the parsing of the KAOSTools declaration language. On parsing, a model is built in memory using the helpers provided in Core.

Propagators. This package implements the techniques presented in Chapter 4. The classes there implement both the Pattern-based and BDD-based computations. The package also supports the computation of satisfaction rates with their

uncertainty margins. This library relies on BDDSharp to support the BDD-based computation. BDDSharp is a freely available library to create and manipulate Reduced Order Binary Decision Diagrams (ROBDDs) in C# [Cai17d]. The ease of deployment and the availability on multiple OSes motivated the creation of the library; If needed, more efficient BDD libraries can replace BDDSharp in the KAOSTools suite.

CriticalObstacles. This package implements the techniques presented in Chapter 4 to highlight most critical obstacles. This package also supports such highlighting taking uncertainty margins into account as presented in Chapter 6. This library also generates violation diagrams.

Integrators. This package implements the techniques presented in Chapter 5 for integration of countermeasures. The package supports both *hard* integration that modifies the goal/obstacle model, and *soft* integration using the constructs presented in Section 5.4.3.

Optimizers. This package implements the techniques presented in Chapter 5 for selection of most appropriate countermeasures. Single-value and multi-value leaf obstacle estimates are both supported.

ExpertCombination. This package implements the techniques presented in Chapter 6. It supports the combination of multiple experts estimates using both Cook's and Mendel-Sheridan's techniques. This library relies on ExpertOpinionSharp to combine expert opinions. The latter was developed specifically for combining expert estimates using calibration variables as a separate library [Cai17e]; other approaches can thereby reuse these combination techniques.

Monitoring. This package supports the techniques presented in Chapter 7. The library relies on LTLSharp for the generation and update of LTL_3 monitors. LTLSharp is a library that provides model checking algorithms, algorithms for building Buchï automata, and LTL_3 monitors for C# libraries and applications [Cai17f].

8.3 KAOSTools: Tool suite

In addition to the various libraries developed to support our techniques, a series of command-line tools enables the analyst to run the techniques on their goal/obstacle models. This section briefly describes the tools. More detailed descriptions are available through the option `--help` to the command-line tools.

Utils.Propagator. This tool computes the satisfaction rate of the input high-level goals.

```
$ mono Propagator.exe --root=make_up_water_provided model.kaos
Achieve [Make Up Water Provided When Loss Of Cooling]: 80.31 %
```

Utils.UncertaintyPropagator. This tool computes the satisfaction uncertainty and violation uncertainty for the input high-level goals.

```
$ mono UncertaintyPropagator.exe --root=make_up_water_provided \
  model-uncertainty.kaos
Achieve [Make Up Water Provided When Loss Of Cooling]:
  Mean: 78.67 %
  Required Satisfaction Rate: 80.00 %
  Violation Uncertainty: 71.80 %
  Uncertainty Spread: 0.02694
```

Utils.ViolationDiagram. This tool generates the data for violation diagrams and violation diagrams with uncertainty. The latter ones can then be generated using R [Teaoo].

Utils.CMIntegrator. This tool provides an interactive console for integrating countermeasures into an original goal model. The latter can then be exported.

```
$ mono CMIntegrator.exe model.kaos
> resolve pump_failure
[0] Achieve [Cooling System Repaired]
[1] Achieve [Redundant Pump Motor On When Primary Pump Failure]
Select the countermeasure goal to integrate: 0
> export model_with_countermeasure.kaos
```

Utils.CMSelector. This tool returns a list of most appropriate countermeasures. **Utils.CMSelectorUncertainty** returns a list of most appropriate countermeasures when leaf obstacles are estimated together with their uncertainty margins.

```
$ mono CMSelectorUncertainty.exe --root=make_up_water_provided \
  model-uncertainty.kaos
Violation Uncertainty without countermeasures: 71.80%
Uncertainty Spread without countermeasures: 0.02694
Required Satisfaction Rate: 80.00%
Minimal cost to guarantee RSR: 1
Optimal selections (2):
* [OptimalSelection: Resolutions={cooling_system_repaired},
                     Cost=1,
                     ViolationUncertainty=0,
                     UncertaintySpread=0]
* [OptimalSelection: Resolutions={redundant_pump_started},
                     Cost=1,
                     ViolationUncertainty=0,
                     UncertaintySpread=0]

--- Statistics ---
Number of countermeasure goals: 6
Number of possible selections: 63
Number of safe selections: 48
Number of tested selections (for minimal cost): 63
Number of tested selections (for optimal selection): 6
Maximal safe cost: 6
Time to compute minimal cost: 00:00:00.3229049
Time to compute optimal selections: 00:00:00.3229049
```

Utils.Monitor. This tool runs the monitoring of probabilistic obstacles, selects more appropriate countermeasures and triggers the activation/deactivation procedure to be applied. Section 9.3.3 provides concrete details about the monitoring infrastructure in action on a case study.

8.4 Summary

This chapter briefly presented the tool suite supporting the techniques discussed in the previous chapters. The tools accept a textual declaration of the goal/obstacle model as input. The modular architecture of KAOSTools enables new techniques to be implemented and integrated while leveraging the existing ones.

The tools presented in this chapter were used to generate all diagrams, charts, and computation results provided in the various examples throughout the thesis. These were also used to evaluate our techniques, as presented in the next chapter.

Chapter 9

Evaluation

This chapter illustrates the techniques proposed in the previous chapters while showing them in action on real case-studies of significant size. Our techniques are evaluated according to five hierarchical criteria. Each criterion is a prerequisite for the next [Dam14; Let14]. The criteria are the following:

- *Correctness*: Are the proposed techniques correct with regard to their specifications?
- *Performance and Scalability*: Are the techniques efficient for realistically-sized problems?
- *Applicability*: Are the techniques applicable to real-world problems?
- *Utility*: Are the techniques solving a real problem?
- *Usability*: Are the techniques usable by risk analysts and requirements engineers?

These criteria here refer to the techniques presented in the thesis, namely:

- the propagation procedures as seen in Section 4.2.1 and Section 4.2.2;
- the procedure for identifying critical obstacles as seen in Section 4.3 and Section 6.2.3;
- the selection of most appropriate countermeasures as seen in Section 5.3;
- the integration of countermeasures into the goal model as seen in Section 5.4;

- the assessment of leaf obstacles together with uncertainty margins as seen in Section 6.2;
- the runtime monitoring of probabilistic obstacles as seen in Section 7.4.

As for applicability, the case studies considered for application are real, mission-critical systems with non-trivial requirements and significant risks that could prevent their correct operation. They includes a Car Pooling System, an industrial Yoke Lifting System, and an Ambulance Dispatch System.

The chapter is structured as follows. Section 9.1 discusses the correctness of our techniques. Section 9.2 briefly presents the complexities and the performance of the proposed approach. Section 9.3 describes the three case-studies, evaluates the applied techniques regarding their specific objectives, and briefly discusses the applicability of the techniques. Section 9.4 discusses the extent to which our techniques solve a real problem. Section 9.5 shows how our techniques are usable by risk analysts and requirements engineers.

9.1 Correctness

This section shows that the techniques proposed in Chapters 4-7 meet their specification.

In particular, the propagation procedures in Chapter 4 compute 'safe' estimates, that is, lower bound estimates. Otherwise, the computed goal satisfaction rates might overestimate the actual one. As a consequence, the goal would be stated to meet its required satisfaction rate even if does not.

Proposition 9.1. In a model with complete, consistent, and minimal goal refinements and with domain-complete obstacle refinements, the satisfaction rate of the considered high-level goals is greater or equal to the computed satisfaction rate computed by the propagation procedures in Chapter 4.

Proof. (Ab absurdum.) Assume a high-level goal and its lowest descendant G whose actual satisfaction rate is lower than the computed satisfaction rate. The latter goal is either refined or is leaf goal.

- If goal G is refined, its subgoals have an actual satisfaction rate greater or equal to their respective computed satisfaction rate. If not, G is not the lowest descendant as there exists a subgoal with an actual satisfaction rate lower than its respective computed satisfaction rate.

  – In the pattern-based computation, the satisfaction rate is computed using a specific propagation equation. Assuming that the satisfaction rates for the subgoals are correct, if the estimated satisfaction rate is lower than the computed satisfaction rate, then at least a term in the generic propagation equation was not taken into account; the appropriate specific propagation equation was therefore not applied.

  – In the BDD-based computation, the satisfaction rate would be lower than the computed one if and only if the obstruction supersets include fewer obstacles; however, by construction, all obstacles are included; there is no unidentified obstacles as the model is assumed to be domain-complete.

- If goal G is a leaf goal, its satisfaction rate is lower than the estimated one. The actual satisfaction rate of the root obstacle is then higher than the computed satisfaction rate. However, the computed satisfaction rate of the root obstacle might be higher than the actual one if and only if obstacles are missing. If obstacles are missing the model is not domain-complete. □

Proposition 9.2. The obstacles returned by the critical obstacle identification procedure cause a loss of satisfaction for the specified goal.

Proof. By definition of violation severity (Definition 3.11), obstacles with a positive violation severity cause a loss in the computed satisfaction rate of the high-level goal. The critical obstacle identification procedure presented in Section 4.3 returns all obstacles with positive violation severity. □

Proposition 9.3. Any countermeasure not returned by the procedure for selection most appropriate countermeasures costs more or would result in a lower computed satisfaction rate of the considered high-level goal.

Proof. If a countermeasure not returned by the procedure would cost less, its cost would be lower than the minimal one returned by the first optimization discussed in Section 5.3. The minimal cost is therefore not minimal. If this countermeasure would result in a higher computed satisfaction rate of the considered high-level goal, the second optimization would not return the highest solution. □

Proposition 9.4. The integration of a countermeasure in a complete, consistent and minimal model produces a complete, consistent and minimal model.

Proof. This is trivially enforced by the *correctness preservation* property, as seen in Section 5.4. □

As for our propagation techniques, we want the monitored satisfaction rate to be 'safe', that is, the monitored satisfaction rate of the leaf obstacles must be an upper bound. Otherwise, the monitored satisfaction rate might underestimate the actual satisfaction rate, while would lead to unsafe conclusions.

Proposition 9.5. For a given state decomposition, the monitored satisfaction rate of a leaf obstacle is lower or equal to the actual satisfaction rate obtained by the monitoring procedure in Section 7.3.

Proof. (Ab absurdum.) By definition, the monitored satisfaction rate of the formula $\Diamond(C \wedge \Theta OC)$ is the highest state probability to satisfy the obstacle condition $C \wedge \Theta OC$. If the actual satisfaction rate of a leaf obstacle is greater than the monitored satisfaction rate, there is a state probability greater than the monitored statisfaction rate; the monitored satisfaction rate is therefore not the highest state probability. □

Note. As seen in Section 7.3, the monitoring procedure relies on observed states. However, the same system might be observed with a different state decomposition. The latter may impact state probabilities. For example, consider the flooding system, and assume two states $RiverHigh$ and $RiverLow$. Both have a state probability to satisfy the obstacle Dusty Environment, say p_{high} and p_{low}. A decomposition with more states or with fewer states changes the values of state probabilities; this impacts the monitored satisfaction rate of obstacles and goals.

Consider a decomposition with one state *RiverLowHigh* instead of two. As a decomposition forms a partition over all possible states, this single state is a combination of the two states in the previous decomposition. The state probability $p_{low/high}$ for the single state is a combination of the two state probabilities:

$$p_{low/high} = t_{high} \times p_{high} + t_{low} \times p_{low},$$

where t_{high} and t_{low} represent the fraction of time spent in the corresponding state, respectively. However, given that the system might be in one of the two states but cannot be in both states, $t_{high} + t_{low} = 1$. No combination of the probabilities p_{high} and p_{low} can be higher than p_{high} or p_{low}. The state probability $p_{low/high}$ for the single state is therefore at least smaller than the state probabilities for the two states, i.e.,

$$p_{low/high} \leq p_{high} \qquad p_{low/high} \leq p_{low}.$$

Instead of a single state, consider now a decomposition with three states. Reversing the above reasoning, it is possible that one of the states in this decomposition has a state probability higher than the state probabilities p_{high} or p_{low}. The satisfaction rate of the leaf goal would then be higher, leading our technique to underestimate the satisfaction rate of an obstacle. This limitation calls for a finer-grained analysis of monitored satisfaction rates.

9.2 Performance and scalability

This section discusses the theoretical complexity of our techniques together with their performance in realistic problems.

Propagation procedures. The *Pattern-based computation* performs at most $g+o$ computations where g is the number of goals and o the number of obstacles. The complexity of this procedure is therefore $\mathcal{O}(\mathrm{g} + \mathrm{o})$.

The *BDD-based computation* performs at most $r_g + o_b + r_o$ BDD operations where r_g is the number of goal refinements, o_b is the number of obstructions, and r_o is the number of obstacle refinements. The complexity of a BDD operation is linear in the size of the combined BDDs [Mei12]. However, the size of the BDDs might be exponential in the number of variables [Mei12]. The size of our BDDs is $\mathcal{O}(2^{\mathrm{o}})$, where o is the number of obstacles. Computing the satisfaction

rate of an obstruction superset is linear in the number of nodes of the BDD. Therefore, the complexity of the BDD-based computation is bounded by $\mathcal{O}((\mathrm{r_g} + \mathrm{o_b} + \mathrm{r_o}) \times 2^{\mathrm{o}} + 2^{\mathrm{o}})$. In practice, however, BDDs often have a much smaller size than $\mathcal{O}(2^{\mathrm{o}})$. Given the specific structure of the encoded formula, we may expect tighter bounds on the size of the BDD.

In practice, the computation cost of the satisfaction rate of the considered high-level goals is often small. For example, computing the satisfaction rate of the high-level goal in the Ambulance Dispatch System and in the Car Pooling System discussed in Sections 9.3.3 and 9.3.1 hereafter took less than a second. The timing includes the time to start the .NET virtual machine, read, and parse the model. As seen in Section 4.2.3, the time to compute the satisfaction rates of high-level goals on large, randomly generated goal/obstacle models is small. As a recall, on models with 10.000 goals and 10.000 obstacles, it takes 13 seconds for the Pattern-based computation and 18 seconds for the BDD-based computation (including 5 milliseconds for computing the probability of an obstruction superset.)

When computing satisfaction rates with their uncertainty margins, the propagation procedure is repeated. The complexity of the repeated procedure is bounded by $\mathcal{O}(\mathrm{n} \times 2^{\mathrm{o}})$ where n is the number of samples computed. In practice, a value such as $n = 1,000,000$ provides sufficiently accurate satisfaction rates in less than 5 seconds for the Ambulance Dispatching System in Sections 9.3.3. Note that on a model with 10.000 goals and 10.000 obstacles, it would take about 1 h 20 m; this would call for specific techniques to be developed event though such model size appears questionable in practice.

Identification of critical obstacles. To identify likely and critical obstacles, the procedure in Section 4.3 generates all combinations of size 1, then 2, etc. The complexity for such identification of obstacle combinations of size i is $\mathcal{O}(\mathrm{o}^{\mathrm{i}})$, where o is the number of obstacles.

In practice, the identification of the critical obstacles and pairs of obstacles for the Car Pooling System in Section 9.3.1 took about a second. The identification of critical triplets took about 3 seconds whereas the identification of 4-tuples took 30 seconds. Similar times were observed for the running example,

the Ambulance Dispatch System in Section 9.3.3 and the C230 Yoke Lifting System in Section 9.3.2. Again, these timing indications includes the time to start the .NET virtual machine, and read and parse the models.

Selection of most appropriate countermeasures. The selection of most appropriate countermeasure solves two combinatorial optimization problems, as seen in Section 5.3. The complexity of solving these problems is bounded by $\mathcal{O}(2^c)$ where c is the number of countermeasures. As discussed in Section 5.3, future improvements are expected to reduce the computation effort.

In practice, selecting most appropriate countermeasures for the Ambulance Dispatching System in Section 9.3.3 took about 2 minutes.

Integration of countermeasures. Countermeasure integration using the constructs presented in Section 5.4.3 is in $\mathcal{O}(1)$. However, changes may need to be propagated in the goal/obstacle model; this requires, in the worst case, $\mathcal{O}(g + o)$ *Provided* annotations.

In practice, the time to integrate all countermeasures in the Ambulance Dispatching System Section 9.3.3 or in the running example took less than a second.

Runtime monitoring of probabilistic obstacles. As discussed in Section 7.3.2, the building of the monitors is in $\mathcal{O}(2^{2^n})$ where n is the size of the formula [Bau11]. The complexity of updating all monitors at runtime is $\mathcal{O}(n)$ where n is the number of virtual monitors.

As suggested by the Ambulance Dispatch System case study in Section 9.3.3 and the Flooding System in Section 7.2 the approach appears to be applicable in practice. For the former, building the required monitors takes about 10 seconds. Updating the monitors at runtime takes less than 50 ms, which allows monitoring to occur every second without any problem. The monitoring overhead appears very small and could be balanced on different computers.

9.3 Applicability

This section shows our proposed techniques in action in realistic situations. It describes three case-studies and the application of our techniques.

- The first case study, a Car Pooling System (CPS), was chosen to evaluate the applicability of the proposed obstacle assessment techniques as seen in Chapter 4. An existing goal and obstacle models with a large number of obstacles calls for our obstacle prioritization techniques. (See Section 9.3.1.)

- The second case study evaluates the application of the obstacle assessment techniques presented in Chapter 4 in an industrial setting. Our techniques were applied to an industrial yoke lifting system from a leading medical technology company and compared to a risk analysis previously performed by engineers of the company. (See Section 9.3.2.)

- Last, the techniques proposed in Chapter 4 to 7 are applied to a benchmark commonly used for evaluating obstacle analysis techniques [Alr12; Alr16; Fin96; Lamoo; Leto2] and other techniques such as self-adaptation [Sou12]. This case study, an Ambulance Dispatching System (ADS), was chosen for the availability of existing goal and obstacles models, domain experts, and for the experience of the author in this type of systems. (See Section 9.3.3.)

9.3.1 A car pooling system

The techniques presented in Chapter 4 were evaluated on a first case study of a car pooling system. This section focuses on evaluating the techniques in Section 4.3 for determining the likely and critical obstacles. The system is briefly described as follows.

> *The system should act as a marketplace for drivers to offer empty seats in real time and for travellers to use them under agreed conditions. A driver is matched in real time with anyone searching for a ride along a common route. Effective carpooling may critically depend on marketplace size; the system should therefore be attractive to drivers, in particular by not over-constraining them. Drivers are assumed to have a GPS-based navigation device and a PDA/iPhone-like touch screen.*

A more detailed description of the system can be found in [Dam10].

Goal and Obstacle model. The goal model for this system contains 44 goals with 2 root goals refined through 20 refinements leading to 24 leaf goals. The obstacle model contains 109 obstacles with 12 root obstacles and 69 leaf obstacles. The model contains 108 obstacle refinements. The complete model is found in Section A.2.

The root goals are Achieve [Ride Need Eventually Served] and Achieve [Ride Proposal Eventually Served]. Both have a required satisfaction rate of 95%.

Goal Achieve [Ride Need Eventually Served]
Def Passengers that want to go to a destination shall arrive at that destination within the specified time constraints.
RSR 95%

Goal Achieve [Ride Proposal Eventually Served]
Def Rides proposed by drivers shall be fulfilled within time constraints. A ride is fulfilled if the driver did not travel alone.
RSR 95%

The goal Achieve [Ride Need Eventually Served] is refined through 3 subgoals whereas Achieve [Ride Proposal Eventually Served] is refined through two subgoals; both refinements follow the *milestone-driven* refinement pattern. Figure 9.1 and Figure 9.2 show the corresponding goal model fragments. The non-assigned goals are refined in turn using a variety of refinement patterns [Lam09] until all goals are assigned to single agents. The system includes 3 agents: Driver, Passenger and Software.

The root obstacles were generated by negating the leaf goals from the goal model. For example, the goal Maintain [Ride Offer Accurate] is obstructed by the obstacle Ride Offer Not Accurate:

Goal Maintain [Ride Offer Accurate]
Def Ride offers proposed by the drivers shall always be as accurate as possible.
Obstacle Ride Offer Not Accurate
Def A ride offer encoded in the system does not accurately correspond to a ride proposal in the real world.

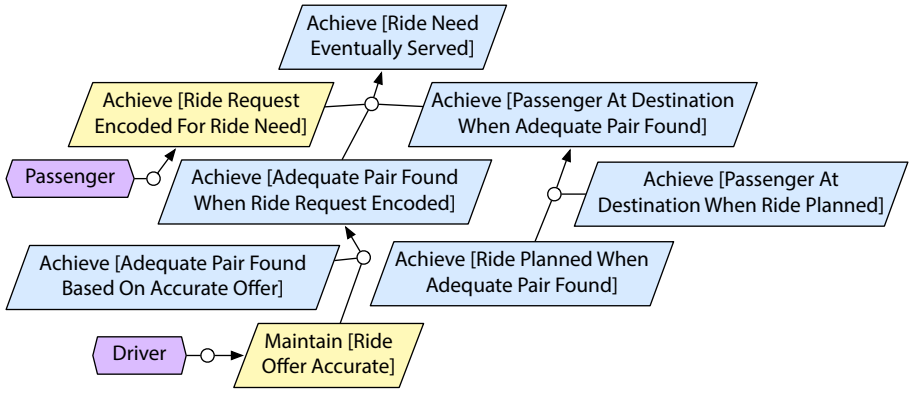


Figure 9.1 Goal model fragment for the Car Pooling System.

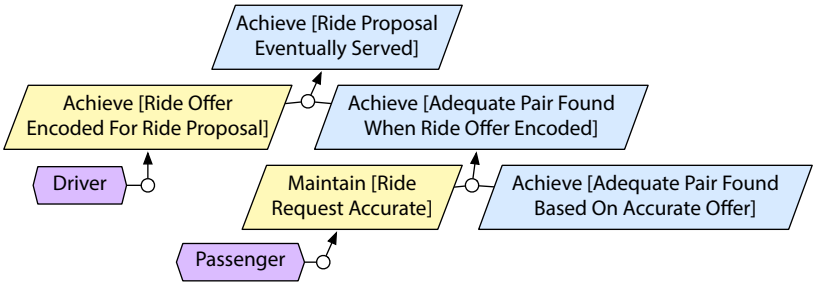


Figure 9.2 Goal model fragment for the Car Pooling System.

These obstacles are recursively refined until the satisfaction rate of the leaf obstacles can be estimated by experts using available techniques [Lam09]. Figure 9.3 shows a fragment of the obstacle model.

Obstacle assessment. The 69 leaf obstacles were then annotated with estimates of their satisfaction rate. As they are grounded on the domain, such estimates can be elicited from users' experience or from statistical data. As the system provides specific features for ride evaluation by riders, the evaluation questionnaire might be designed for non-positive evaluations to reflect the obstacle model so as to acquire relevant data.

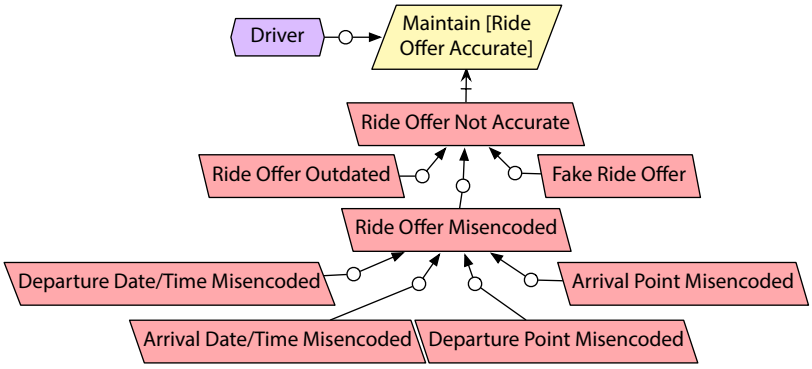


Figure 9.3 Obstacle model fragment for the Car Pooling System.

Table 9.1 provides leaf obstacle estimates based on personal carpooling experience by colleagues and us. These estimates should be refined from statistical data when available. For example, for the obstacle Drop Point Inaccessible By Car 5 behaviors at least were estimated to satisfy the obstacle condition out of 1,000 possible behaviors, the satisfaction rate of this leaf obstacle is therefore estimated to be .5%.

Global impact analysis. The propagation of the estimated satisfaction rates from the leaf obstacles to the root obstacles, then from the root obstacles to the leaf goals, and then from the leaf goals to the root goals yields the satisfaction rate of the two high-level goals. Using the BDD-based and the pattern-based procedure in Section 4.2.2, the satisfaction rate of Achieve [Ride Need Eventually Served] is found to be only only 37.28%. The satisfaction rate of the goal Achieve [Ride Proposal Eventually Served] is 81.02%.

The satisfaction rate of the high-level goal Achieve [Ride Need Eventually Served] means that the probability of serving a ride request is only about 40% if all leaf obstacles were correctly estimated and no countermeasures to likely problems are provided; there would thus be approximately 2 chances out of 5 to serve a ride request. This is far from the RSR of 95% prescribed on this main goal.

Obstacle	SatRate	Obstacle	SatRate
Fake Ride Offer	0.1%	Ride Offer Outdated	1.0%
Fake Ride Request	0.1%	Ride Request Out-dated	1.0%
Departure Date/Time Miscoded	0.9%	Departure Point Miscoded	1.5%
Arrival Date/Time Miscoded	0.9%	Arrival Point Miscoded	1.5%
No Request Matching For That Time	2.0%	No Request Matching For That Position	1.0%
No Offer Matching For That Time	2.0%	No Offer Matching For That Position	1.0%
Ride Offer Cancelled And Ride List Proposed	2.0%	Request Cancelled And Ride List Proposed	2.0%
Ride List Not Convenient	1.0%	Ride Offer Modified	1.0%
Ride Request Modified	1.0%	Ride Offer Cancelled And Selected	1.0%
Ride Request Cancelled And Selected	1.0%	Ride Offer Cancelled And Elected	1.0%
Ride Request Cancelled And Elected	1.0%	No Pickup Point Near Departure Point	0.5%
Detour Too Important For Reaching Pickup Point	1.0%	No Pickup Point Accessible To Driver	0.5%
No Pickup Point Accessible To Passenger	0.5%	No Drop Point Near Destination	1.0%
No Drop Point Accessible To Passenger	0.5%	No Drop Point Accessible To Driver	0.5%
Detour Too Important For Reaching Drop Point	0.5%	Pickup Time Incompatible With Road And Pickup Point	0.5%
Pickup Time Incompatible With Other Pickups	0.5%	Journey Longer Than Proposed Time	0.5%
Drop Time Incompatible With Road And Drop Point	0.5%	Drop Time Incompatible With Other Pickups	0.5%
Wrong Contact Information	7.0%	Failed Communication	3.0%
Ride Offer Cancelled And Instructions Sent	1.5%	Ride Request Cancelled And Instructions Sent	1.5%
Instruction In Wrong Language	0.5%	Printing Failed	1.0%
Ride Offer Cancelled Last Minute	3.0%	Ride Request Cancelled Last Minute	2.0%
Too Many Luggage	0.5%	Too Many Riders	0.5%
Not Enough Seats	0.5%	Riders Do Not Recognize	1.0%
Location Imprecise	2.0%	Passenger Late At Pickup Point	6.0%
Driver Late At Pickup Point	2.0%	Ride Request Cancelled And Driver At Pickup Point	1.0%
Driver GPS Broken Down	20.0%	Passenger Forgot	1.0%
Driver Forgot	2.0%	Passenger Get Lost	2.0%
Ride Offer Cancelled And Passenger At Pickup Point	1.0%	Driver Get Lost At PickupPoint When GPS Broken	30.0%
Instructions Not Received In Time	2.0%	Pickup Point Not Reachable To Passenger	0.5%
Pickup Point Not Reachable To Driver	0.5%	Pickup Point Confused	1.0%
Instructions Not Clear	0.5%	Other Passenger Late	0.5%
Detour On Planned Road	1.5%	Stuck In Traffic Jams	10.0%
Wrong instructions Sent	0.1%	Car Broken Down	0.5%
Driver Get Lost To Drop Point When GPS Broken Down	20.0%	Drop Point Inaccessible By Car	0.5%
Drop Point Confused	1.0%		

Table 9.1 Estimates for the identified leaf obstacles.

Given the low goal's satisfaction rates obtained by propagating leaf obstacle estimates, likely and critical obstacles need be resolved. Due to a large number of leaf obstacles (68), we need to prioritize them so that we focus our resolutions on most critical ones.

Local impact analysis. To complement our global analysis, we may therefore consider one single leaf obstacle at a time, setting the satisfaction rates of all the other leaf obstacles to 0. By up-propagation of these satisfaction rates, we obtain the violation severity for the root goal. Among the 68 single leaf obstacles, only three revealed to cause a severe violation, as Table 9.2 depicts. Our two propagation procedures both provide the same results.

Obstacle	SatRate	Violation Severity
Stuck In Traffic Jams	10%	5%
Wrong Contact Information	7%	2%
Passenger Late At Pickup Point	6%	1%

Table 9.2 Critical obstacles for the Car Pooling System.

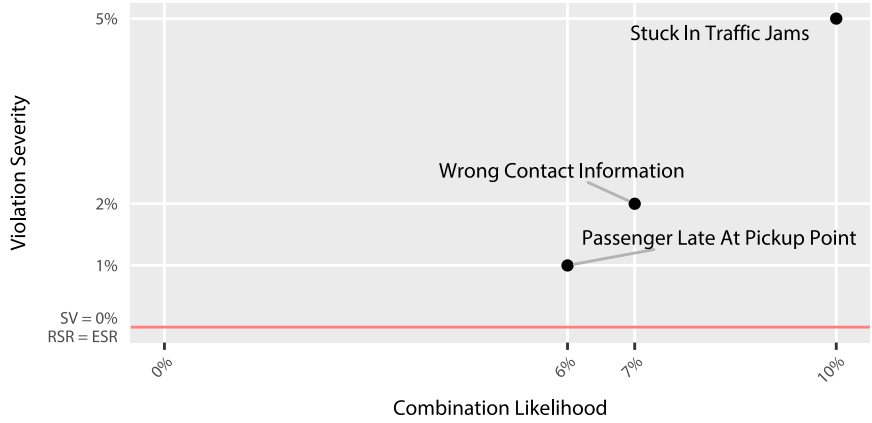


Figure 9.4 Violation Diagram for the Car Pooling System with single obstacle.

As seen in Table 9.2, the leaf obstacle Stuck In Traffic Jams, with an estimated satisfaction rate of 10%, causes a violation severity of 5% for the root goal. This means that only 90% of requested rides would then be served. The leaf obstacle

Wrong Contact Information, with an estimated satisfaction rate of 7%, causes a violation severity of 2% for the root goal. The leaf obstacle Passenger Late At Pickup Point, with an estimated satisfaction rate of 6%, yields a violation severity of 1% with respect to the root goal's prescribed RSR of 95%. Figure 9.4 shows the violation diagrams for these obstacles.

A similar table and violation diagram can be generated for the second root goal Achieve [Ride Proposal Eventually Served]. However, no obstacle alone is sufficient here for dropping the satisfaction rate below the required satisfaction rate.

After having found the single leaf obstacles that are critical, we need to turn our attention to the critical obstacle pairs. Assuming the preceding critical singletons are selected for resolution, the critical pairs should not redundantly include them. Using the brute-force approach discussed in Section 4.3, we found 206 pairs of leaf obstacles that are critical with respect to the root goal out of the 2415 generated pairs in less than a second. Among those 206 pairs, 2 do not include any of the critical singletons in Table 9.2. The most critical pairs among these are shown in Table 9.3. Together with the critical singletons, they are the ones having higher priority for resolution. Once resolved, if the required satisfaction rate of the high-level goals is not yet reached, triple combinations excluding the preceding critical singletons and pairs could reveal more critical obstacle combinations.

Obstacles	SatRate	Violation Severity
Ride Offer Cancelled Last Minute, Failed Communication	0.09%	0.91%
Driver Get Lost To Pickup Point When GPS Broken, Driver GPS Broken Down	6.00%	1.00%

Table 9.3 Critical pairs of obstacle for the Car Pooling System.

Note that the ranking of leaf obstacles by their satisfaction rate is not the same as their ranking by the resulting violation severity for the root goal. The obstacle pair {Ride Offer Cancelled Last Minute, Failed Communication} has a lower satisfaction rate compared to the pair {Driver Get Lost To Pickup Point When GPS Broken, Driver GPS Broken Down} even though their violation severities are close to each other.

Discussion. Our global impact analysis revealed that the original model is not compliant with our probabilistic requirement of serving 95% of encoded ride requests. Prioritization was required to handle the large number of leaf obstacles obtained. Our local impact analysis revealed critical single obstacles and critical obstacle pairs to be resolved first in the next phase of risk control.

Some of the leaf obstacles appeared more critical than others, even if they have a small satisfaction rate, especially combined with other obstacles. For example, the obstacle Ride Request Cancelled And Ride List Proposed, with a satisfaction rate of 0.02, was seen to potentially obstruct the leaf goals Achieve [Suggestions Selected When Proposed by Passenger] and Achieve [Suggestions Selected When Proposed by Driver]. Even if its estimated satisfaction rate is low, the obstacle may be critical.

Leaf obstacles with lower satisfaction rate may thus be more critical than ones with a higher satisfaction rate. When they obstruct more goals, an increase in their satisfaction rate may have a major impact on high-level goals' satisfaction rate.

Critical combinations with one or two leaf obstacles appeared to include most of the critical obstacles. Combinations with more obstacles were often supersets of those critical combinations. The resolution of critical singletons and pairs is therefore expected to substantially reduce the number of critical combinations of a larger size.

In short, the large number of obstacles made it quite difficult to identify most critical obstacles to be considered first for resolution. The prioritized list of obstacles produced by our technique helped significantly in that direction.

Most likely and critical obstacles should be resolved next; this case study focused on evaluation the techniques for *obstacle assessment*. The techniques for *obstacle control* are evaluated in Section 9.3.3 on a different case study.

9.3.2 The IBA yoke lifting system

We applied the techniques from Chapter 4 to a second case study. This section focuses on evaluating the applicability of the techniques for *obstacle assessment* in an industrial setting.

This second case study is an industrial system in the medical domain. IBA is a medical technology company based in Louvain-la-Neuve (BE), founded in 1986, and active in the field of proton therapy. IBA is currently the world leader in proton therapy solutions for cancer treatment [Lin11]. Among their products, cyclotrons accelerate particles using a high-frequency alternating voltage applied between electrodes inside a vacuum chamber. A two-part yoke encloses the necessary equipment; *yoke* refers to the pieces sustaining the distance between the electrodes, as seen in grey in Figure 9.5. The upper part of the yoke can be lifted to access the inside of the cyclotron for operation and maintenance purposes. Given the weight (> 110 tons) and size of the upper yoke, the lifting system is critical for operators safety. Figure 9.5 shows a IBA Cyclone 230® cyclotron open (left) and closed (right), showing the yoke lifting system in action.

Figure 9.5 IBA Cyclone 230® Cyclotron.

Our risk analysis focussed on the human safety aspects for the IBA Cyclotron Service Engineers, named *operators* in the following, on the IBA C230 Yoke Lifting System (C230-YLS).

The goal and obstacle models were built and analyzed in collaboration with G. Gérard (IBA R&D Requirement Engineer) and P. Cailliau (IBA R&D C230 System Owner).

The purpose of the analysis was two-fold:

(a) Identifying potential failures, assessing their criticality and defining mitigations when required;

(b) Evaluating the proposed approach for *obstacle assessment* on this kind of risk analysis.

A first risk analysis (FMECA) was available before building the goal/obstacle models.

Building the goal and obstacle models. A generic goal model was built for human safety requirements based on an available document entitled *Occupational Injury and Illness Classification Manual* [Sta92]. Our model contains 220 goals with 47 refinements and 173 leaf goals. The model provides a set of detailed safety requirements. Most of the goals in the model are *Avoid* goals such as Avoid [Contact with hot objects or substances] or Avoid [Fall from moving vehicle or mobile equipment]. These goals were directly mapped to categories of injuries and illness from the available classification manual; this manual provides definitions and appropriate examples. Figure 9.6 shows the refinement for the root goal Avoid [Occupational Injury and Illness].

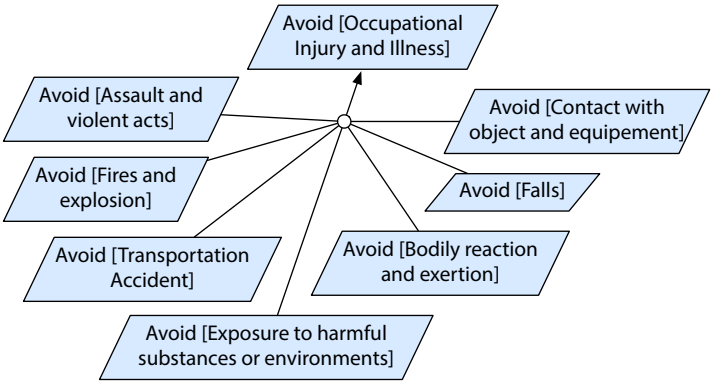


Figure 9.6 The refinement of Avoid [Occupational Injury and Illness].

The goal model was then specialized to identify the specific human safety requirements for the C230-YLS. Thanks to the refinement structure, large parts of the generic model were pruned. For instance, as no transport is involved in the system, the goal Avoid [Transportation Accident] in Figure 9.6 could be removed from the model together with its descendants. The high-level goal Avoid [Assault and violent acts] was considered to be out of scope; this and its descendants were removed too. Such pruning was applied recursively on the refinement structure,

to eliminate goals that are either not relevant or out of scope. The resulting specific goal model contains 22 goals refined through 9 goal refinements into 13 leaf goals. These are the specific human safety goals to be enforced by the C230-YLS.

Obstacle analysis was anchored on those leaf goals. From the negation of these leaf goals, obstacles unique to the C230-YLS were produced based on prior experience, existing risk analysis and incident reports. Figure 9.7 shows one of the obstacle trees. The obstacle model contains 43 obstacles and 28 obstacle refinements. Thanks to a systematic obstacle identification process, we identified risks that were absent from the prior risk analysis performed by IBA engineers. The latter used Failure Modes, Effects, and Criticality Analysis (FMECA) [Levo1].

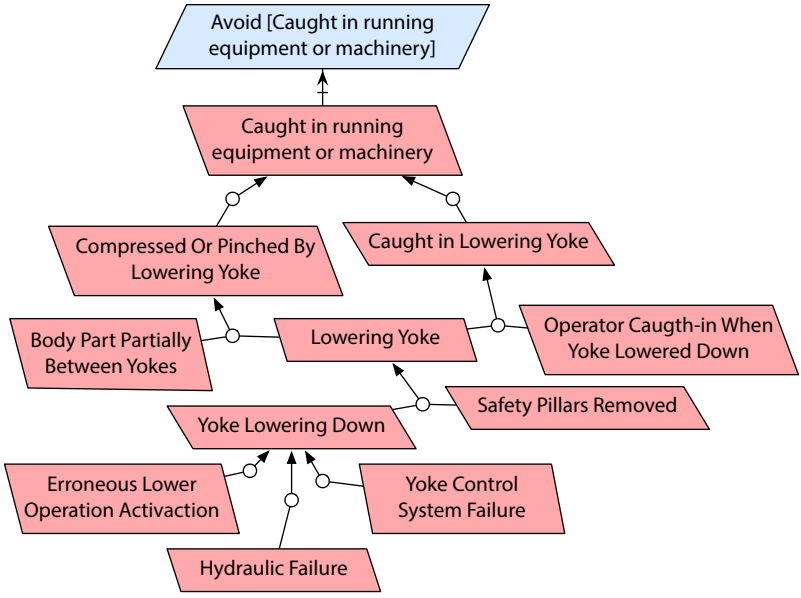


Figure 9.7 Obstacle tree fragment for the goal Avoid [Caught in running equipment or machinery].

Obstacle assessment. To assess the risks, FMECA tables were generated from the obstacle refinement trees (See Section 10.1 for a detailed comparison between our techniques and FMECA.) Each row in a FMECA table corresponds to a failure mode. A FMECA table contains the following columns:

- **ID:** a unique identifier.
- **Failure Mode Description:** describes the unsatisfied requirement.
- **Cause:** describes why the requirement is not satisfied.
- **Effect:** describes the consequences of not satisfying of the requirement.
- **Risk:** assessment of the risk likelihood and criticality.
- **Mitigated by:** describes the mitigations for the risk.
- **Risk after:** assessment of the risk likelihood and criticality after the mitigations.

The FMECA-obstacle mapping is as follows. The *Failure Mode Description* corresponds to the obstructed leaf goal; *cause* was generated from the obstacle AND-refinements containing leaf goals, *effect* was initially generated from the ancestors of the obstructed leaf goal but was later reformulated manually to match IBA practices.

We generated 5 tables corresponding to the level-1 goals refining the root goals in Figure 9.6, namely, Avoid [Bodily reaction], Avoid [Exposure to harmful substances or environments], Avoid [Falls], Avoid [Contact with object and equipement], Avoid [Fires and explosion].

Assessing the likelihood and criticality of obstacles. Risk assessment was performed by IBA engineers based on the FMECA tables. This assessment was introduced back into the goal/obstacle model afterwards. Risk likelihoods were assessed using a standard semi-quantitative scale ranging from A to F [Ayy14]. To enable our techniques, we converted this scale into quantitative values using reference values provided in [Ayy14]. Table 9.4 shows the correspondence between the two scales. The required satisfaction rate (RSR) for the top-level goal was defined to 99.99%. Interestingly, during risk assessment process engineers split some of the leaf obstacles into subobstacles to support more fine-grained evaluation; the changes were reflected in our models.

Category	Description	Numeric value
A	Frequent	.1
B	Probable	.01
C	Occasional	.001
D	Remote	.0001
E	Improbable	.00001
F	Incredible	.000001

Table 9.4 Semi-quantitative scale for risk assessment at IBA.

Global and local impact analysis. Our global analysis revealed that the satisfaction rate of Avoid [Injury and Illness Caused By YLS] is 91.46%, using the BDD-based propagation procedure in Section 4.2.1. This does not meet the required satisfaction rate. Applying our techniques for local analysis revealed 12 obstacles that were sufficient alone to cause the satisfaction rate of the high-level goal to fall below its required threshold. Two groups of obstacles could be identified: obstacles with a .01 satisfaction rate and obstacles with a .001 satisfaction rates. All obstacles identified as likely and critical by our techniques were considered as intolerable using the IBA Risk Criticality Matrix. This matrix classifies risks in three categories: *Intolerable* (I), *As low as reasonably practicable* (II) and *Broadly Acceptable* (III).

Controlling obstacles. Based on the generated FMECA tables, IBA engineers identified 32 mitigations; these were introduced as countermeasures goals. The techniques presented in Chapter 5 could however not be applied to select most appropriate countermeasures. The current practice at IBA intertwines assessment and control; it assesses the *risk after* as the severity of the risk once all mitigations are integrated. The mitigations are elicited as long as the *risk after* does not meet the target severity. Our approach separates the countermeasures identification from assessment and applies assessment to each individual countermeasure.

Discussion. By separating countermeasure identification from assessment, our approach may improve the cost-effectiveness of the selected mitigations. Such separation is discussed in the context of conflict resolutions in [Eas94]. It enables the selection of most appropriate countermeasures at a system level but requires

some extra assessment from domain experts. It appeared that *(a)* some of the selected mitigations might be redundant and *(b)* that more global mitigations resolving multiple risks could be favored over more local mitigations.

Overall the goal/obstacle approach appeared to be helpful as it provides a complementary top-down approach to the current bottom-up approach in use at IBA. The goal/obstacle models provide a more global view of the system under scrutiny compared to the more detailed approach proposed by FMECA. Risks not accounted for in the risk analysis previously performed were found during the goal/obstacle analysis together with extra mitigations. How top-down and bottom-up risk analysis approaches could be integrated remains, to us, an open issue.

In addition, the anchoring of probabilistic obstacles on real-world phenomena was felt helpful for estimating obstacle satisfaction rates, making the mapping to available statistics easier. IBA currently works on improving the quantitative aspects of their risk analysis to enable the reuse of such available statistics.

Our techniques revealed that some obstacles were considered as likely and critical by our techniques while being considered as '*As low as reasonably practicable* (II)'; those were risks causing minor injuries. Differentiating between minor and major injuries turned to be challenging. For example a minor injury, while moving a pillar of 20kg and a major injury resulting from being crushed by the 110 tons yoke both result in the non-satisfaction of the high-level goal Avoid [Injury and Illness Caused By YLS]. There is, however, a major distinction, not accounted for in our approach, between the two.

Our techniques for obstacle control, coping with knowledge uncertainty and runtime adaptation are evaluated on the next case study.

9.3.3 The Brussels ambulance dispatch system

The techniques from Chapter 4-7 were applied to a benchmark commonly used for evaluating obstacle analysis techniques [Alr12; Alr16; Fin96; Lam00; Let02] and for evaluating other software engineering techniques, in particular self-adaptation [Sou12]. The aim is evaluate the applicability of our *obstacle assessment* and *control* techniques, with or without knowledge uncertainty, and both at RE time and at runtime, to a system of significant size.

The model used for the evaluation was inspired by the one used for evaluating obstacle analysis techniques in [Lamoo; Leto2]. In addition, the model was adapted and enriched based on the author's experience as a volunteer Emergency Medical Technician (EMT) in this type of system. When appropriate, the system was tailored to match the regulations and practice applicable in the Brussels area (Belgium) [Pub13].

The following briefly describes the Ambulance Dispatch System (ADS).

> *A "112" call connects the caller to a dispatch operator who inputs information about the incident into the dispatch software. The system then has to allocate and mobilize an appropriate ambulance, and transmit all relevant details to the selected vehicle. The system has also to track the actual availability and position of the ambulances.*

For a more detailed system description, see [Leto2].

Goal and obstacle model. The goal model contains 56 goals with 3 root goals. There are 27 refinements leading to 34 leaf goals. The obstacle model includes 84 obstacles. The root obstacles obstruct 17 leaf goals. There are 69 refinements for these root obstacles, leading to 54 leaf obstacles. Section A.1 details the complete specification for the goals and obstacles.

The primary root goal is Achieve [Incident Resolved], with a 95% required satisfaction rate.

Goal Achieve [Incident Resolved]
Def All incidents shall be resolved. An incident is resolved if the victim is treated on the incident scene or transported to an hospital.
RSR 95%

A *milestone-driven* refinement decomposes the root goal Achieve [Incident Resolved] into three subgoals: Achieve [Incident Reported], Achieve [Ambulance On Scene When Incident Reported], and Achieve [Incident Resolved When Ambulance On Scene]. The Public agent is responsible for the goal Achieve [Incident Reported]; the Ambulance staff agent in collaboration with the Automated Dispatch Software is responsible for the goal Achieve [Incident Resolved When Ambulance On Scene];

the goal is therefore refined until assigned to these agents, not seen in Figure 9.8. Three subgoals refine the goal Achieve [Ambulance On Scene When Incident Reported] as Figure 9.8 shows. The latter goals are recursively refined until assigned to single agents.

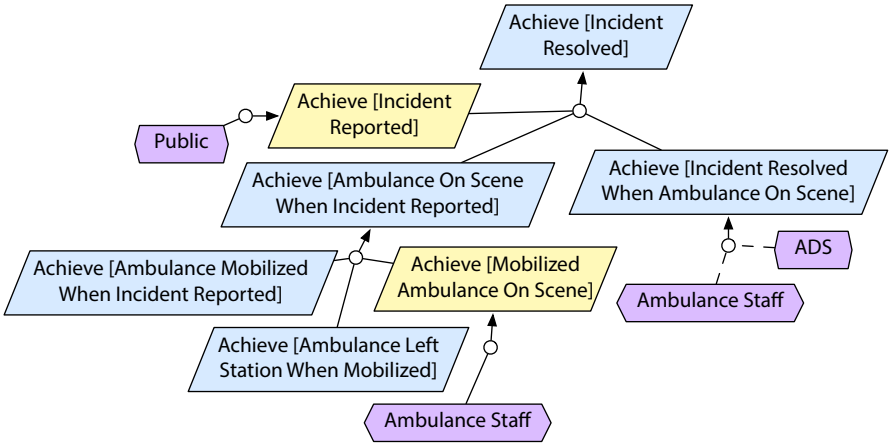


Figure 9.8 Goal model fragment for the Ambulance Dispatch System.

The two other root goals are Maintain [Accurate Ambulance Status Known] and Maintain [Accurate Ambulance Location Known]:

Goal Maintain [Accurate Ambulance Status Known]
Def The status of the ambulance shall be accurately known to the dispatching software. A status is accurate if the known status corresponds to the status of the ambulance in the real-world. The status shall be known to the dispatching software within 3 minutes when encoded by the staff.
RSR 90%

Goal Maintain [Accurate Ambulance Location Known]
Def The accurate location of the ambulance is known by the dispatching software. A location is accurate if the position of the ambulance in the real-world and the dispatching position do not differ by more than 250 meters.
RSR 90%

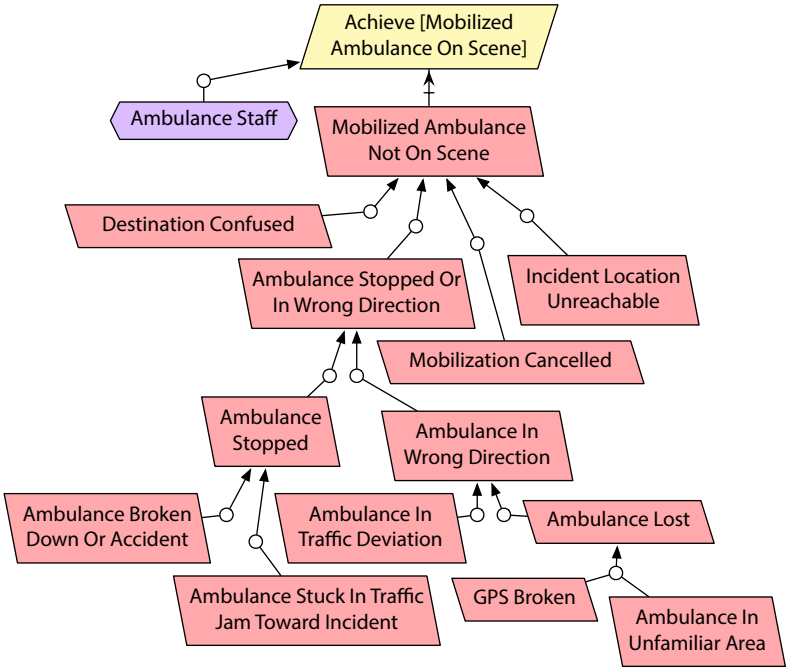


Figure 9.9 Obstacle model fragment for the Ambulance Dispatch System.

Negating the leaf goals generates obstacles that are later refined. For example, the leaf goal Achieve [Mobilized Ambulance On Scene] generates the root obstacle Mobilized Ambulance Not On Scene. This obstacle is refined until experts can estimate the satisfaction rates for the leaf obstacles. Figure 9.9 shows an obstacle tree for the goal Achieve [Mobilized Ambulance On Scene].

The available resolution tactics [Lamoo] produce countermeasure goal candidates. For example, here are five countermeasure goals for two leaf obstacles:

Obstacle Mobilization Taken By Other Ambulance
Resolvedby Achieve [Mobilization By Other Ambulance Known]
Resolvedby Avoid [Mobilization Without Order]

Obstacle Displayed Mobilization Order Ignored
Resolvedby Achieve [Alarm When Mobilization Order Displayed]
Resolvedby Achieve [Failed Mobilization Recovered]
Resolvedby Achieve [Late Mobilization When Crew Not Responsive]

Assessing likelihood and criticality of obstacles

In order to evaluate the techniques presented in Chapter 4, the satisfaction rate of 34 leaf obstacles was estimated based upon the author's experience as a volunteer Emergency Medical Technician (EMT) exposed to the Brussels ambulance system.

Our global impact analysis using the techniques in Section 4.3 revealed that the high-level goal Achieve [Incident Resolved] in Figure 9.8 has a computed satisfaction rate of 6.20%; this is far from the required satisfaction rate of 95% to be achieved by the system-to-be while agreeing with the observed rate of ideal ambulance interventions. The satisfaction rates for higher-level goals were here obtained using the BDD-based propagation procedure described in Section 4.2.1.

Local impact analysis using the techniques in Section 4.3 revealed that 17 obstacles are sufficient alone to prevent the required satisfaction rate to be achieved. Table 9.5 presents the likely and critical obstacles to be resolved with highest priority. There are 142 pairs of obstacle, not containing one of the 17 obstacles previously identified, that cause the computed satisfaction rate to drop below the required satisfaction rate.

Controlling likely and critical obstacles

The number of obstacles and countermeasure goals called for countermeasure integration according to the techniques presented in Chapter 5. As a result, the countermeasure goals appeared to focus on a small number of important goals; e.g., the goal Achieve [Incident Resolved When Ambulance On Scene] has 8 *Except* constructs. The overall integration produced 28 *Except* constructs distributed among 8 goals only.

Obstacle	Probability	Violation Severity
Out Of Paper	40.00%	35.00%
Allocated Ambulance Not At Station	33.33%	28.33%
Patient Not Transportable	13.33%	8.33%
Patient Cannot Reach Ambulance	13.33%	8.33%
Special Unit Required	13.33%	8.33%
Crew Not In Ambulance	13.33%	8.33%
Paper Jam	10.00%	5.00%
GPS Black Spot	10.00%	5.00%
Ambulance In Traffic Deviation	10.00%	5.00%
Ambulance Stuck In Traffic Jam Toward Incident	7.00%	2.00%
Printer Off	6.67%	1.67%
Destination Confused	6.67%	1.67%
Hazardous Environment	6.67%	1.67%
Unreachable Patient	6.67%	1.67%
Ressource Out Of Order	6.67%	1.67%
Insufficient Capacity	6.67%	1.67%
Crew Distracted	6.67%	1.67%

Table 9.5 Critical obstacles for the Ambulance Dispatch System.

The techniques presented in Chapter 5 helped significantly for the following reasons.

Model simplification by separation of concerns. The goals referring to normal situations are systematically distinguished from those handling obstacle occurrences. Emerging assumptions were incrementally down-propagated to the obstructed descendants of the corresponding anchor goals. This produced 62 *Provided* annotations distributed over 13 goals. Without these annotations, details related to exceptional cases would have cluttered the formal specification of those goals.

For example, the goal Achieve [Allocated Ambulance Mobilized When Mobilization Order Printed And Phone Contact] is defined as follows after integration in the model:

Goal Achieve [Allocated Ambulance Mobilized
When Mobilization Order Printed And Phone Contact]
ProvidedNot Mobilization Taken By Other Ambulance
ProvidedNot Allocated Ambulance Not At Station
ProvidedNot Allocated Ambulance Not Available
ProvidedNot Printed Mobilization Order Ignored

The full equivalent specification of this goal without *ProvidedNot* annotations would have completely hidden the ideal case. It would then appear hard to distinguish the part of the goal antecedent related to the ideal case from the parts related to exceptional cases. An example is provided here.

The *Detach-Except* operator was applied to the *case-driven* refinement of the goal Achieve [Allocated Ambulance Mobilization]. Allocating an ambulance when not at its station was estimated fairly rare—5% of cases according to typical figures in the domain. The parent goal of these two goals was therefore modified accordingly:

Goal Achieve [Allocated Ambulance Mobilization]
Except AllocatedAmbulanceNotAtStation
then Achieve [Allocated Ambulance Mobilization On Road]

Such refactoring reduces model complexity by hiding the part of the model handling the mobilization of an ambulance when on the road. The resulting ideal goal model contains fewer refinements and fewer goals, making it easier to understand and separate typical behaviors from exceptional ones.

Compositionality. Without the techniques in Section 5.4.3, the integration of so many exceptions for only 8 goals would have resulted in large, complicated refinements with a combinatorial blow-up of exceptional cases. To illustrate this important point, consider the goal Achieve [Ambulance Mobilized When Allocated]. The original, ideal specification is:

$$\forall amb : Ambulance, inc : Incident \cdot \\ Allocated(amb, inc) \Rightarrow \Diamond_{\leq 1min} Mobilized(amb, inc)$$

After obstacle analysis, this goal is ensured through 5 countermeasure goals (see Figure 9.10). A brute-force integration of only three of those (in red, see Figure 9.10) would have resulted in the following formal specification for the final version of the goal Achieve [Ambulance Mobilized When Allocated]:

$$
\begin{aligned}
&\forall amb : Ambulance, inc : Incident \cdot \\
&\quad Allocated(amb, inc) \\
&\qquad \Rightarrow \Diamond_{\leq 1min} Mobilized(amb, inc) \\
&\qquad\quad \vee [\Box_{>3min} \neg AmbAvailable(amb, inc) \\
&\qquad\qquad\quad \rightarrow \Diamond_{\leq 6min} \exists amb' : Ambulance \cdot Mobilized(amb', inc)] \\
&\qquad\quad \vee [\Box_{>3min} DisplayedMobilizationIgnored(amb, inc) \\
&\qquad\qquad\quad \rightarrow \Diamond_{\leq 6min} \exists amb' : Ambulance \cdot Mobilized(amb', inc)] \\
&\qquad\quad \vee [\Box_{>3min} PrintedMobilizationIgnored(amb, inc) \\
&\qquad\qquad\quad \rightarrow \Diamond_{\leq 6min} \exists amb' : Ambulance \cdot Mobilized(amb', inc)]
\end{aligned}
$$

In addition to this complex specification, the goal refinement structure would have been heavily modified as follows:

Achieve [Ambulance Mobilized When Allocated]
- ← Achieve [Other Ambulance Mobilized When Allocated Ambulance Not Available]
- ← Achieve [Ambulance Mobilized When Allocated And Available]
  - ← Achieve [Late Mobilization When Crew Not Responsive]
  - ← Achieve[Ambulance Mobilized When Allocated And Available And Displayed Mobilization Order Not Ignored]
    - ← Achieve [Other Ambulance Mobilized After Timeout]
    - ← Achieve [Ambulance Mobilized When Allocated And Available And Displayed Mobilization Order Not Ignored And Printed Mobilization Order Not Ignored]

With such brute-force integration, each countermeasure goal must be refined by taking other countermeasures into account. This lead to a combinatorial blow-up of cases. Thanks to our technique, the original specification of this goal and its refinement structure are preserved. The *Except* and *Provided* constructs encapsulate the modifications for a more robust system.

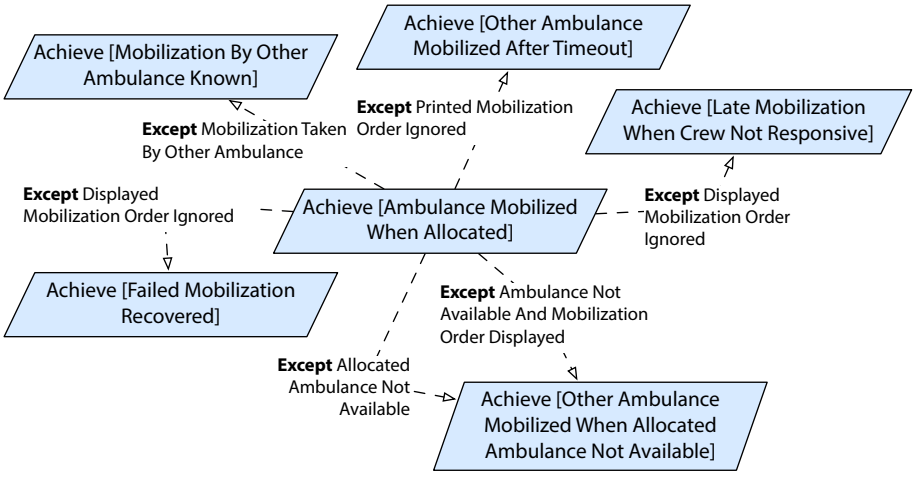


Figure 9.10 Exception Diagram for Achieve [Ambulance Mobilized When Allocated].

No premature decision and freedom of choice. The exceptions separate the specification and documentation of unusual behaviors from the usual ones. This allowed delaying the decision of how and when the handling of exceptional cases should occur.

Other benefits. The *Replaces* annotation was felt useful for documenting the replacing countermeasure goals—e.g., Achieve [Ambulance On Scene Or Cancelled When Incident Reported] replacing Achieve [Ambulance On Scene When Incident Reported] to resolve the obstacle Mobilization Cancelled. Without this annotation, we would have lost the previous version of the goal.

Exception diagrams significantly helped understand the model where all countermeasures are integrated; they document exceptions one single goal at a time, as shown in Figure 9.10. For the ADS, 14 exception diagrams document exceptional cases and countermeasure goals.

Handling uncertainty in satisfaction rates

To evaluate the techniques in Section 6.2 for obstacle assessment in presence of knowledge uncertainty, we asked five experienced EMTs involved in the system to estimate missing or unavailable data. The satisfaction rates for the leaf

obstacles in the Brussels area are not publicly available although they are partially recorded. Table 9.6 outlines some of the collected data. The experts provided estimates based on a custom number of interventions; all estimates were then converted into percentages (which explains decimal values in Table 9.6).

		Expert 1	Expert 2	Expert 3	Expert 4	Expert 5
Ambulance In Unfamiliar Area	Min	6.67%	15.%	15.%	20.%	20.%
	Mode	13.33%	30.%	18.%	20.%	24.%
	Max	20.%	50.%	25.%	30.%	30.%
Ambulance In Traffic Deviation	Min	6.67%	15.%	30.%	40.%	20.%
	Mode	20.%	50.%	50.%	50.%	30.%
	Max	33.33%	75.%	65.%	70.%	40.%
Printer Off	Min	0.%	0.%	0.%	1.%	0.%
	Mode	6.67%	0.%	0.%	1.%	0.%
	Max	6.67%	0.%	0.%	1.%	0.%
MDT Turned Off	Min	53.33%	0.%	0.%	1.%	0.%
	Mode	66.67%	0.%	0.%	1.%	0.%
	Max	80.%	0.%	0.%	1.%	0.%
AVLS Out of Service	Min	0.%	0.%	0.%	0.%	0.%
	Mode	0.%	0.%	0.%	0.%	0.%
	Max	0.%	0.%	0.%	0.%	0.%
Forget To Encode Leaving Status	Min	13.33%	0.%	6.%	20.%	10.%
	Mode	20.%	0.%	10.%	25.%	12.%
	Max	26.67%	10.%	18.%	40.%	16.%
Other Status Than Leaving Encoded	Min	0.%	0.%	0.%	10.%	0.%
	Mode	0.%	0.%	0.%	10.%	0.%
	Max	6.67%	5.%	0.%	10.%	0.%
Patient Not Transportable	Min	0.%	0.%	10.%	5.%	2.%
	Mode	13.33%	0.%	15.%	5.%	2.%
	Max	20.%	0.%	20.%	5.%	4.%

Table 9.6 Expert estimates for the Ambulance Dispatch System.

As Table 9.6 shows, experts may disagree strongly on estimated satisfaction rates such as the one for MDT Turned Off. Surprisingly, two leaf obstacles, Mobilization Taken By Other Ambulance and AVLS Out of Service, were estimated by all experts to have a satisfaction rate of 0%. This calls for further evaluation before removing the obstacles from the model.

For calibration, statistical data were obtained about the following obstacles: Allocated Ambulance Not At Station (50%), Mobilization Cancelled (13%), and MDT Turned Off (33.3%). These leaf obstacles were used as calibration variables.

The results produced by the techniques presented in Chapter 6 proved helpful in the following respects.

Managing knowledge uncertainty. Based on the calibration and on collected data, the violation uncertainty obtained for the top goal Achieve [Incident Resolved] was 100%, with an uncertainty spread of 0.9258. Figure 9.11 shows the satisfaction uncertainty for this goal. This might seem low; the reason is that the model only captures the ideal case without taking any countermeasure to obstacles into account. The rate for ideal ambulance intervention was experienced to be roughly similar to the curve obtained with our technique.

Figure 9.11 Satisfaction Uncerainty for Achieve [Incident Resolved].

Violation diagrams helped identify most likely and critical obstacles together with obstacles requiring further elicitation. The gray dots in Figure 9.12 show the violation uncertainty and uncertainty spread for the top goal, taking all experts into account.

Seven obstacles were identified to cause the top goal not to meet its RSR with more than 90% of certainty, namely, Ambulance In Traffic Deviation, Allocated Ambulance Not At Station, GPS Black Spot, Forget To Encode Leaving Status, Forget To Encode AvailableRadio Status, Forget To Encode AvailableStation Status, Wrong Info About Patient. Adequate countermeasures to these critical obstacles should therefore be elaborated and integrated.

Among all obstacles, 30 have an uncertainty spread higher than 0.10. These obstacles might, therefore, be further refined; or more experts should be asked to reduce the uncertainty margins further. The uncertainty spread for the 24

other critical obstacles was low. This indicates that experts roughly agreed on their satisfaction rate. Over the 54 leaf obstacles, 20 are, with certainty, not causing the satisfaction rate of the top goal to fall below its RSR.

Capturing uncertainty about risk estimates. Emergency Medical Technicians were asked to estimate a lower bound, the most probable value, and an upper bound for the satisfaction rate of all leaf obstacles. To mitigate the common difficulty of estimating strict and accurate lower and upper bounds [Voso8], the ones collected were used as 10th- and 90th- quantiles, with a 10% overshoot being integrated. Eliciting probability distributions would have been impossible in practice as the required statistical background is far too important for the considered personnel.

Integrating estimates from multiple experts. The uncertainty spread for the goal Achieve [Incident Resolved] taking only *Expert 1* into account is 0.9487 which is very high. The uncertainty spread caused by each obstacle is then high on average, as the orange triangles in Figure 9.12 shows. Using more than one expert helped us reduce the general uncertainty spread. However, in some cases, it increased the uncertainty spread as experts may disagree.

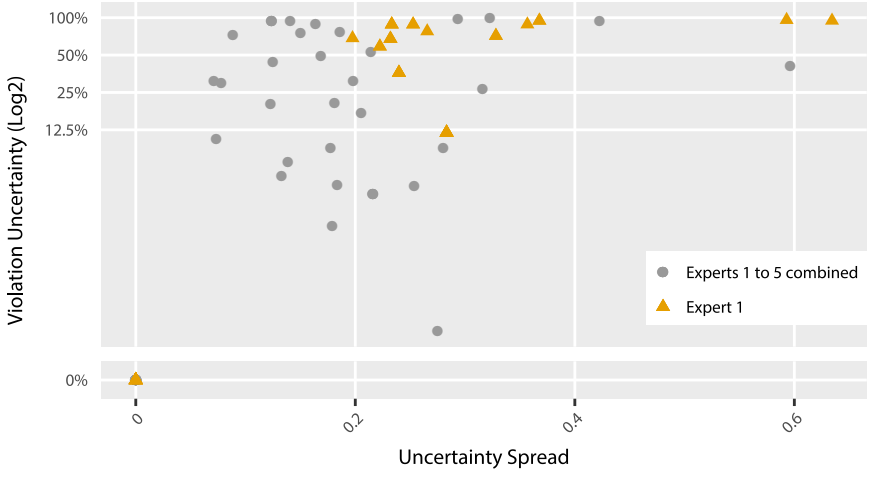


Figure 9.12 Violation Diagram for the Ambulance Dispatch System.

Handling obstacles at system runtime

To evaluate the risk-driven runtime adaptation techniques in Section 7.4, we simulated the operation of the ambulance dispatching system in Brussels, considering 15 ambulances and 10 simultaneous incidents. Over a 4-hour simulation, a total of 100 incidents was reached. In comparison, the real Brussels ambulance dispatching system handles about 25 ambulances and 300 incidents per day. The simulation was performed on a MacBook Pro 3Ghz equipped with 16Gb of RAM.

Dispatching software implementation and simulator. The ambulance dispatch software, developed in C#, implements the identified software requirements. The software is freely available to replicate our experiments [Cai17c]. Among its components, an alternative *OnRoadAllocator*, which first allocates ambulances that are on the road, can replace the default ambulance allocator. Extra components include *TrafficJamAllocator* or *StatusDetector*. The former re-allocates incidents for which the allocated ambulance is in a traffic jam; the latter automates the status reporting. Figure 9.13 shows a screenshot of the web application to visualize incidents and ambulances; dots represent ambulances and incidents together with their status.

The environment simulator simulates the behavior of the ambulance staff, the dispatchers, and other environment agents. For example, the simulator 'presses' buttons on mobile data terminals (MDT), 'drives' the ambulance to the incident location, and so forth. The simulator also generates incidents; the user can manually encode incidents using the ADS Console tool. The simulator runs in a separate process (and might run on a different machine) and communicates with the dispatching software through RabbitMQ queues [Vid12].

Figure 9.14 shows the global architecture with communication queues. The *ADS Monitoring Client* takes snapshots of the software system and generates the truth value for the corresponding predicates. Utils.Monitor, presented in Chapter 8, consumes these predicates and feeds a queue with the activation/deactivation procedures that need to be applied to guarantee the RSR of the high-level goal.

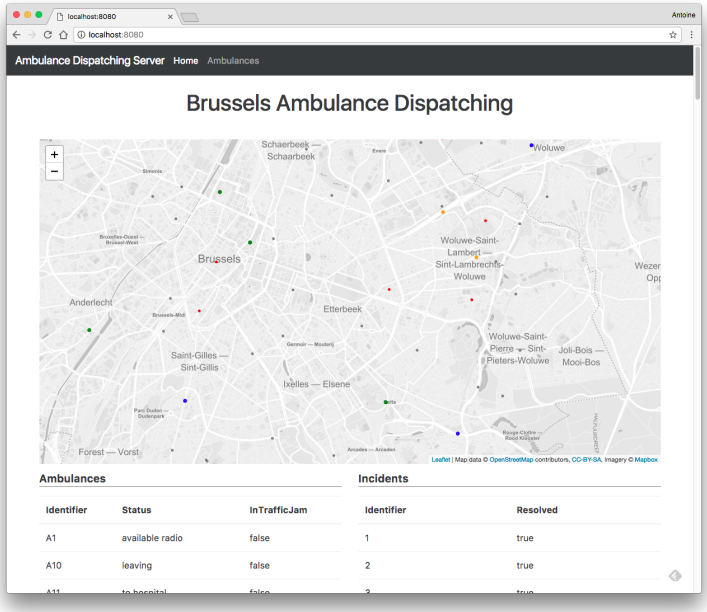


Figure 9.13 ADS Visualizaton Web Application.

Runtime monitoring and adaptation. At RE time, our monitoring tool built 109 monitors for all probabilistic leaf obstacles. At runtime, every second, 351 predicates were monitored, the monitors were updated, and the satisfaction rate of the high-level goals Achieve [Ambulance On Scene When Incident Reported] and Avoid [Ambulance On Road Mobilized] was computed. Every 5 minutes, the tool compared the monitored satisfaction rate of these goals with their respective RSR. When required, it computed most appropriate countermeasures. Later on, the tool called the corresponding activation/deactivation procedure for reconfiguring the ADS. The optimization process took about 2 minutes. During simulation, however, one elapsed second corresponds to ten seconds in reality. This explains why Figure 9.15 exhibits a substantial delay between the time at which adaptation is found necessary and the time at which it is deployed.

Simulation scenarios. We ran three simulations over 4 hours to cover the following three scenarios.

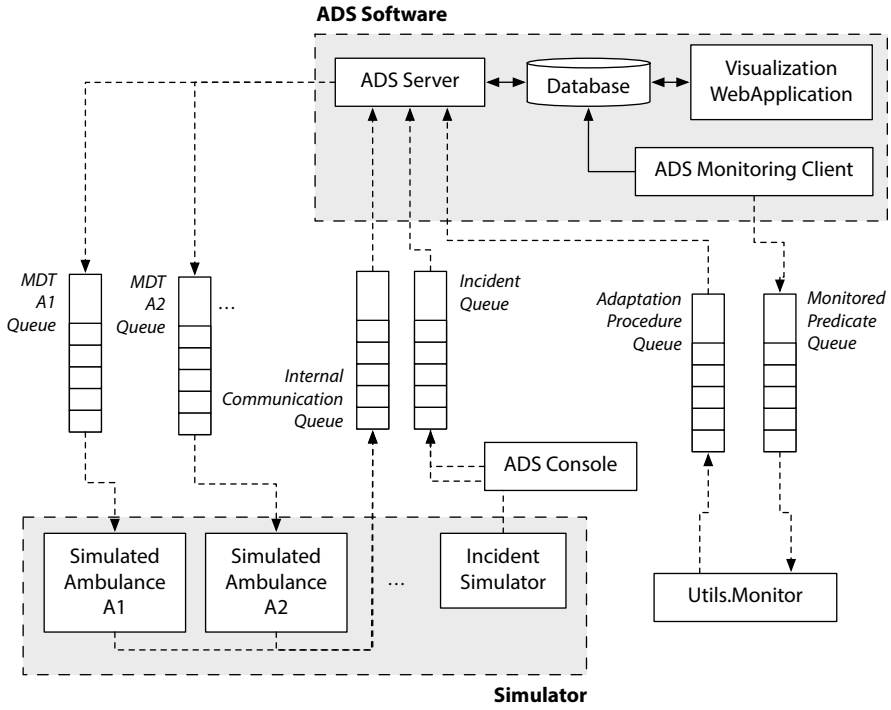
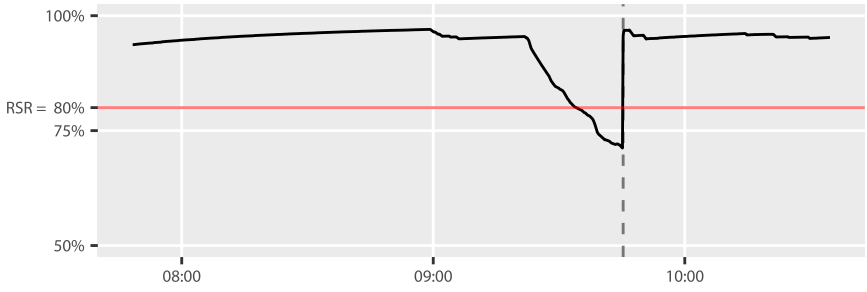
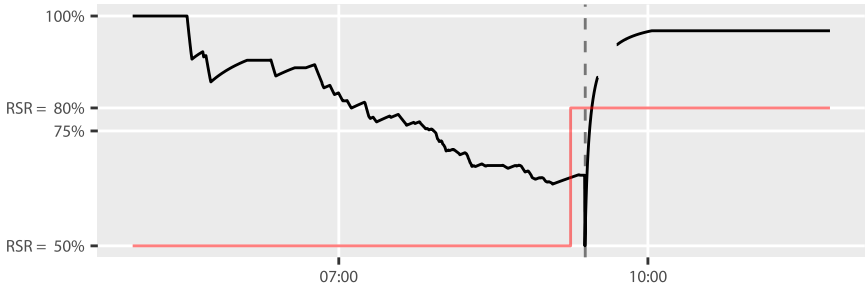


Figure 9.14 ADS Architecture.

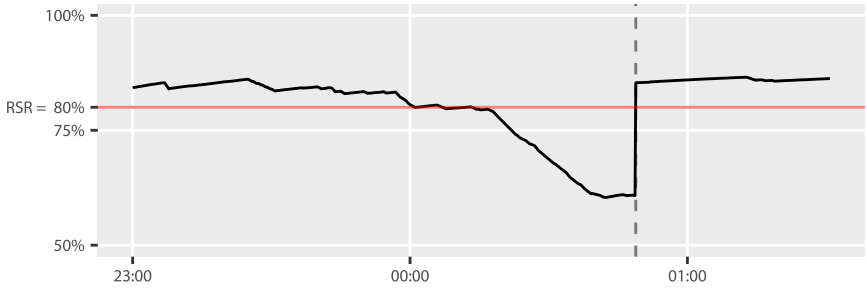
(a) **Rush Hour.** During rush hour, the obstacle Ambulance Stuck In Traffic Jam Toward Incident has an increased satisfaction rate. This increase caused the countermeasures Achieve [Ambulance Stuck In Traffic Jam ReAllocated] and Achieve [Patient Transported At Hospital When Required And Traffic Jam] to be selected, integrated and deployed in the running system. Figure 9.15(a) shows the satisfaction rate of the root goal increasing again beyond its RSR threshold, as the two obstacles no longer obstruct it. The deployment



(a) Rush Hour



(b) Night Mobilization



(c) Status Forgetting

Figure 9.15 Simulation Scenarios.

of the first countermeasure changes the software. The second countermeasure does not alter it; the goal specification is only relaxed to allow more time between the incident scene and the hospital.

(b) **On-Road Mobilization.** At night, the RSR for the goal Avoid [Ambulance On Road Mobilized] is .5. During the day, however, ambulance staff prefer to not intervene in multiple incidents without going back to their station. The RSR is increased to .8, as Figure 9.15(b) shows. As a result, the countermeasure Achieve [Ambulance Allocated At Station When Incident Reported] was deployed during the day. The adaptation replaced the *DefaultAllocator* component. The new allocation strategy reduced the satisfaction rates of the obstacle Ambulance Mobilized On Road and guaranteed the goal's RSRs.

(c) **Forgetting Ambulance Status.** Late at night and early in the morning, ambulance staff tends to forget to push buttons. This results in an increase in the satisfaction rate of obstacles such as Accurate Leaving Status Not Encoded On MDT Or Innaccurate. During that period, the countermeasure goals such as Achieve [Automated Leaving Status Detection] and Achieve [Automated OnScene Status Detection] were selected, integrated and deployed. The integration caused the computed satisfaction rate to increase again beyond their RSR threshold. Figure 9.15(c) shows the satisfaction rate of Achieve [Ambulance On Scene When Incident Reported].

Our techniques were felt to help significantly for the following reasons.

Precise semantics regarding behaviors. Thanks to formal specifications being anchored on real-world phenomena, the mapping between predicates and the running software system was straightforward. For example, the evaluation of the predicate ambulanceA9OnScene triggered a simple database query.

All monitored items had a clear, precise meaning. Surprising results were easily understandable. In particular, our technique identified inaccurate specifications with missing conditions or unrealistic time constraints.

Traceability of monitored indicators and deployed countermeasures. The monitored satisfaction rates of high-level goals significantly helped understand whether an increase in the satisfaction rate of an obstacle is critical. Comparing their monitored satisfaction rate with their RSR provided a traceable criterion for system adaptation.

For example, the activation of the *TrafficJamAllocator* software component in the ADS is directly traceable to its countermeasure goal Achieve [Ambulance Stuck In Traffic Jam ReAllocated]. The latter in turn is traceable to high-level goals such as '*an ambulance shall be on scene within 12 minutes*'.

Model-based adaptation. The goal/obstacle model drives the selection of most appropriate countermeasure. As the results showed, their dynamic selection ensured that the high-level probabilistic goals remained satisfied. Without our technique, the monitoring and adaptation of the ambulance dispatching software would have required dedicated, application-specific code to be written.

No explicit behavior modeling. The ambulance dispatching system exhibits complex states and parallelism among processes. Building a complete, consistent, and adequate state machine model for this system appears quite hard.

Other benefits. The formalization effort was kept minimal as only leaf obstacles needed to be formalized. In addition, a change in the formal specification of a leaf obstacle did not require source code modification.

This validation case study also highlighted areas for improvement. In particular, the satisfaction rates estimated by experts (as shown in Section 6.3) might be used to improve the accuracy of the monitored satisfaction rates in case only a few data are available. The monitored satisfaction rates should also be filtered to smooth out the noise caused by a low number of observations. The technique in [Fil15a] might be used to improve the quality of monitoring.

As in many monitoring-based self-adapting systems, the monitoring task may impact on the performance of the monitored system. Our preliminary experience suggests that most of the impact may be transferred to a separate computer to reduce the footprint on the monitored system.

9.3.4 Discussion

In addition to the discussion regarding the specific objectives of the techniques as presented in the previous subsections, the following briefly discusses the applicability of our techniques:

- Goal and obstacles models have been applied to a wide variety of application domains; see for example, [Cai13b; Dar07; Lam09; Let02; Lut07; Pon07]. We see no critical difficulty in building goal and obstacle models.

- The other inputs required by our approach are the estimated satisfaction rates for leaf obstacles and estimated probability distributions to be elicited from domain experts. Tools and methodologies are available for this as discussed in [Coo91; Otw92; Vos08]. Other approaches also rely on estimated values, such as [Hea11; Lam00; Let04; Let14], providing additional examples of the applicability of eliciting the required inputs.

- The Car Pooling System and Ambulance Dispatching System showed how the proposed techniques for *obstacle assessment* and *control* are applicable to those type of software systems. The C230-YLS system showed how the proposed techniques are applicable for assessing the human safety risks. The applicability of our techniques to other application domains and to other type of risk analysis, such as security risk analysis, should be further evaluated.

- We played the role of the analysts in applying the techniques. Future studies should demonstrate the applicability of the techniques by people with a different expertise. In particular, the cost for training risk analysts, building the goal and obstacle models and applying the proposed technique should be compared to the cost of other risk assessment and control techniques. Whether our techniques are cost-effectively applicable by risk analysts should be assessed.

- The C230-YLS case study suggests that our techniques may integrate with existing practice such as FMECA. However, how the proposed techniques may be integrated, at a larger scale, within the existing practices in industry remains an open question. For example, the extend to which the risk analysis process currently in use at IBA need be modified should be carefullyevaluated.

9.4 Utility

This section briefly discuss the extent to which our proposed techniques solve a real problem. The problems the thesis focuses on are:

- the requirements completeness problem addressed by the techniques presented in Chapter 3, 4 and 5;
- the handling of knowledge uncertainty addressed by the techniques presented in Chapter 6;
- the risk-driven runtime adaptation addressed in Chapter 7.

Addressing the completeness problem. This problem is recognized as a major challenge in software engineering. As discussed in the introduction of Chapter 3, such incompleteness arises from our tendency to conceive over-ideal systems; integrating risk analysis into the requirement process helps producing more complete and accurate requirements [Lam09]. Moreover, increasingly many software systems are built according to models [Bra12; Sta06] and are driven, at runtime, by models [Ben07]. These models shall thereby be as complete as possible to provide robust software systems. Our techniques complement the obstacle analysis process to support quantitative reasoning on the satisfaction rate of obstacles and goals. Our techniques felt significantly helpful to identify missing risks (as seen in the IBA C230-YLS case study) or inaccurate specifications (such as in the ADS case study). Besides, our techniques felt helpful for focusing on likely and critical risks and most appropriate countermeasure selection; these problems were confirmed by IBA engineers to be major challenges, as seen in the case study.

Addressing uncertain estimates. As discussed in the introduction of Chapter 6, knowledge uncertainty impedes the application of standard risk prioritization techniques. The problem of managing uncertainties is recognized to be a major issue [Che09b; Coo91; Let14; Men03]. Our techniques were felt significantly helpful identifying and controlling uncertainty margins, as seen in the Ambulance Dispatching System case study in Section 9.3.3.

Addressing risk-driven runtime adaptation. Today's software systems should be flexible, robust, and efficient in adapting to changing operational contexts and environments. The problem of software adaptation at runtime is recognized to be a challenge for modern software engineering practice [Cheo9a; DeL13; Gie13]. Our techniques can be used to drive software adaptation by high-level goal satisfaction, relying on identified obstacles to highlight most appropriate adaptations to deploy.

9.5 Usability

This section briefly discusses the usability of the techniques proposed in this thesis. Criteria proposed in [Nie94] for evaluating the usability of a design are adapted here for our techniques:

- *Learnability*: How easy is it for users to apply the techniques?
- *Errors*: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?
- *Efficiency*: Once users have learned the techniques, how quickly can they perform tasks?
- *Memorability*: When users return to the techniques after a period of not using it, how easily can they reestablish proficiency?

Learnability. The goal/obstacle languages are used in industrial settings, as seen in [Cai13b; Daro7; Lamo9; Leto2; Luto7; Pono7]. Other techniques such as CORAS [Lun10] or Fault Tree Analysis [Rui15; Ves81] require a similar level of expertise in model building.

Specifically,

- For obstacle assessment, as seen in the ADS or the IBA C230-YSL case-studies Sections 9.3.3 and 9.3.2, domain experts were, in practice, able to estimate the satisfaction rates for leaf obstacles.

- For obstacle control, applying our techniques presents no specific difficulty once the goal and obstacle models are built and leaf obstacles are estimated. How easily risk analysts can select most appropriate countermeasures remains an open question and should be further evaluated.

- For uncertain estimates, as seen in the ADS in Section 9.3.3, domain experts were able to estimates the satisfaction rates for leaf obstacles together with their uncertainty margin. The extends to which the metrics *violation uncertainty* and *uncertainty spread* are easy to interpret by risk analysts remains an open question to us.

- For runtime adaptation, our techniques rely on formal specifications of probabilistic obstacles. While formal specification remains a challenge [Fra91], elicitations techniques and specification patterns [Dar95; Dwy98] may be used to ease the production of formal specifications [Lam09]. Future work should more precisely evaluate the impact of formal specifications on the usability of our monitoring techniques.

Errors. The level of feedback in case of errors provided by the tools implementing the techniques should be improved for better usability by users less familiar with the internal details of our techniques. In addition, the interpretation of the results should be performed with care. With proper training, we see no major impediment here. This should however be rigourously tested.

Efficiency and memorability. Future work should consider case studies where the role of the risk analysts is not played by ourselves in order to better evaluate the efficiency and memorability of the proposed techniques.

9.6 Summary

The chapter evaluated the correctness, performance and scalability, applicability, utility and usability of the techniques for *obstacle assessment* and *obstacle control*, in presence or not of knowledge uncertainty, and both at RE time and

at runtime. The proposed techniques are shown to be correct with respect to their specification. The complexities and performance of those appears to be reasonable in practice. Three case-studies suggest that our techniques may be applicable to realistically-sized problems. Two case-studies, the CPS and the ADS are realistic case-studies showing the applicability of our techniques to this kind of software systems; the IBA C230-YLS is a real industrial case study showing the applicability of our techniques for assessing human safety risks. Last, our techniques are shown to be useful and usable in realistic situations.

Chapter 10

Related Work

This chapter reviews relevant work on models and techniques for risk analysis, quantitative requirements, uncertainty in requirements, exception handling in requirements and requirements-driven runtime adaptation. The work discussed here is restricted to software systems.

The chapter is structured as follows. Section 10.1 reviews the work related to risk analysis not explicitly connected to requirements, such as Fault Tree Analysis (FTA), HAZOP/HAZAN, FMEA/FMECA, and CORAS. Section 10.2 discusses approaches for handling quantitative requirements. Section 10.3 discusses models and techniques for risk analysis related to requirements. Section 10.4 addresses requirements uncertainty handling. Section 10.5 discusses models and techniques for exception handling. Last, Section 10.6 details other runtime adaptation models and techniques and compares these to our approach.

Our review does not include specific security risk analysis techniques such as *attack trees* and *threat trees* [Hel02; Sch11; Sch99] (these are however close to *fault tree* addressed here), *UMLSec* [Jur01; Jur02], *SecureUML* [Lod02], *Abuse Case* [McD99], or *Misuse Case* [Sin05].

10.1 Risk Analysis

This section briefly presents common risk analysis models used in the industry. A more detailed overview can be found in [Lev01].

Fault Tree Analysis. Fault Tree Analysis (FTA) is a widely used method for analyzing risks related to safety [Bed01; Lam09; Lev01; Rui15; Ves81]. Application domains range from aeronautics to software systems [Lev83]. A comprehensive

introduction to FTA is provided in [Ves81]. We briefly discuss Standard Fault Trees (SFT) and Dynamic Fault Trees (DFT), and refer to [Rui15] for a more detailed discussion about FTA state-of-the-art techniques.

Fault trees (FTs) capture how failures propagate through a system, more specifically, how a basic event leads to a system failure. An FT is an acyclic graph with two types of nodes: events and gates. The events of an FT capture failures whereas the gates are non-leaf nodes capturing failure propagation. A leaf event is called a *basic event*; a non-leaf event is known as an *intermediate event*. Gates includes *AND* and *OR* gates for AND-combinations and OR-combinations of the child events, respectively. A *k/N* gate captures a *voting* gate where at least k child events among N must be *true* for the output to be *true*.

The analysis of a fault tree is either qualitative or quantitative. *Qualitative* analysis focuses on the structure of the fault tree; it includes identifying *cut sets*, i.e., combinations of basic events leading to the failure of the system. Different techniques are available for efficiently computing these cut sets, such as bottom-up propagation or BDD-based computation [Rui15]. *Quantitative* analysis focuses on computing failure probabilities such as the probability that a system fails given some horizon or the average duration between two subsequent failures. Low-level events are annotated with probabilities that are propagated from causing events to their consequences in the fault tree. The probabilities can be single-value or multi-value estimates such as probability distributions. A large number of techniques are available for computing the probability of root events [Rui15].

Standard Fault Trees (SFTs) provide a simple model for risk analysis but lack expressiveness. Dynamic Fault Trees (DFTs) extend SFTs to support finer-grained analysis. DFTs include additional gates such as *Priority-AND* (inputs must be *true* in a specific order), *FDEP* (Function DEPendency where inputs influence other inputs) and *SPARE* (model components replaced by others). The quantitative analysis of a DFT is usually performed by transforming it into a Markov Chain that can be further analyzed using standard techniques such as Model Checking [Bai08].

HAZOP/HAZAN. This approach focuses on the identification of hazards and operability problems [Kle99; Lev01]. It deals with deviations from the intended design, their possible causes and their consequences [IEC61882]. Originally designed for chemical plants, the HAZOP approach has been extended and used in different settings such as administrative procedures, medical devices [IEC61882], and software systems [Bur93; Chu93; Ear92; Fen94; McD02; McD94; McD95; Par07].

An accurate and detailed design is required to carry out an HAZOP analysis. At later stages, it is however more costly and sometimes impossible to introduce major changes in the design; [Kle99] reports on the application of HAZOP analysis on preliminary designs.

The system under scrutiny is divided into parts. Each part is examined and challenged by the use of guide words, such as NO OR NOT, MORE, and LATE, to feed the elicitation process. Specific guide words were developed for software systems [Bur93; Han04; McD02; McD94; McD95]. For these, UML attributes and entities may be used as a basis for the application of guide words [Han04].

In the context of chemical processes, [Vai96] extends expert systems, aimed at generating HAZOP analysis with quantitative information about the processes and the materials used. The consequences of a hazard are then ranked according to their severities.

Once identified, hazards need be assessed. The assessment may be based on on standard consensus techniques or rely on Fault Tree Analysis to compute more accurate estimates [Kle99]. Once assessed, the consequences for the surrouding environment are estimated. Hazard assessment and consequence assessment are then compared with respect to target criteria [Kle99].

FMEA/FMECA. Failure Modes and Effects Analysis (FMEA) is a systematic technique for identifying failure modes of a system and for evaluating the consequences of failure modes [Haa02; Lev01]. Failure Modes, Effects and Criticality Analysis (FMECA) extends FMEA with quantitative analysis. The identified failure effects are organized according to their criticality (on an x-axis) and likelihood (on a y-axis) in a *criticality matrix*. Such criticality matrix helps risk engineers identify likely and critical failures. The reader is referred to [Haa02] for a detailed survey related to the application of FMEA to software systems.

CORAS. CORAS is a model-driven risk analysis methodology with a strong focus on defensive security analysis [Lun10]. It is concerned with protecting a set of identified assets. The approach is structured in eight steps. Steps 1 to 4 focuses on preliminaries for risk analysis. These steps are somewhat out of scope of this review as these are not specific for risk assessment and control. Step 5 to 8 however directly relate to identify-assess-control cycles.

- Step 5 (Risk identification) is driven by the protection of the assets. This step produces a threat diagram identifying the risks harming the identified assets. An *asset* is defined as 'something to which a stakeholder assigns value and hence for which the stakeholder requires protection'. It may refer to physical objects (such as computers), virtual elements (such as customer database or online store), or desirable properties (such as compliance or reputation) [Sol11]. An *unwanted incident* is an event that harms or reduces the value of an asset; a *threat* is a potential cause of an unwanted incident; a *threat scenario* is a chain or series of events that is initiated by a threat and may lead to an unwanted incident; a *vulnerability* is a weakness, flaw or deficiency that results in a threat. Those elements and their inter-relations are identified during a structured workshop where experts and risk analysts identify as many elements as possible.

- Step 6 (Risk estimation) focuses on risk estimation. During a structured workshop, the unwanted incidents are annotated with a likelihood. A *likelihood* is defined as the frequency or probability of something occuring. The estimates may be qualitative or quantitative. To help support the estimation, a calculus is provided to compute the likelihood of a dependent artifact. The calculus may be used to check whether the estimate of the unwanted incident is coherent with the other estimates of threats and threat scenarios. Such technique may prove effective for identifying incomplete, inconsistent or ambiguous risk analysis [Ref15]. Likelihoods can be specified using equations combining key indicators whose value might change over time [Ref09]. Little support is provided to identify key indicators, combine these and monitor them.

The uncertainty on estimates can be captured using intervals instead of single-point values [Sol11]. The calculus is extended to support such intervals. Distributions can replace intervals for a more refined analysis—to the best of our knowledge, this has not been done so far.

In CORAS, a *consequence* is the impact of an unwanted incident on an asset; it is captured in terms of harm or reduced asset value. The consequences are estimated on a qualitative or quantitative scale. Most commonly, they are classified as insignificant, minor, moderate, major or catastrophic. For specific assets, a quantitative value, such as monetary or time, might be used.

- Step 7 (Risk evaluation) prioritizes estimated risks. The unwanted incidents are displayed on a risk matrix whose rows are frequencies and columns are consequences. To identify likely and critical risks, this matrix indicates risks that are acceptable and risks that are unacceptable. Which cell is acceptable is determined by risk analysts and domain experts.

- Step 8 (Risk treatments) is concerned with the identification, assessment, and selection of countermeasures to unacceptable risks. In CORAS, *treatments* are appropriate measures for risk reduction. Risk analyst and domain experts identify possible treatments during a structured workshop. Treatment categories such as avoid, reduce consequences, reduce likelihood, transfer, or retain help explore alternative treatments. Once those are identified, their cost and reduction effect in both likelihood and consequences are estimated. The benefit of a treatment regarding an asset is computed as a combination of the risk probability, the risk reduction, and the consequences. Best treatments are then selected during the workshop. A technique for systematic treatment selection when estimates are quantitative is proposed in [Sol13b].

More details about these steps can be found in [Sol11]. The approach is supported by dedicated tool support [See12; Sol11]. A wide variety of case-studies [Bra07a; Bra07b; Ref15; Sol13a] from industries with varying size illustrate and validate the practical applicability of the approach.

Comparing common risk analysis techniques with our approach. Some differences between the preceding approaches and our may be observed:

- Their lack of connection with a goal model may make it hard to identify root events for starting backward causal analysis. Fault Tree analysis is often anchored on Event Trees [Lev01] or used as a complementary technique to provide more precise estimates for the risks identified with HAZOP/HAZAN or FMEA/FMECA. The CORAS methodology anchors risk analysis on assets. This appears to be a fairly vague and unprecise starting point for a risk analysis.

- Unlike the preceding approaches, our work is anchored on a refinement structure refering to probabilistic goal/obstacle assertions. In Fault Tree Analysis and CORAS, the risk trees are not grounded on any formal apparatus; their nodes are just event names and their causal links have no precise semantics. As a consequence, the correctness of event decompositions cannot be established, and the propagation rules may appear ad hoc. In HAZOP/HAZAN and FMEA/FMECA, there is no structure at all for linking risks.

- In the preceding approaches, likelihoods and their contributions have no precise semantics that would enable formal reasoning or runtime monitoring.

- Our approach is integrated into a comprehensive method for risk identification, assessment, and resolution. FTA proposes no mechanism for risk resolution. None of the approaches reviewed in this section integrates the selected countermeasures back into some model. Lastly, none of these approaches supports risk analysis cycles where risks to countermeasures are identified, assessed and controlled (see Section 5.2).

10.2 Quantitative Requirements Frameworks

The work reviewed in this section focuses on models and techniques for quantitative requirements that are not explicitly connecting requirements to risks. This section exclusively refers to quantitative analysis. The reader may check [Hor11] for available qualitative analysis techniques on requirements models.

KAOS-based approaches. An approach for probabilistic reasoning on goal models to select the most appropriate alternatives is proposed in [Leto4]. Goals there are annotated with quality variables such as *Failure Rate* or *Response Time*. These quality variables are estimated on leaf goals using probability distributions that are analytically combined to obtain probability distributions for the quality variables of top goals. The combinations are obtained by applying domain-specific propagation equations that annotate the refinement structure. High-level goals are decorated with objective functions to be maximized or minimized, together with some target value.

The work in [Ell14; Lut12a; Lut12b] discusses how quantitative goal-oriented RE helped in designing molecular systems. In particular, [Lut12b] introduces *THRESHOLD* refinement nodes. In such refinement, the parent goal is satisfied if 'enough' subgoals are satisfied. 'Enough' subgoals are satisfied if at least k subgoals are satisfied among all N possible subgoals.

The work in [Dar15; Pon17] evaluates how quantitative reasoning can be used to assess accessibility in physical buildings. The approach differs from common approaches by providing a domain-specific assessment tool for goal satisfaction.

Tropos-based approaches. A formal framework is introduced in [Gio02; Gio03] for reasoning about goal models. This framework was applied in [Gio05a] to Tropos models [Bre04]. In addition to goal refinement links, the model includes qualitative relationships among goals. These relations include:

- $G_1 \xrightarrow{++} G_2$: the satisfaction of G_1 implies the satisfaction of G_2.
- $G_1 \xrightarrow{+} G_2$: there is evidence that the satisfaction of G_1 implies the satisfaction of G_2.
- $G_1 \xrightarrow{-} G_2$: there is evidence that the satisfaction of G_1 implies the denial of G_2.
- $G_1 \xrightarrow{--} G_2$: the satisfaction of G_1 implies the denial of G_2.

Those relations might be annotated with a weight to enable quantitative analysis. This weight represents the strength by which the satisfiability/deniability of the source goal influences the satisfiability/deniability of the target goal.

Goals there are annotated with two attributes, *SAT* and *DEN*, that quantify the value of evidence for the goal being satisfied or denied, respectively. The values of these attributes are qualitative and range from *Full* to *Partial* to *None*, or from a real variable *inf* to a real variable *sup* (to be specified by the user).

An algorithm uses propagation rules to compute the *SAT* and *DEN* values for the root nodes. The algorithm starts from leaf goals and propagates the values to the goals directly connected to these leaf goals. The process is then repeated until a fixpoint is reached.

Use of i* models for selecting architecture components. The work in [Fra03; Fra04] reports on the use of i* goal models [Yu97] to select architectural components. The latter are modeled as i* actors; the selection of most appropriate components is driven by the satisfaction of soft goals. Soft goals are quantified using architecture-specific metrics such as diversity, packaging, uniformity, connectivity. These metrics are computed for alternative architectures to drive the selection of a most appropriate one. It is not clear how these metrics are estimated though.

Quantitative GRL. A quantitative reasoning layer to the Goal-oriented Requirement Language (GRL) is introduced in [Amy10]. The reasoning is aimed at comparing the relative effectiveness of alternative designs. Satisfaction values are assigned to nodes of a goal model graph and propagated to the other nodes through decomposition, contribution and dependency links. The propagation may be forward (from leaf nodes to root nodes) or backward (from root nodes to leaf nodes), or a mix of both. Forward analysis answers 'what if?' questions whereas backward analysis focus on 'is it possible?' questions. Qualitative assessment and hybrid assessment mixing both qualitative and quantitative information, are also proposed.

Markov Logic Network-based approach. In [Cha13], the degree of satisfaction for a goal is determined by the degree of satisfaction of its subgoals together with the degree of satisfaction of other goals connected through *Contribution Links*.

The latter indicate whether the source goal contributes to the satisfaction or violation of the target goal. These links are weighted according to the probability for a source goal to satisfy or deny the target goal. This approach generates a Markov Logic Network, similar to a Bayesian network, from a goal model. The network is then trained with past data to obtain numerical values for the contributions links.

Comparing quantitative requirements frameworks with our approach. The preceding approaches are not connected to risks; their focus is more on selecting alternative designs, such as selecting alternative goal refinements or selecting alternative agents to satisfy high-level objectives.

Our approach differs in several directions from the previously presented approaches.

- Unlike all preceding approaches, our approach links the partial satisfaction of goals to the occurrence of risks. Requirement satisfaction is determined from the risks preventing satisfaction. With a risk AND/OR-refinement structure for propagating probabilities from fine-grained risks, the estimates are easier to obtain and interpret in terms of application-specific phenomena. Moreover risks document the rationale for the partial satisfaction of the goals. (A notable exception is [Let04] where quality variables can be obtained and interpreted fairly easily.)

- Unlike [Amy10; Cha13; Fra03; Fra04; Gio02; Gio03; Gio05a], our approach relies on a precise semantics in terms of behaviors. This enables estimates to be grounded on real-world phenomena.

- The approaches proposed in [Amy10; Cha13; Gio02; Gio03; Gio05a] propagate partial satisfaction through refinements and contribution links. Our approach does not include such contribution links. As the goals there appear to have no precise definition in terms of behaviors, it is unclear how contributions links are defined and how they differ from partial refinements in a quantitative setting.

- Unlike [Amy10; Cha13; Fra03; Fra04; Gio02; Gio03; Gio05a], our approach introduces probabilistic goals and compares *computed* satisfaction rates with *required* satisfaction rates. In [Let04], objective functions are decorated with a target value—somewhat similar to our required satisfaction rate.

- Our framework takes a probabilistic approach; goals and obstacles are estimated with a single value indicating both their satisfaction (sr) and their non-satisfaction ($1-sr$). In [Amy10; Cha13; Gio02; Gio03; Gio05a], goals are estimated with two values indicating the satisfaction (SAT) and their non-satisfaction (DEN). This increases the number of elements to be elicited by experts. Moreover, the underlying theory might not be as well understood as probability theory. In particular, experts might overlook that $SAT \neq 1 - DEN$.

- The approaches in [Fra03; Fra04; Let04] rely on domain-specific variables and equations. The latter are measurable and falsifiable. They might also guide the construction of the models and perhaps be easier to understand/estimate in terms of application-specific phenomena. They appear to be finer-grained as they rely on domain-specific knowledge. However, those variables and equations must be defined in an ad-hoc manner. A deeper comparison involving practitioners would be interesting for better understanding of the advantages and disadvantages of two approaches.

- The domain-specific assessment tools in [Dar15; Pon17] might be easily integrate with our proposed technique for obstacle assessment.

- The work in [Ell14; Lut12a; Lut12b] introduces a new type of refinement node called *THRESHOLD* node. This is somewhat similar to a k/N gate in Fault Tree Analysis. *THRESHOLD*-refinements are not probabilistic refinements; the parent goal is satisfied if 'enough' subgoals *are* satisfied, not if 'enough' subgoals *could be* satisfied; this differs from our approach.

- The backward reasoning applied in [Amy10] for identifying the leaf goals that are sufficient for satisfying the parent goal is not relevant to our framework as we do not consider alternative goal OR-refinements. However,

the sets of leaf obstacles preventing the satisfaction of the parent goal can be computed in our approach through obstruction supersets (see Section 4.2.1).

10.3 Risk-Driven Requirements Engineering Approaches

This section reviews approaches where requirements and risks are integrated in a unified qualitative or quantitative framework.

DDP. The 'Defect Detection and Prevention' (DDP) process originated from NASA Jet Propulsion Lab [Coro1; Fea03; Fea08a]. It is designed as an aid to decision making in the early phases of a project. In these phases, information on which decisions are made are often incomplete and uncertain. The process is structured in four steps: *(1)* Overview the problem and DDP methodology; *(2)* Develop the impact information; *(3)* Develop effect information; *(4)* Decision making. The main concepts of the approach are *requirements* (what the system shall satisfy), *failure modes* (what could prevent the requirements from being satisfied) and *PACTs* (what can be done to reduce the likelihood or impact of a failure mode). (PACT stands for 'Preventions, Analyzes, process Controls, and Tests.') A detailed presentation of the concepts and the process can be found in [Fea03]. The approach is supported by a tool, with a strong focus on decision making and visualization to help in decisions [Fea03]. The process has been used internally at NASA for a long period in various settings, see [Fea08b; Fea08c; Sha06].

A *requirement* is defined as '*whatever the system under scrutiny is to achieve, and the constraints under which it must operate*'. Requirements are specified in natural language. They are organized in trees that can be seen as folders grouping requirements according to the modeler's conventions. The trees do not convey any structural meaning. Leaf requirements are decorated with a weight, indicating their relative importance with respect to other requirements. The weight of leaf goals can be inferred from the weight of higher-level goals [Mes04].

A *failure mode* is defined as '*a thing, that, should it occurs, will lead to loss of requirement*'. Failure modes are informally specified using natural language. They are organized in OR-trees, indicating an alternative way of causing a parent failure mode to happen. Failures mode are identified during structured workshops where previous DDP analysis helps the experts to determine the risks from similar systems or settings [Fea03].

The identified failure modes impact requirements. An impact matrix quantitatively specifies, for each requirement, the proportion (between 0 and 1) that would be lost if the failure mode occurs . If multiple failure modes impact a requirement, the impacts are summed, leading to values that might be above 1. In such cases, the sums are capped to 1 when evaluating how the requirements are attained. Note that summing without capping would better emphasize the amount of risk to overcome for requirement satisfaction. For example, an impact of 1.8 might requires more effort to be reduced than an impact of 1.1.

Failure modes are decorated with a likelihood and a cost of repair. Likelihoods and costs are specified by experts during workshops where they need to agree on a single value. For each failure mode, a global impact can be computed by summing the impact of the weighted requirements. This helps identify most critical failure modes.

By taking PACTs into account, failure modes that are not enough mitigated can be highlighted. PACTs include preventive measures, detections, and alleviations. *Preventive measures* include, for example, training, standards, selection of high-quality parts. *Detections* discover instances of the failure mode through analysis or testing prior to their use or release. *Alleviations* reduce the severity of the failure mode such as defensive programming. A PACT is decorated with a cost that might be a measure of budget, schedule, physical resources (such as weight or power), scarce resources (such as time from skilled experts), or a combination of these.

The *effect matrix* quantitatively specifies the effect of a PACT on a failure mode. It contains a row for each PACTs and a column for each failure mode. Each combination of PACT and failure mode needs to be assessed. The effects are multiplicative, and the order in which they are applied is not relevant.

The selection of most appropriate PACTs is supported by a wide selection of visualizations [Fea06]. These appear to facilitate the decision making process [Cor02]. Efforts have also been provided to support semi-automated selection of best PACTs given a budget. A large number of combinations requires optimization techniques such as genetic algorithms [Cor02]. A dedicated technique rely on a learning framework to iteratively select the PACTs that reduce uncertainty and increase requirements satisfaction the most [Men03].

Goal-Risk Framework. The work in [Asn06a; Asn11] extends the Tropos framework with risks and countermeasures. A *risk* there is an event having an adverse impact on goals. A *countermeasure* is a treatment that can be adopted to mitigate the effect of a risk. The framework is structured in three layers: a *asset layer*, an *event layer*, and a *treatment layer* capturing *impact* links between events and goals, and *effect* links between treatments and events. An event is annotated with a (risk) likelihood; a (risk) severity is expressed as a contribution (positive or negative) from an event to goal. Events and treatments can be AND-/OR-refined and can contribute to other goals, events, or treatments. Different strategies are available for identifying possible treatments [Asn06b]. The work in [Asn06a] was extended to support treatments mitigating the impact of an event [Asn06b], trust among actors of the system [Asn07a], and quantitative reasoning [Asn07b].

Events (respectively, treatments) are annotated with *SAT* and *DEN* attributes that quantify the evidence that an event (respectively, the treatment) is satisfied or denied. A treatment is *effective* for an event if it introduces a conflict between the *SAT* and the *DEN* values of the event (e.g., the event is fully satisfied and fully denied).

The analysis proposed in [Asn06a] aims at identifying solutions that satisfy the desired values for high-level goals while minimizing a total cost. A solution is a set of leaf goals and treatments. As input, their analysis takes the desired satisfaction values (i.e., minimal SAT-value accepted) and the acceptable risk values (i.e., maximal DEN-value accepted) for the top goals. Using backward analysis [Hor10; Seb04], leaf goals that are sufficient for satisfying the root goals are identified. Forward analysis [Gio05a] propagates the estimated SAT/DEN-values of the events to the root goals in order to assess whether treatments need

be introduced. The algorithm identifies admissible values for the events and then possible countermeasures values that guarantee these event values. Such analysis can also be performed on quantitative inputs [Asn07b].

The work in [Sha12; Sha13] proposes an extension to the Goal-Risk framework for prioritizing risks with a Risk/Cost analysis. It is, however, unclear how prioritization is achieved and how costs are obtained.

KAOS-based approaches. In [Sab11], KAOS goal models are extended with probabilities and propagation rules for technology qualification. Domain experts there assess the probability of satisfaction of leaf obstacles. These probabilities are then propagated bottom-up up to the root goals using propagation rules.

RiskML. A risk-oriented framework is introduced in [Sie14] for selecting Open Source Software components. The approach introduces a language, called *RiskML*, to relate risks to goals. A *risk* there is defined as '*a composed concept modeling a lack of knowledge about some happening and what could be the negative consequences*' [Sie14]. Risks are not captured as first-class entity but as a triplet of situations (in which the events occurs), events (that prevent the satisfaction of goals) and goals (that are not satisfied). A *situation* is a state of the world that exposes or prevents events, and increases or decreases their probability of occurrence. *Indicators* quantify the occurrence of situations. These need to be estimated by experts. A propagation algorithm, inspired from [Gio03], computes scores representing the exposure of goals to the identified risks using the estimated indicators. These scores provide a relative measure for comparing alternatives; they are not meant to be interpreted as a probability or degree of belief. Integrating RiskML and i* models enables the degree of belief of higher-level goals to be determined from the estimated indicators [Cos15].

Comparing risk-driven RE approaches with our approach. Like ours, the preceding techniques integrate both requirements and their related risks. The following differences can however be noted:

- Unlike [Asn11; Fea03; Sab11; Sha12; Sha13; Sie14], our approach relies on a precise semantics in terms of behavior. This enables estimates to be grounded on real-world phenomena. In general, these numbers turn to be subjective, with no or little physical interpretation in the application

domain, see [Kaio2]. In [Sab11], the lack of precise semantics in terms of behavior limits the degree of precise reasoning and even questions the correctness of propagations. For example, the OR-propagation rule for obstacles can be made simpler thanks to obstacle disjointness.

- Our framework follows a probabilistic approach; In [Asn11; Sha12; Sha13] a different approach is taken where goals are estimated with two values indicating satisfaction (SAT) and non-satisfaction (DEN).

- Unlike [Asn11; Fea03; Sab11; Sha12; Sha13; Sie14], we support probabilistic goals and make a distinction between *computed* and *required* satisfaction rates. In [Sab11], the notion of required satisfaction rate does not appear relevant to the objectives of this approach; obstacle prioritization is therefore different.

- Unlike [Fea03], the refinement structure of goals and obstacles convey a structural meaning enabling the propagation of probabilities through risks and consequences at various levels of granularity.

10.4 Requirements Uncertainty Handling

Work on handling uncertainty in RE has been mainly devoted to physical uncertainty rather than knowledge uncertainty (as defined in Chapter 6). For a detailed discussion and state of the art related to uncertainty in the specific context of self-adaptive systems, see [Esf13].

Fault Tree Analysis. Knowledge uncertainty is supported in Fault Tree Analysis through probabilistic distributions on leaf events and Monte-Carlo simulation for propagation to root causes [Rui15].

DDP. In [Meno3], a dedicated technique helps in automatically identifying the most important countermeasures in the presence of uncertainty. The approach iteratively selects countermeasures to reduce uncertainty and increase requirements satisfaction.

KAOS-based approaches. Work on handling physical uncertainty through obstacle analysis was discussed in the previous section. The approach in [Leto4] is extended in [Hea11] for selecting best system design alternatives in presence

of uncertainty. Quality variables there are random variables whose value is modeled as a probability distribution. A top probability distribution is obtained through repeated single-value propagation. The approach is supported by a tool [Bus17]. Based on a framework for statistical decision making, the technique in [Let14] evaluates the gain of perfect information and helps identify alternatives maximizing benefits while reducing project risks.

The work in [Sab11] extends KAOS goal/obstacle models with a probabilistic layer for technology qualification (as introduced in Section 10.1). The probability of satisfaction for leaf obstacles is estimated by a triangular distribution. The distribution for the root goal is then computed by Monte-Carlo simulation and a single-value propagation algorithm.

Learning-based approach. Uncertainty may also refer to missing or (partially) wrong obstacles. A learning-based approach may be used to generate a set of obstacle specifications systematically from known domain properties [Alr12]. The generated obstacles are guaranteed to be complete and consistent with the domain. They are learned from traces generated by model checking the domain against a goal which produces counterexample traces and witness traces.

Fuzzy-based approaches. Fuzzy goals may be introduced for coping with satisfaction uncertainty. The available approaches are more oriented towards self-adaption at system runtime. The concern there is to reason in terms of proximity of goals to being fully satisfied—rather than in terms of probabilities of satisfaction.

RELAX is a structured natural language including operators for explicit capturing environmental uncertainty [Whi09; Whi10]. Operators, such as *MAY*, *AS EARLY AS POSSIBLE*, or *AS MANY AS POSSIBLE*, constraints how a goal can be relaxed at runtime. The semantic of RELAX is defined in terms of fuzzy logic. A variation of threat modeling, based on obstacles, identifying the sources of uncertainty in the satisfaction of goals [Cheo9c]. Countermeasures there either change the goal model or RELAX goal specifications. In [Fre14; Ram12a], RELAXed goals are automatically learned using an executable specification of the system. The approach can be summarised as follows: *(a)* a search process semi-randomly generates RELAXed goal models; *(b)* the RELAXed goals are scored by running the executable specification and monitoring the satisfaction of the goals; *(c)* if enough iterations have been performed, the process stops and

outputs a RELAXed model; otherwise, new RELAXed models are generated and *(b)* is executed again. In [Welo9; Wel10; Wel11], claims are introduced to document and support decisions and are RELAXed in [Ram12b] to improve the number of required adaptations.

[Bar10] introduces *fuzzy goals* to provide more flexibility for systems where goals might not be measured or whose specification is not entirely known. Crisp goals are goals that must be fully satisfied, in constrast with to fuzzy goals where fuzzy membership functions quantify the degree of goal satisfaction. A fuzzy logic-based approach integrates both crisp and fuzzy goals [Cha16a; Cha16b]. A bottom-up propagation algorithm computes the degree of satisfaction for both types of goals. The degree of satisfaction of a fuzzy goal depends on the degree of satisfaction of the children in the refinements, together with the degree of satisfaction of the goals that positively or negatively contribute to that goal. The algorithm might be parallelized to enable fast computation on large graphs [Cha16b].

Dempster-Shafer theory-based approach. The theory of belief functions (or Dempster-Shafer) is another approach for reasoning about uncertainty. In this approach, beliefs about possibilities (element of the domain of interest) are bounded by two values: belief and plausibility. The work in [Liu05] proposes to evaluate risks using Dempster-Shafer theory. It allows experts to independently assess risk likelihoods and how risks combine. Experts can also assess their uncertainty about estimated values. The experts' opinions are then combined to provide a unified assessment of the risks with their uncertainty margins.

Model uncertainty. Other approaches focus on 'design-time' uncertainties which enable reasoning about the presence or absence of model elements. In [Hor14], the i* goal models presented in [Hor16] are integrated with the MAVO framework [Sal12] to highlight uncertainties in the model structure in terms of absence and presence of model elements, to be resolved for meeting the desired goal satisfaction levels. In [Fra06], metrics on i* models are introduced to quantitatively reason about properties of an i* model. Another approach [Esp13] defines metrics over KAOS models to measure their complexity and completeness.

Comparing requirement uncertainty handling with our approach. Our approach differs from the preceding approaches in the following respects.

- None of the presented approaches support assessment by multiple experts. The reviewed approaches could benefit from integrating multiple sources of assessment to reduce uncertainty margins.

- Unlike [Bar10; Cha16a; Cha16b; Cheo9c; Whio9; Whi10], our approach is not based on fuzzy logic. Our approach might be applied there to elicit membership functions and regulate the fuzziness of goal satisfaction.

- The approach proposed in [Liu05], based on Dempster-Shafer theory, allows experts to disagree on both the estimated probabilities and the risk model structure; this is not supported in our framework. However, the Dempster-Shafer theory is less known by risk analysis experts; this might prevent the approach for being applied in real, industrial settings.

- With respect to FTA, as seen in Section 10.1, our approach exploits a goal refinement graph to assess the severity of risk consequences. The approach for computing satisfaction rates of high-level goals appears somewhat similar to the BDD-based propagation techniques used for computing the probability of a root event in [Bedo1].

- Unlike [Sab11], our approach is not limited to one triangular distribution per obstacle, and we support multiple experts. Our propagation procedure appears more efficient as it only requires a single propagation through the obstacle/goal model.

- As pointed in the previous section, the technique proposed in [Hea11; Leto4] appears more finer-grained at the cost of needing to elicit domain-specific propagation variables and equations.

- The approach in [Let14] might be adapted for selecting most critical obstacles.

10.5 Approaches for Controlling Requirement-Related Risks

This section discusses relevant work related to risks control and, specifically related to techniques presented in Chapter 5.

Exploring countermeasures. The work in [Sut98] proposes scenario-based heuristics for identifying risks together with generic requirements that could mitigate these.

In the context of KAOS, goal-oriented formal techniques are available such as a regression calculus [Lam00] or a learning-based approach [Alr16], see Chapter 2.

The work in [Bry06b; Bry06c] extends Secure Tropos, a Tropos variant with specific constructs dedicated to modeling of security-related concepts [Gio05b], to automatically generate alternative models. The problem of generating alternatives is formulated as a planning problem. Actions correspond to the application of model primitives, such as adding or removing links, whereas the planning objective is to satisfy predicates encoding 'correct' models. The planning actions do not introduce new intentional elements such as goals or actors. A planner then produces a plan that, once applied to the initial model, corresponds to an alternative design satisfying the initial goals. Alternative designs may then be evaluated to identify most promising ones [Bry07]. Alternative configurations may be generated at runtime when a reconfiguration is required [Bry06a]. Alternative designs that minimize the risk related to some specified high-level goals may be automatically generated by combining planning [Bry06b; Bry06c] with the quantitative Goal-Risk Framework [Asn06d].

Model revision with countermeasures. We are not aware of any work on the systematic integration of countermeasures in a requirement model based on a clear, precise semantics. The relevance and importance of default-based reasoning have been recognized in the context of elaborating requirements or specifications.

In [Zow97], a formal framework is proposed for reasoning about evolving requirements. The framework is based on belief revision and default theory; operators for adding new requirements and retracting requirements from a model are defined together with formal conditions for their valid application. The tracing of exceptional requirements is not discussed there.

The work in [Bro06] proposes an extension to the KAOS framework to specify changes in the goal's formal specifications based on A-LTL [Zha06]. A-LTL is an adaptation-based extension to linear temporal logic that introduces an extra *adapt* operator. Using the latter, the user can specify changes in specification over time.

In [Rya93], a specification is structured through axioms and *Overrides* relations. Such relations are derived from the structural decomposition of the system. Specific axioms predominate more general ones when a conflict occurs. This framework comes with formal foundations and well-defined procedures for identifying conflicts and predominance among axioms. These appear more oriented towards specification elaboration. In [Sch93], default specifications are introduced together with exceptions to increase the completeness of algebraic specifications.

Learning-based approach. A learning-based may also automatically revise goals that might be underspecified or partially wrong in order to resolve obstructions in a known domain [Alr16]. The obstructed goals are iteratively revised from traces generated by model checking an obstacle against a known domain; a learning-based revision task is then fed with those example traces. Changes are automatically propagated to other goals in the goal model so as to preserve model correctness.

Comparing risk control approach with our approach. Our approach from those previous efforts in the following directions.

- Unlike [Bro06; Rya93; Sch93; Zow97], our techniques operate at requirements level and benefit from the refinement structure of a goal model. This structure helps building a model where exception handling is integrated and required changes are propagated throughout the model.

- Unlike [Asn11; Bro06; Rya93; Sch93; Zow97], new requirements for a more robust system are incrementally integrated. Model updates are traceable back to the obstructed goals and their obstacles.

- The *But* relation introduced in [Sch93] somewhat corresponds to our *Except* relation.

- Our *Replace*, *RelaxedTo* and *Provided* annotations could be encoded into the specification using the *adapt* A-LTL operator as in [Bro06]. However, this would clutter the specification as exceptional cases would be mixed with the ideal one.

- The approach presented in [Alr16] could be integrated within ours in order to explore alternative resolutions to likely and critical obstacles.

At the programming level, aspects may be used for separating exception handling from normal code [Lip00]. At modeling level, the work in [Ali12] convincingly shows how aspects can be used for separating exceptional behaviors from normal ones. As an alternative to the approach advocated in this thesis, robustness aspects might be incorporated in a goal model by use of constructs similar to the ones presented in [Gil09; Mor13; Mus07]. We are not aware of aspect-oriented requirements engineering models or techniques that focus on exceptions handling as proposed in Chapter 5.

10.6 Requirements-Driven Runtime Adaptation

The literature on runtime adaptation of software systems is vast and covers a wide range of concerns. See [Che09b; DeL13; Jur15; Saw10; Yan14] for an overview of state-of-the-art techniques and approaches. We here discuss approaches closely related to the risk-driven runtime adaptation techniques presented in Chapter 7.

Early (KAOS-based) monitoring approaches. Following [Fic95], the work in [Fea98] proposes an approach that builds on KAOS to monitor the satisfaction of requirements at runtime. System parameters identified at RE-time from goal specification are monitored at runtime; when a violation is observed, a reconciliation tactic, identified at RE-time, is chosen and deployed.

ReqMon monitoring. The work in [Rob03; Rob05; Rob06] builds on [Fea98; Fic95] to monitor requirements and obstacles. Based on a goal/obstacle model, monitor specifications are extracted from obstacles assigned to monitoring agents. These specifications are turned automatically into executable monitors within the ReqMon infrastructure; specification patterns drive the transformation. The approach is aimed at raising alerts. The approach was extended to support goals formalized in UCM Object Constraint Language (OCL) [Rob07].

Adaptive goals. The work in [Bar10b] extends KAOS with adaptive goals. The latter prescribe a set of possible adaptations that can be performed when goals are violated. Adaptations include adding and removing goals, modifying goal specifications, etc.

Context-based approaches. The work in [Ali09; Ali10] details a tool-supported approach where goals, refinements, agent assignments and contributions to soft goals can be decorated with context conditions. Context conditions there are Boolean AND/OR formulas that are then monitored at runtime to switch to more appropriate alternative refinements. In [Ali11], a context is both a target state to be reached by a set of requirements and a current state that enables requirements (like a precondition) or activate requirements (like a trigger condition). Based on this notion of context, the approach in [Ali11] identifies 4 type of assumptions: *(a) activation assumptions* capture that a context requires a set of specific requirements; *(b) adaptability assumptions* capture that a context is needed by a requirement; *(c) refinement assumptions* capture that subgoals satisfy their parent goal; *(d) requirements achievement Assumptions* capture that a requirement is satisfied if a target variant is achieved. These assumptions are represented as a Boolean formula where atomic propositions are monitored at runtime. The system there decides what variant should be selected and deployed. Two metrics drive the decision process: *assumption validity* that captures the statistical evidence of a variant to satisfy the root goal; and *assumption criticality* that captures the extend to which a context condition prevents a variant from satisfying the root goal.

The work in [Gui15; Gui17] extends [Ali10]. It introduces *pragmatic goals* capturing variations in goal interpretations. It focuses on variations of quality variables such as time, errors, delays, etc. For example, consider the requirement '*Ambulances shall be on scene in a timely fashion*'; the interpretation of '*timely fashion*' would vary from one context (a simple dizziness for example) to another (a heart attack). The framework proposes to annotate goals with context conditions and their quality constraints. A pragmatic goal is satisfied if it satisfies the quality constraints of the current context. In the example above, the goal might be annotated with $time < 8m$ in the context $CriticalIncident$

and with $time < 14m$ otherwise. A bottom-up propagation algorithm is proposed to select the tasks that would guarantee the requirements given a current context [Gui15; Gui17]. If multiple plans are available, the best plan regarding the quality variable is selected.

Yet another work [Men16] combines [Dal13] and [Ali10]. Based on goal refinement annotations, a Discrete Time Markov Chain (DTMC) is built from the leaf goals to the root goals. The DTMC for a leaf goal is a simple DTMC with success and failure probabilities. The DTMC for a non-leaf goal is obtained by combining the DTMC of the subgoals according to the annotation (parallel combination, sequential combinations, number of retries, etc.) Once the DTMC for the root goal is obtained, a probabilistic model checker provides an estimate of the probability of satisfying the root goal; at runtime the model checker computes the probability to satisfy $\Diamond(\wedge_i G_i)$ where G_i are leaf goals. To enable computation, a parametric model checker computes a parametric expression representing the probability to satisfy the root goal ; the parameters are the values of the success and failure probabilities for the leaf goals. The impact of context on the failure probabilities is briefly discussed in [Men14]. Note that in this context the DTMC does not model the behavior of the system.

DTMC-based approaches. The approach in [Epi09] learns transition probabilities in a Discrete Time Markov Chain (DTMC) modelling the system behavior. At runtime, transitions between the nodes of the DTMC are observed, and probabilities for these transitions are computed. These probabilities feed a Bayesian inference to learn revised probabilities taking into account both the previously observed probabilities and the last observed one. A probabilistic model checker determines whether the reliability and performance are satisfied. [Ghe09] builds upon [Epi09] to detect and predict requirements failures.

In [Ghe13], the software is modeled as a dynamic product line (DSPL); different configurations define the possible space for adaptations. From DSPL models, a parametric Markov Chain is generated. An efficient runtime verification technique drives the selection of most appropriate configurations at system runtime [Fil16].

An approach for filtering the monitored probabilities is described in [Fil15a] to reduce the noise of observations while maintaining an accurate tracking for the real probabilities.

Control-theoretic approach. In [Fil11; Fil14; Fil15b], a control-theoretical approach is taken to automatically generate controllers for self-adaptive systems. The user provides a running software and a parameter that impacts some non-functional requirements. The approach first learns a linear model binding the parameter to the non-functional requirements; a controller is then generated to tune the parameter according to the non-functional requirement. Multiple non-functional requirements and multiple parameters may be integrated in the approach [Fil15c].

Awareness and Evolution Requirements. *Awareness Requirements* (*AwReq*) [Sou11b] reason about requirement satisfaction. These are meta-requirements about the system requirements that are monitored at runtime using ReqMon [Robo6]. *Evolution Requirements* (*EvoReq*) enable reasoning about the evolution of requirements [Sou12]. They are meta-requirements contraining how system requirements should change when required. *AwReq* and *EvoReq* complement each other to specify a feedback loop for self-adaptive systems [Sou13]. *AwReq* indicate situations in which an adaptation is required whereas *EvoReq* prescribe what adaptation should be deployed. How awareness requirements interact in a control-theoretic approach for self-adaptive systems is discussed in [Sou11a].

Tropos for adaptive systems. The work in [Moro8; Mor17b] extends the Tropos framework for automated systems along three dimensions: a goal dimension, an environment dimension, and a failure dimension. In the framework, goals are decorated with conditions formally specifying the goal; conditions includes *creation condition*, *pre-conditions*, *target conditions*, etc. The conditions are expressed in a non-temporal propositional logic. The failure model captures exceptional situations and possible mitigations. A failure model includes *Failures* capturing states where goals are not satisfied; *Error* events, formalised as Boolean conditions, lead to failures; and *Recovery actions* anticipate, prevent and recover from errors. Reasoning about errors is not quantitative; no dedicated algorithm is available for selecting most appropriate recovery actions.

Olympus. In [Ram11a] a genetic algorithm is described for choosing the best configuration for an overlay network. Sequences of actions to change a software system from one configuration to another may then be generated [Ram10]. In [Ram11b] utility functions are generated from a goal model with RELAXed requirements to enable the monitoring of goals. Depending on the

type of goal, the technique automatically generates a state-based, metric-based or fuzzy logic-based utility function that, fed with monitored values, provides an estimate for goal satisfaction. Utility scores are then propagated in the goal model from monitored goals to root goals. The work in [Che13] illustrates how these techniques intertwine to enable self-adaptation of software systems.

Self-adaptation for service-based systems. The approach in [Ori12; Qur09; Qur10a; Qur10b; Qur11a; Qur11b] introduces a framework for requirements on self-adaptive service-based systems. At design-time, a forest of goal models is elicited to cover a wide variety of refinement alternatives. At runtime, the best alternatives are selected to satisfy high-level requirements and maximize user preferences. The latter are determined explicitly by the user or detected by the adaptive software.

Surprise-based approach. A preliminary approach is outlined in [Ben14; Ben15] where the need for adaptation is inferred from surprise. A surprise is defined as '*how much monitored information differs from previously monitored information*'.

Comparing requirement-driven runtime adaptation with our approach. The preceding efforts may be related with ours as follows.

- Our approach builds on a precise semantics defined in terms of behaviors. This is particularly important for monitoring; otherwise, the results are somewhat limited by lack of precise characterization in terms of observed states and behaviors. Unlike [Ali10; Gui17; Men16; Ram11a; Sou13], we are able to monitor behaviors and not just state predicates. We can therefore detect complex behaviors (such as P occurs between S and Q) that appear to be hard to model with other approaches.

- Unlike some of the preceding approaches [Fil16; Ghe13; Ori12; Rob03; Rob05; Rob06], we propagate monitored satisfaction rates through goal and obstacle refinement trees. This enables computation of the satisfaction rates of goals that are not directly monitorable. Our risk-driven adaptation thus benefits from this goal/obstacle refinement structure.

- Unlike most of the preceding approaches, our approach does not focus on selecting alternatives or configuration parameters but rather on selecting appropriate countermeasures to risks. The approach in [Rob07] focuses on raising alerts. Our approach appears to be closer to [Bar10] as the goal model itself might be adapted.

- Our approach shares similarities with [Fil16; Ghe09; Ghe13] in terms of probabilistic formalization and use of formal verification techniques. It does however not require building an explicit behavior model. The approach in [Men16] also relies on probabilistic model checking, but the Markov Chain there does not model the behavior of the system.

- The focus in [Sou11b; Sou12] is on meta-requirements about other requirements. Our approach is restricted to an implicit meta-requirement (the computed satisfaction rates remain above required satisfaction rates) and an implicit evolution meta-requirement (most appropriate countermeasures are selected.)

- Unlike [Mor08; Ram11b; Rob07], our monitoring procedure does not rely on specification patterns or goal types. We also support temporal operators such as $\mathcal{U}$ or $\mathcal{W}$. The monitored leaf obstacles are formalized in Linear Temporal Logic.

- Unlike [Ori12; Qur09; Qur10a; Qur10b; Qur11a; Qur11b], our approach is not specific to self-adaptive service-based systems. The type of adaptation is therefore different.

- Unlike [Ali09; Ali10; Mor17b; Sou13], our approach is quantitative. This enables a different, finer-grained set of techniques.

Beyond those frameworks for software adaptation, other monitoring techniques could relate to our approach. Monitoring techniques such as [Bar04; Sam05; Tha05] focus on failure detection; our focus in on probabilistic assessment. In [Lee07; Sam05], the observed trace is divided into smaller samples so that properties can be checked on each sample; statistical reasoning is then applied to extract a probability. Our approach proposes an alternative without

sampling the observed behavior into fixed-size samples. These approaches do not incorporate requirements or risk models; they are restricted to monitoring LTL assertions.

Our approach could benefit from [Fil15a] to smooth out monitored satisfaction rates. The approach drafted in [Ben14; Ben15] could be applied to our monitored satisfaction rates to detect critical changes. This, however, needs to be further studied as no precise semantics appears to be available.

The problem of deploying countermeasures and computing sequence of actions required to adapt the running software system turns to be more complex than the simplistic approach presented in Section 7.5. Approaches such as [Ram10] could be integrated to take the complexity of the problem into account.

Other approaches to runtime adaptation, such as Rainbow [Gar03; Gar04; Gar09], focus on architectural models. These approaches tend to consider low-level requirements; they do not benefit from refinement structures enabling more complex adaptations; A detailed comparison between a requirements-based approach [Sou13] and the architecture-based approach Rainbow [Gar04] is available in [Ang13]. Other approaches such as [Che14; Kra07] mix both requirements and architectural models to propose better adaptations [Nus01; Saw10]. In [Cas11] a different problem of localizing faulty components by monitoring a software system is considered

Our study of the relevant literature led us to some further connections.

- The approach in [Asn11] could be combined with their previous effort [Asn06d] to achieve risk-driven software adaptation in a similar spirit to our techniques.

- The approach proposed in [Dal13] appears to be similar to Dynamic Fault Trees with temporal requirements as described in [Rui15].

- The approach proposed in [Men16] to compute the probability to satisfy a root goal based on bottom-up propagation through Markov Chains appears to be similar to the computation procedure used in Dynamic Fault Trees [Dug92; Rao09].

10.7 Summary

This section discussed the relevant work related to our approach. To sum up, our results are mainly distinguished from this work in the following aspects: our quantitative risk analysis framework is anchored on requirements; the risks are identified, assessed and controlled with regard to these requirements. The tree structure in the goal/obstacle models enables finer-grained obstacles to be assessed by experts and monitored at runtime while enabling obstacles to be prioritized and best countermeasures to be selected with respect to high-level objectives. The semantics of our probabilistic goals and obstacles is precisely defined in terms of observable states and behaviors. Lastly, multiple cycles of risk analysis can be performed where risks to countermeasures are identified, assessed and controlled.

Chapter 11

Conclusion

This chapter summarizes the contributions of this thesis, discusses the strengths and limitations of our results, and discusses open issues and future research directions.

11.1 Summary of contributions

While engineering requirements, obstacles preventing the satisfaction of system objectives need be identified, assessed and controlled to produce more adequate, complete, consistent, and precise requirements. The model-based techniques presented in the thesis are intended to fill current gaps in the *obstacle assessment* and *control* steps by proposing a quantitative extension to an existing goal-oriented RE framework to integrate probabilistic goals and obstacles.

For *obstacle assessment*, the thesis proposes a simple yet effective quantitative model-based technique. The KAOS framework is extended with a probabilistic layer allowing behavioral goals to be specified in terms of their *required* and *computed* satisfaction rates. This layer support probabilistic goals and probabilistic obstacles. Their satisfaction rate is precisely defined in terms of state probabilities; the latter capture the probability that the behaviors rooted in a system state satisfy a given assertion. The refinement structure and satisfaction arguments for non-probabilistic goals and obstacles are generalized accordingly.

Obstacles are assessed with respect to their likelihood and the severity of their consequences in terms of impact on the goal model. The satisfaction rate of non-leaf obstacles is determined by up-propagation in the corresponding obstacle refinement trees. The criticality of obstacles, in terms of severity of their consequences, is systematically determined by up-propagating the satisfaction rate of obstacles along the corresponding goal refinement trees. The

thesis proposes two propagation algorithms for computing the satisfaction rate of high-level goals from the estimated satisfaction rate of leaf obstacles. The *BDD-based* propagation procedure is efficient when repeated propagations are required whereas the *pattern-based* propagation procedure is finer-grained at an increased specification cost. Most likely and critical obstacle combinations are then highlighted to guide the selection of countermeasures in the next *control* step.

For *obstacle control* step, the thesis proposes techniques for selecting most appropriate countermeasures that guarantee the satisfaction of the *required* satisfaction rate of high-level goals in the goal refinement graph while minimizing the cost of obstacle resolution. An integration mechanism is defined as a model transformation that guarantees progress towards a more complete goal model, minimal changes to the original model, and preservation of the correctness of the refinements. Anchor goals define where the countermeasures goals should be integrated. Countermeasures then enrich the original goal model in a modular way with dedicated exception constructs. Refactoring operators allow the analyst to attach and detach those exceptions.

Those techniques for *obstacle assessment* and *obstacle control* are extended to cope with knowledge uncertainty about satisfaction rates of probabilistic goals and their obstructing obstacles. The uncertainty margins on satisfaction rate estimates are captured as probability distributions. The impact of such uncertainty on the criticality of obstacles is quantified via two metrics, namely, the *goal violation uncertainty* and the *uncertainty spread*. The satisfaction rate of non-leaf obstacles and the satisfaction rate of goals, together with their uncertainty margins, are computed by repeated up-propagation through the goal/obstacle models. As a result, the critical obstacles with problematic uncertainty margins are highlighted. These problematic uncertainties may be reduced by combining multiple sources of estimates. Appropriate countermeasures may then be selected so as to guarantee the *required* satisfaction rate of high-level goals up to some uncertainty threshold.

Lastly, the thesis proposes an obstacle-driven runtime system adaptation approach aimed at increasing the actual satisfaction rate of probabilistic system goals at runtime. Leaf obstacles together with their criticality are monitored

at runtime to let the system dynamically switch to more appropriate countermeasures goals. The proposed monitoring technique extends an available non-probabilistic approach [Bau11] to support the monitoring of probabilistic assertions. Most appropriate countermeasures are selected on-the-fly to increase the satisfaction rate of the considered high-level goals under current conditions. Selected countermeasures are then deployed in the running system.

All techniques proposed in the thesis are supported by a set of tools enabling their application on real case studies. The tools and techniques were evaluated on three realistic case studies: a car pooling system, an industrial yoke lifting system for proton therapy and an urban ambulance dispatching system.

11.2 Strengths and limitations

The main *strengths* of our techniques for obstacle assessment and control can be summarized as follows.

- *Probabilistic goals enable the handling of properties required to hold in X% of cases.* Probabilistic goals enable such requirements to be included as first-class citizens together with non-probabilistic ones, within the same goal-oriented RE framework.

- *Probabilistic obstacles enable precise risk assessment.* Probabilistic obstacles obstructing non-probabilistic goals or probabilistc ones allow for a precise assessment of the likelihood of requirements-related risks in the RE process.

- *Probabilistic goals and obstacles have a precise semantics anchored on real-world phenomena.* This semantics, in terms of behaviors refering to domain-specific phenomena, enables domain experts to estimate and interpret satisfaction rates of probabilistic obstacles and goals in domain-specific terms. The precise semantics also enables the monitoring of satisfaction rates at runtime (as seen in Chapter 7.)

- *Refinement trees on probabilistic obstacles provide a structure for the assessment of risk likelihood.* The satisfaction rate of non-leaf obstacles is systematically computed from the estimated satisfaction rates for leaf obstacles by up-propagation through the obstacle refinement trees. Only leaf obstacles need be estimated by domain experts.

- *Refinement trees on probabilistic goal provide a structure for the assessment of obstacle criticality.* The consequences of an obstacle are captured in terms of decrease in the satisfaction rate of goals in the goal model. The severity of such consequences is systematically determined by up-propagation of obstacle satisfaction rates through goal refinement trees. An obstacle is critical if the satisfaction rate of the considered goals fall below their required satisfaction rate.

- *The control of probabilistic obstacles guarantees progress towards complete models.* Valid countermeasures, as defined in Section 5.1, guarantee that the satisfaction rate of high-level goals increases when properly integrated in the original goal model. Integrating increasingly effective countermeasures results in high-level goals being increasingly satisfied.

- *Probabilistic goals and obstacles enable cost-effective selections of appropriate countermeasures.* Selecting most appropriate countermeasures amounts to solving an optimization problem where the satisfaction rate of high-level goals is maximized while the resolution cost is minimized.

- *The integration of countermeasures to probabilistic obstacles preserves the correctness of the goal refinement structure.* The propagation of required specification changes through the obstacle/goal models guarantees that the goal refinement graph remains complete, consistent and minimal (as seen in Section 5.4.2).

- *The integration in the goal model of countermeasures to probabilistic obstacles preserves all normal behaviors.* The two integration schemas introduced in Section 5.4.1 guarantee that the goals not affected by the resolved obstacles are not modified. These schemas either replace anchor goals or refine them, leaving the rest of the goal model intact. Normal behaviors not impacted by the obstacle are thereby preserved.

- *The proposed exception handling mechanism separates ideal from exceptional cases.* The *Except*, *Provided* and *RelaxedTo* constructs presented in Section 5.4.3 separate the specification of ideal cases from exceptional ones. The specification of ideal goals is thereby not cluttered with conditions related to exceptional cases.

- *The proposed exception mechanism enables incremental integration of countermeasures in the goal model.* Refactoring operators allows the analyst to attach and detach exceptions, thereby deferring the decision of what is exceptional from what is not to later stages of the requirement process.

- *The proposed exception handling mechanism mitigates the combinatorial explosion of countermeasures.* The absence of such mechanism impact both the anchor goal specification and its refinement structure. Each countermeasure goal would need be refined by taking other countermeasures into account. This would lead to a combinatorial blow-up of cases. Thanks to the *Except* and *Provided* constructs, the original specification of the anchor goal and its refinement structure are preserved.

- *The uncertainty about probabilistic goals/obstacles satisfaction is explicitely taken into account.* It allows domain experts to express uncertainty margins about estimates through probability distributions.

- *Two metrics allow for precisely measuring problematic uncertainties about goal satisfaction rates.* When applied to the probability distributions obtained by up-propagation to high-level goals in the goal graph, the metrics allow critical leaf obstacles with most problematic uncertainty margins to be highlighted.

- *Our framework supports multiple sources of estimation in order to reduce uncertainties.* Multiple sources for estimates are recognized to produce more accurate estimates [Bed01]. Problematic uncertainty margins are reduced here by combining such multiple sources.

- *The satisfaction rate of probabilistic goals and obstacles is monitored at system runtime.* Monitors, built at RE time from the formal specification of leaf obstacles, are used to determine the actual satisfaction rate of leaf

obstacles at runtime. The latter are propagated through the obstacle/goal model up to the system's high-level goals in order to determine the actual criticality of those leaf obstacles.

- *Obstacle-driven runtime adaptations ensure that only those adaptation that are really required by the system specifications are performed.* Our approach selects most appropriate countermeasures. The latter maximize the satisfaction rate of high-level goals while minimizing the resolution cost. This guarantees that the required satisfaction rate of high-level goals remains satisfied when the obstacle satisfaction rates are changing.

- *The monitoring of probabilistic goals and obstacles does not require explicit behavior models.* Building a consistent and complete behavior model for large distributed systems with many complex states and parallelism is often quite challenging. Our monitoring technique does not require such model to be provided.

- *Our approach supports traceability to system's goals.* Decision criteria for countermeasure selection and system adaptation are traceable to system objectives. The exception handling constructs document why goal revisions are required. Lastly, why such or such monitored information is required is documented as the predicates to be monitored are infered from the formal specification of the leaf obstacles to be monitored.

The current *limitations* of our techniques pave the way for future work.

- *Lack of support for probabilistic goal elicitation and refinement.* Unlike the approach presented in [Leto4], there is currently no support for building probabilistic goal models. In [Leto4], domain-specific equations may guide the elicitation process; the equations components may correspond to different subgoals.

- *Lack of support for identification of countermeasures to probabilistic obstacles.* Currently, the countermeasures are identified using a variety of available resolution tactics [Lam09] or a learning-based approach based on observed behaviors [Alr16]. One might think of exploration techniques specific to probabilistic obstacles.

- *Countermeasures are assumed to be identified at RE time.* There is currently no support for identifying or learning countermeasures to probabilistic obstacles at runtime. The proposed techniques for runtime system adaptation rely on a set of countermeasures identified at RE time.

- *Lack of support for assessment and control of agents assigned to probabilistic goals.* Some agents might be more reliable than others and different agent instances might exhibit different characteristics. One resolution tactic transfers responsibilities from a less reliable agent to a more reliable one [Lam09]; such transfer is not considered by the techniques presented in the thesis. It is currently not clear how agent reliability can be precisely defined at the requirements level and how it impacts the obstacle assessment and control.

- *Optimization techniques are limited to exhaustive search.* Optimization techniques for highlighting the most critical obstacle combinations and identifying most appropriate countermeasures are not really worked out and compared with the brute force exhaustive search used in the case-studies.

- *The obstacle control technique overlooks soft goal contributions.* The proposed technique for obstacle control does not take contributions to soft goals into account. Trade-offs among competing soft goals are not considered.

- *The proposed techniques for obstacle assessment and control ignore interdependencies among obstacles and countermeasures.* A specific occurrence of an obstacle might increase the satisfaction rate of another obstacle; a specific selection of countermeasures might increase the satisfaction rate of a set of obstacles; satisfaction rates might increase or decrease over time depending on whether a countemeasure is selected, and so forth. Unlike Dynamic Fault Trees [Rui15], the proposed techniques here do not currently support such dynamic views.

11.3 Open issues and perspectives

This section discusses currently open issues and perspectives for future work.

Obstacle and countermeasure identification. Incompleteness detection in goal and obstacle models at runtime could improve the obstacle identification step. The monitoring of high-level goals and obstacles might highlight obstacles that were overlooked. This might be achieved by comparing the computed goal satisfaction rates obtained by up-propagation with the monitored ones. This appears to be a promising approach for highlighting behavior traces exposing unknown obstacles or countermeasures. Such work could be integrated with learning obstacles and countermeasures from observed traces [Alr12; Alr16] or from an executable specification [Ram12a].

It is common for risk analysis to be performed once during system analysis without being updated as the software evolves. The risk analysis may quickly become obsolete, incomplete and inadequate. Previous research efforts show that missing requirements may be detected during architectural design [Nus01] or during testing [Lut03]. Integrating architectural design, testing and risk analysis could continuously ensure that the software system covers the identified risks, and that the risk analysis remains adequate, precise, complete, and consistent with regard to the evolving system. Requirements-driven [Raj08; Wha06] and risk-driven testing [Erd14; Klo11; Red05] approaches appear in particular to be a promising direction.

Support for other type of satisfaction rates. The thesis focussed on probabilistic constraints requiring some target condition to hold in X% of cases; the satisfaction rate of a goal was accordingly defined in terms of state probabilities and ratios over behaviors. Alternative notions of satisfaction rate, not based on lowest/highest state probabilities, would be worth considering. This would support other classes of properties of interest—such as a target condition should hold in 95% of the time, or in 95% of cases in some specific context. One might envision a constraint specification language whose 'compilation' would generate parameter instantiations in generalized model up-propagation equations according to the specific notion of satisfaction rate subsumed by the input specification. Such generalization of the quantitative framework in the thesis might reuse our mechanisms for model up-propagation, countermeasure

selection and integration, knowledge uncertainty management, and runtime reconfiguration based on the corresponding notion of satisfaction rate. Such generalized framework might perhaps integrate domain-specific variables, such as in [Leto4].

Satisfaction rates might be time-dependent, as suggested in Chapter 7. Expressing well-known and widely used metrics such as Mean Time Between Failure (MTBF), Mean Time Between Repair (MTBR), or Mean Down Time (MDT) within a generalized framework would obviously extend the applicability of our techniques. A comparative assessment of those metrics and others used in risk analysis, within such framework, would provide an added value to it.

Dynamic obstacle models. Some of the techniques behind Dynamic Fault Trees [Rui15] might improve the expressiveness and accuracy of probabilistic goal/obstacle models. How probabilistic goals and obstacles with a high-level of interdependency may be elicited, modeled, and analyzed remain an open question to us.

Obstacle-driven design. Our approach is focused on determining whether the required satisfaction rate of high-level goals are met, given the estimated satisfaction rates of leaf obstacles. Another type of analysis would include the following reverse question: what are the acceptable satisfaction rates of low-level obstacles given a required satisfaction rate of high-level goals? Parametric model checking might be used to answer questions on the low-level obstacles, such as '*how many ambulances are required to guarantee an obstacle 5% satisfaction rate?*'. Combining such work in the reverse direction with our proposed quantitative framework might answer questions on high-level goals such as '*how many ambulances are required to guarantee a 95% required satisfaction rate of the ambulance to be timely on scene?*'.

Finer-grained obstacle assessment and control. The following work directions might improve obstacle assessment and control:

- The satisfaction rate of a goal (respectively, an obstacle) is currently defined as the highest (respectively, the lowest) state probability that the specification is satisfied. Considering instead highest, lowest and average

state probabilities might provide finer-grained analysis of the system internals. States might also be weighted according the time spent in the state. In particular, this extra information could be used to make better decisions about which software adaptations to performed at runtime—e.g., it might be worthless to trigger an adaptation if the state with the highest state probability to satisfy an obstacle condition is not visited often.

- How positive and negative interactions between countermeasure goals can be quantitatively taken into account appears to be worth investigating and might improve the quality of the countermeasure selections.

- Domain-specific variables and equations were used to provide fine-grained selection of alternative designs, e.g. [Bus17; Hea11; Let04]. These techniques do not consider obstacles; integrating such techniques within our framework for risk-driven framework might be worth investigating.

- Our approach selects countermeasures so as to minimize some cost. Design decisions often result from fine-grained trade-off analysis involving multiple competing criteria [Che06]. Refined criteria taking soft goals or domain-specific variables into account could lead to better countermeasure selections.

- A known drawback of Monte Carlo simulations is that the number of simulations required for obtaining accurate estimations grows in the presence of rare events [Jeg13; Rui17]. Specific propagation procedures could be developed to better support rare obstacles.

Security risk analysis. How the proposed approach transpose to anti-goals [Lam03], a goal-oriented form of security risk analysis, is an open question to us; *anti-goals* are a special kind of obstacle caused by malicious agent [Lam03]. Moreover, an in-depth comparison with existing security-based approaches such as *attack trees* and *threat trees* [Hel02; Sch11; Sch99], *UMLSec* [Jur01; Jur02], *SecureUML* [Lod02], *Abuse Case* [McD99], or *Misuse Case* [Sin05] could exhibit interesting technique transfer from one community to the other.

Obstacle-driven adaptation at runtime. The proposed approach for runtime adaptation might be improved in the following directions.

- The obstacle-driven adaptation technique relies on monitoring satisfaction rates. However, monitoring is not always possible or desirable, for example, in ultra-low power systems. On-the-fly adaptations are not necessarily desirable in very critical safety applications where runtime adaptation may make the certification of such systems hard to obtain. An explicit modeling of the evolution of the satisfaction rates over time would enable optimal countermeasure selections to be computed a priori at RE time for software systems where monitoring is not possible or desirable.

- Our runtime monitoring technique is limited to probabilistic LTL assertions that are propositional. It should be extended to support first-order formulas. This might be achieved by replacing the LTL_3 monitoring technique from [Bau11] by other monitoring techniques such as [Bas11] that support richer logics.

- Actual satisfaction rates could abruptly change over time; the monitored satisfaction rates however tend to slowly reflect abrupt changes. Algorithms for detecting such abrupt changes, e.g., [Epi10; Fil14; Liu13] and filtering as proposed in [Fil15a] could improve the accuracy of monitored satisfaction rates.

- The question of 'when' precisely to adapt remains an open issue. If deploying an adaptation is cheaper than the penalty of a too frequent violation, it might be preferable to adapt before violation occurs. However, some adaptations are costly and should not be performed unless the violation lasts enough for the penalty to exceed the cost. In addition, software adaptations exhibit latencies that should be taken into account at RE time [Cam14; Cam16]. Prediction approaches, such as [Mor16; Pol07], appear to be a promising direction for more cost-effective adaptations.

Software is increasingly present in our daily life. Critical aspect of our health, economy, and environment are therefore increasingly impacted by the failure or success of software. The way software is designed and operated significantly influences the quality of our life. Among the challenges raised by this observation, we believe that bringing high-quality requirements much closer to the running systems will improve the analysis, adaptation, and operation of such critical

systems. The satisfaction of high-level requirements considered as first-class citizens might drive the detection of negative trends, faulty patterns, intentional or non-intentional misuse, and so forth. It might anticipate the occurence of severe problems. It might also improve the quality of the optimizations and trade-offs performed at system runtime. Closing such gap might enable a collaborative integration of operators in charge of running these software. The same way as air controllers delegate airplane routing to software, operators might interactively collaborate with adaptation mechanisms to delegate the satisfaction of some high-level requirements while manually ensuring others. In such scenario the software itself should have a comprehensive knowledge of all what could go wrong in order to diagnose and repair human errors and its own errors.

References

[Akh01] F. Akhmedjanov, *Reliability databases: state-of-the-art and perspectives*, Forsknings-center Risoe, 2001.

[Ali09] R. Ali, F. Dalpiaz, and P. Giorgini, "A Goal Modeling Framework for Self-contextualizable Software", *Proc. of EBPISM'09—the 10th International Workshop on Enterprise, Business-Process and Information Systems Modeling*, 2009.

[Ali10] R. Ali, F. Dalpiaz, and P. Giorgini, "A goal-based framework for contextual requirements modeling and analysis", *Requirements Engineering*, Vol. 15, No. 4, 2010, 439–458.

[Ali11] R. Ali, F. Dalpiaz, P. Giorgini, and V.E.S. Souza, "Requirements Evolution: From Assumptions to Reality", *Proc. of EBPISM'11—the 12th International Workshop on Enterprise, Business-Process and Information Systems Modeling*, 2011.

[Ali12] S. Ali, L.C. Briand, and H. Hemmati, "Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems", *Software and Systems Modeling*, 2012, 1–38.

[Alr12] D. Alrajeh, J. Kramer, A. Van Lamsweerde, A. Russo, and S. Uchitel, "Generating obstacle conditions for requirements completeness", *Proc. of ICSE'12—the 34th International Conference on Software Engineering*, IEEE, 2012.

[Alr16] D. Alrajeh, A. van Lamsweerde, J. Kramer, A. Russo, and S. Uchitel, "Risk-driven revision of requirements models", *Proc. of ICSE'16—the 38th International Conference on Software Engineering*, ACM, 2016.

[Amy10] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, and E. Yu, "Evaluating goal models within the goal-oriented requirement language", *International Journal of Intelligent Systems*, Vol. 25, No. 8, 2010, 841–877.

[Ang13] K. Angelopoulos, V.E.S. Souza, and J. Pimentel, "Requirements and architectural approaches to adaptive software systems: A comparative study", *Proc. of SEAMS'13—the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, IEEE, 2013.

[Ant98] A.I. Antón and C. Potts, "The use of goals to surface requirements for evolving systems", *Proc. of ICSE'98—the 20th International Conference on Software Engineering*, IEEE, 1998.

[Asn06a] Y. Asnar, P. Giorgini, and J. Mylopoulos, *Risk modelling and reasoning in goal models*, Technical report, University of Trento, 2006.

[Asno6b] Y. Asnar and P. Giorgini, "Modelling risk and identifying countermeasure in organizations", *Lecture Notes in Computer Science*, Vol. 4347, 2006, 55-66.

[Asno6d] Y. Asnar, V. Bryl, and P. Giorgini, "Using risk analysis to evaluate design alternatives", *Proc. of AOSE'06—Agent-Oriented Software Engineering VII*, Springer, 2006.

[Asno7a] Y. Asnar, P. Giorgini, F. Massacci, and N. Zannone, "From trust to dependability through risk analysis", *Proc. of ARES'07—Second International Conference on Availability, Reliability and Security*, IEEE, 2007.

[Asno7b] Y. Asnar and P. Giorgini, *Analyzing Risk-Countermeasure in Organizations: a Quantitative Approach*, Technical report, University of Trento, 2007.

[Asn11] Y. Asnar, P. Giorgini, and J. Mylopoulos, "Goal-driven risk assessment in requirements engineering", *Requirements Engineering*, Vol. 16, No. 2, 2011, 101–116.

[Ayy14] B.M. Ayyub, *Risk analysis in engineering and economics*, CRC Press, 2014.

[Bai08] C. Baier and JP. Katoen, *Principles of model checking*, MIT press Cambridge, 2008.

[Bar04] H. Barringer, A. Goldberg, K. Havelund, and K. Sen, "Rule-based runtime verification", *Proc. of VMCAI'04—Verification, Model Checking, and Abstract Interpretation*, Springer, 2004.

[Bar10] L. Baresi, L. Pasquale, and P. Spoletini, "Fuzzy goals for requirements-driven adaptation", *Proc. of RE'10—the 18th International Requirements Engineering Conference*, IEEE, 2010.

[Bar10b] L. Baresi and L. Pasquale, "Live goals for adaptive service compositions", *Proc. of SEAMS'10—Workshop on Software Engineering for Adaptive and Self-Managing Systems*, ACM, 2010.

[Bas11] D.A. Basin, M. Harvan, F. Klaedtke, and E. Zalinescu, "MONPOLY: Monitoring Usage-Control Policies", *Proc. of RV'11—the 2nd International Conference on Runtime Verification*, 2011.

[Bau11] A. Bauer, M. Leucker, and C. Schallhart, "Runtime verification for LTL and TLTL", *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 20, No. 4, 2011.

[BDNET] "Benchmark.NET", https://github.com/dotnet/BenchmarkDotNet.

[Bed01] T. Bedford and R. Cooke, *Probabilistic Risk Analysis: Foundations and Methods*, Cambridge University Press, 2001.

[Ben07] N. Bencomo, G. Blair, and R. France, "Model-driven software adaptation", *Proc. of ECOOP'07—European Conference on Object-Oriented Programming*, Springer, 2007.

[Ben08] N. Bencomo, P. Grace, C. Flores, D. Hughes, and G. Blair, "Genie: Supporting the model driven development of reflective, component-based adaptive systems", *Proc. of ICSE'08—the 30th International Conference on Software engineering*, ACM, 2008.

[Ben10] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier, "Requirements reflection: requirements as runtime entities", *Proc. of ICSE'10—the 32nd International Conference on Software Engineering*, IEEE, 2010.

[Ben14] N. Bencomo and A. Belaggoun, "A world full of surprises: Bayesian theory of surprise to quantify degrees of uncertainty", *Proc. of ICSE'14 (Companion)—the 36th International Conference on Software Engineering*, ACM, 2014.

[Ben15] N. Bencomo, "Quantun: Quantification of uncertainty for the reassessment of requirements", *Proc. of RE'15—the 23rd International Requirements Engineering Conference*, IEEE, 2015.

[Boe91] B.W. Boehm, "Software risk management: principles and practices", *IEEE Software*, Vol. 8, No. 1, 1991, 32-41.

[Bor01] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator", *Proc. of ICDE'01—the 17th International Conference on Data Engineering*, IEEE, 2001.

[Bra07a] F. den Braber, I. Hogganvik, M. Lund, K. Stølen, and F. Vraalsen, "Model-based security analysis in seven steps–a guided tour to the CORAS method", *BT Technology Journal*, Vol. 25, No. 1, 2007, 101–117.

[Bra07b] G. Brændeland, H. Dahl, and K. Stølen, *A modular approach to the modelling and analysis of risk scenarios with mutual dependencies*, Technical report, SINTEF, 2008.

[Bra12] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice", *Synthesis Lectures on Software Engineering*, Vol. 1, No. 1, 2012.

[Bre04] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology", *Autonomous Agents and Multi-Agent Systems*, Vol. 8, No. 3, 2004, 203–236.

[Bro06] G. Brown, B.H. Cheng, H. Goldsby, and J. Zhang, "Goal-oriented specification of adaptation requirements engineering in adaptive systems", *Proc. of SEAMS'16—the 11th International Workshop on Self-Adaptation and Self-Managing Systems*, ACM, 2006.

[Bry06a] V. Bryl and P. Giorgini, *Self-configuring socio-technical systems: Redesign at runtime*, Technical report, University of Trento, 2006.

[Bry06b] V. Bryl, F. Massacci, J. Mylopoulos, and N. Zannone, "Designing security requirements models through planning", *Proc. of CAiSE'06—the 18th International Conference on Advanced Information Systems Engineering*, Springer, 2006.

[Bry06c] V. Bryl, P. Giorgini, and J. Mylopoulos, "Designing cooperative IS: Exploring and evaluating alternatives", *Proc. of OTM'06—On the Move to Meaningful Internet Systems: CoopIS, DOA, GADA, and ODBASE*, 2006.

[Bry07] V. Bryl, P. Giorgini, and J. Mylopoulos, "Supporting Requirements Analysis in Tropos: A Planning-Based Approach.", *Proc. of PRIMA'07—the 10th Pacific Rim International Conference on Multi-Agents*, Springer, 2007.

[Bur93] D. Burns and R. Pitblado, "A modified HAZOP methodology for safety critical system assessment", *Directions in Safety-critical Systems*, Springer, 1993, 232-245.

[Bus17] S.A. Busari and E. Letier, "Radar: A lightweight tool for requirements and architecture decision analysis", *Proc. of ICSE'17—the 39th International Conference on Software Engineering*, IEEE, 2017.

[Cai12] A. Cailliau and A. van Lamsweerde, "A probabilistic framework for goal-oriented risk analysis", *Proc. of RE'12—the 20th International Requirements Engineering Conference*, IEEE, 2012.

[Cai13a] A. Cailliau and A. van Lamsweerde, "Assessing requirements-related risks through probabilistic goals and obstacles", *Requirements Engineering*, Vol. 18, No. 2, 2013, 129-146.

[Cai13b] A. Cailliau, C. Damas, B. Lambeau, and A. van Lamsweerde, "Modeling car crash management with KAOS", *Proc. of CMA@RE—International Comparing Requirements Modeling Approaches Workshop*, IEEE, 2013.

[Cai14] A. Cailliau and A. van Lamsweerde, "Integrating exception handling in goal models", *Proc. of RE'14—the 22d International Requirements Engineering Conference*, IEEE, 2014.

[Cai15] A. Cailliau and A. van Lamsweerde, "Handling knowledge uncertainty in risk-based requirements engineering", *Proc. of RE'15—the 23rd International Requirements Engineering Conference*, IEEE, 2015.

[Cai17a] A. Cailliau and A. van Lamsweerde, "Runtime monitoring and resolution of probabilistic obstacles to system goals", *Proc. of SEAMS'17— the 12th International Symposium on Self-adaptation and self-managing systems*, IEEE, 2017.

[Cai17b] A. Cailliau, "KAOSTools", https://github.com/ancailliau/KAOSTools.

[Cai17c] A. Cailliau, "Ambulance Dispatch System", https://github.com/ancailliau/ADS.

[Cai17d] A. Cailliau, "BDDSharp", https://github.com/ancailliau/BDDSharp.

[Cai17e] A. Cailliau, "ExpertOpinionSharp", https://github.com/ancailliau/ExpertOpinionSharp.

[Cai17f] A. Cailliau, "LTLSharp", https://github.com/ancailliau/LTLSharp.

[Cal11] R. Calinescu, K. Johnson, and Y. Rafiq, "Using observation ageing to improve Markovian model learning in QoS engineering", *Proc. of ICPE'11—the 2nd International Conference on Performance engineering*, ACM, 2011.

[Cal14] R. Calinescu, Y. Rafiq, K. Johnson, and M.E. Bakır, "Adaptive model learning for continual verification of non-functional properties", *Proc. of ICPE'14—the 5th International conference on Performance engineering*, ACM, 2014.

[Cam14] J. Cámara, G.A. Moreno, and D. Garlan, "Stochastic game analysis and latency awareness for proactive self-adaptation", *Proc. of SEAMS'14—the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ACM, 2014.

[Cam16] J. Cámara, G.A. Moreno, D. Garlan, and B. Schmerl, "Analyzing Latency-Aware Self-Adaptation Using Stochastic Games and Simulations", *ACM Transactions on Autonomous and Adaptive Systems*, Vol. 10, No. 4, 2016.

[Cas11] P. Casanova, B. Schmerl, D. Garlan, and R. Abreu, "Architecture-based run-time fault diagnosis", *Proc. of ECSA'11—European Conference on Software Architecture*, Springer, 2011.

[Cen92] N.S.W. Center, *Handbook of reliability prediction procedures for mechanical equipment*, Carderock Division, Naval Surface Warfare Center, 1992.

[Cha13] G. Chatzikonstantinou, K. Kontogiannis, and IM. Attarian, "A goal driven framework for software project data analytics", *Proc. of CAiSE'17—International Conference on Advanced Information Systems Engineering*, Springer, 2013.

[Cha16a] G. Chatzikonstantinou and K. Kontogiannis, "Run-time requirements verification for reconfigurable systems", *Information and Software Technology*, Vol. 75, 2016, 105-121.

[Cha16b] G. Chatzikonstantinou and K. Kontogiannis, "Efficient parallel reasoning on fuzzy goal models for run time requirements verification", *Software & Systems Modeling*, 2016, 1-26.

[Che06] SW. Cheng, D. Garlan, and B. Schmerl, "Architecture-based self-adaptation in the presence of multiple objectives", *Proc. of SEAMS'06—the 1st Workshop on Software Engineering for Adaptive and Self-Managing Systems*, ACM, 2006.

[Che09a] B.H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic *et al.*, *Software engineering for self-adaptive systems*, Springer, 2009.

[Che09b] B.H. Cheng, R. De Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic *et al.*, "Software engineering for self-adaptive systems: A research roadmap", *Software engineering for self-adaptive systems*, Springer, 2009, 1-26.

[Che09c] B. Cheng, P. Sawyer, N. Bencomo, and J. Whittle, "A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty", *Model Driven Engineering Languages and Systems*, 2009, 468-483.

[Che13] B.H. Cheng, A. Ramirez, and P.K. McKinley, "Harnessing evolutionary computation to enable dynamically adaptive systems to manage uncertainty", *Proc. of CMSBSE'13—the 1st International Workshop on Combining Modelling and Search-Based Software Engineering*, IEEE, 2013.

[Che14] B. Chen, X. Peng, Y. Yu, B. Nuseibeh, and W. Zhao, "Self-adaptation through incremental generative model transformations at runtime", *Proc. of ICSE'14—the 36th International Conference on Software Engineering*, ACM, 2014.

[Chu93] M. Chudleigh, "Hazard analysis using HAZOP: A case study", *Proc. of SAFECOMP'93—the 12th International Conference on Computer Safety, Reliability and Security*, 1993.

[Cle99] R.T. Clemen and R.L. Winkler, "Combining probability distributions from experts in risk analysis", *Risk analysis*, Vol. 19, No. 2, 1999, 187-203.

[Colo1] T.E. Collins, G. Hubbard, and U.S. Nuclear Regulatory Commission, *Technical study of spent fuel pool accident risk at decommissioning nuclear power plants*, Division of Systems Safety and Analysis, Office of Nuclear Reactor Regulation, 2001.

[Coo08] R.M. Cooke and L.L. Goossens, "TU Delft expert judgment data base", *Reliability Engineering & System Safety*, Vol. 93, No. 5, 2008, 657-674.

[Coo91] R.M. Cooke, *Experts in uncertainty: opinion and subjective probability in science*, Oxford University Press, 1991.

[Coro1] S.L. Cornford, M.S. Feather, and K.A. Hicks, "DDP - A tool for life-cycle risk management", *Proc. of Aerospace Conference*, 2001.

[Coro2] S.L. Cornford, J. Dunphy, and M.S. Feather, "Optimizing the design of end-to-end spacecraft systems using risk as a currency", *Proc. of Aerospace Conference*, IEEE, 2002.

[Coro9] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed., MIT Press, 2009.

[Cos15] D. Costal, L. López, M. Morandini, A. Siena, M.C. Annosi, D. Gross, L. Méndez, X. Franch, and A. Susi, "Aligning business goals and risks in oss adoption", *Proc. of ER'15—the 34th International Conference on Conceptual Modeling*, Springer, 2015.

[Cou06] N.R. Council, *Safety and Security of Commercial Spent Nuclear Fuel Storage: Public Report*, National Academies Press, 2006.

[Dal13] F. Dalpiaz, A. Borgida, J. Horkoff, and J. Mylopoulos, "Runtime goal models: Keynote", *Proc. of RCIS'13—the 7th International Conference onResearch Challenges in Information Science*, IEEE, 2013.

[Dam10] C. Damas, B. Lambeau, and A. van Lamsweerde, *COOL: a Car pOOLing support system*, Technical report, Université catholique de Louvain, 2010.

[Dam14] C. Damas, B. Lambeau, and A. van Lamsweerde, "Analyzing Critical Decision-Based Processes", *IEEE Transactions on Software Engineering*, Vol. 40, No. 4, 2014, 338-365.

[Dar07] R. Darimont and M. Lemoine, "Security requirements for civil aviation with UML and goal orientation", *Proc. of REFSQ'07—the 13th International Working Conference on Requirements Engineering: Foundation for Software Quality*, Springer, 2007.

[Dar15] R. Darimont and C. Ponsard, "Supporting quantitative assessment of requirements in goal orientation", *Proc. of RE'15—the 23rd International Requirements Engineering Conference*, IEEE, 2015.

[Dar95] R. Darimont, *Process support for requirements elaboration,* PhD Thesis, Université catholique de Louvain, 1995.

[Dar96] R. Darimont and A. Van Lamsweerde, "Formal refinement patterns for goal-driven requirements elaboration", *Proc. of FSE'96—the 4th Symposium on Foundations of Software Engineering*, ACM, 1996.

[Dar97] R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde, "GRAIL/KAOS: an environment for goal-driven requirements engineering", *Proc. of ICSE'97—the 19th International Conference on Software Engineering*, ACM, 1997.

[DeB05] PT. De Boer, D.P. Kroese, S. Mannor, and R.Y. Rubinstein, "A tutorial on the cross-entropy method", *Annals of operations research*, Vol. 134, No. 1, 2005, 19-67.

[DeL13] R. De Lemos, H. Giese, H.A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N.M. Villegas, T. Vogel *et al.*, "Software engineering for self-adaptive systems: A second research roadmap", *Software Engineering for Self-Adaptive Systems II*, Springer, 2013, 1-32.

[DeL95a] R. De Lemos, A. Saeed, and T. Anderson, "Analyzing safety requirements for process-control systems", *IEEE Software*, Vol. 12, No. 3, 1995, 42-53.

[DeL95b] R. De Lemos, B. Fields, and A. Saeed, "Analysis of safety requirements in the context of system faults and human errors", *Proc. of ECBS'95—the 2nd International Symposium and Workshop on Systems Engineering of Computer Based Systems*, IEEE, 1995.

[DeV17a] B. DeVries and B.H.C. Cheng, "Automatic Detection of Incomplete Requirements Using Symbolic Analysis and Evolutionary Computation", *Proc. of SSBSE'17—the 9th International Symposium on Search Based Software Engineering*, 2017.

[Dug92] J.B. Dugan, S.J. Bavuso, and M.A. Boyd, "Dynamic fault-tree models for fault-tolerant computer systems", *IEEE Transactions on reliability*, Vol. 41, No. 3, 1992, 363-377.

[Dwy98] M.B. Dwyer, G.S. Avrunin, and J.C. Corbett, "Property specification patterns for finite-state verification", *Proc. of FMSP'98—the 2d workshop on Formal methods in software practice*, ACM, 1998.

[dAm05] d'Amorim Marcelo and G. Roşu, "Efficient monitoring of ω-languages", *Proc. of CAV'05—the 17th International Conference on Computer Aided Verification*, Springer, 2005.

[Ear92] J. Earthy, "Hazard and operability study as an approach to software safety assessment", *Proc. of Colloquium on Hazard Analysis*, IET, 1992.

[Eas94] S. Easterbrook, "Resolving requirements conflicts with computer-supported negotiation", *Requirements engineering*, Academic Press Professional, 1994, 41-65.

[Ebe05] R. Ebendt, G. Fey, and R. Drechsler, *Advanced BDD optimization*, Springer, 2005.

[Ell14] S.J. Ellis, E.R. Henderson, T.H. Klinge, J.I. Lathrop, J.H. Lutz, R.R. Lutz, D. Mathur, and A.S. Miner, "Automated requirements analysis for a molecular watchdog timer", *Proc. of ASE'14—the 29th International Conference on Automated Software Engineering*, ACM, 2014.

[Epi09] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time parameter adaptation", *Proc. of ICSE'09—the 31th international conference on Software engineering*, IEEE, 2009.

[Epi10] I. Epifani, C. Ghezzi, and G. Tamburrelli, "Change-point detection for black-box services", *Proc. of FSE'10—the 18th International Symposium on Foundations of software engineering*, ACM, 2010.

[Erd14] G. Erdogan, Y. Li, R.K. Runde, F. Seehusen, and K. Stølen, "Approaches for the combined use of risk analysis and testing: a systematic literature review", *International Journal on Software Tools for Technology Transfer*, Vol. 16, No. 5, 2014, 627-642.

[Esf13] N. Esfahani and S. Malek, "Uncertainty in self-adaptive software systems", *Software Engineering for Self-Adaptive Systems II*, Springer, 2013, 214-238.

[Esp13] P. Espada, M. Goulão, and J. Araújo, "A Framework to Evaluate Complexity and Completeness of KAOS Goal Models", *Proc. of CAiSE'13—the 25th International Conference on Advanced Information Systems Engineering*, 2013.

[Fea03] M.S. Feather and S.L. Cornford, "Quantitative risk-based requirements reasoning", *Requirements Engineering*, Vol. 8, No. 4, 2003, 248-265.

[Fea06] M.S. Feather, S.L. Cornford, J.D. Kiper, and T. Menzies, "Experiences using visualization techniques to present requirements, risks to them, and options for risk mitigation", *Proc. of REV'06—First International Workshop on Requirements Engineering Visualization*, IEEE, 2006.

[Fea08a] M.S. Feather, "Defect Detection and Prevention (DDP)", *Proc. of 14th Monterey workshop*, 2008.

[Fea08b] M.S. Feather, S. Uckun, and K.A. Hicks, "Technology maturation of integrated system health management", *Proc. of STAIF'08—Space Technology and Applications International Forum*, 2008.

[Fea08c] M. Feather, S. Cornford, K. Hicks, J. Kiper, and T. Menzies, "Application of a broad-spectrum quantitative requirements model to early-lifecycle decision making", *IEEE Software*, 2008, 1673-1680.

[Fea98] M.S. Feather, S. Fickas, A. Van Lamsweerde, and C. Ponsard, "Reconciling system requirements and runtime behavior", *Proc. of IWSSD'98—the 9th international workshop on Software specification and design*, IEEE, 1998.

[Fen00] N.E. Fenton and M. Neil, "Software metrics: roadmap", *Proc. of ICSE'00—Conference on The Future of Software Engineering*, ACM, 2000.

[Fen94] P. Fenelon and B. Hebbron, "Applying HAZOP to software engineering models", *Proc. of SARSS'94—Risk Management And Critical Protective Systems*, 1994.

[Fic95] S. Fickas and M.S. Feather, "Requirements monitoring in dynamic environments", *Proc. of RE'95—the 2nd International Symposium on Requirements Engineering*, IEEE, 1995.

[Fil11] A. Filieri, C. Ghezzi, A. Leva, and M. Maggio, "Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements", *Proc. of ASE'11—the 26th International Conference on Automated Software Engineering*, IEEE, 2011.

[Fil14] A. Filieri, H. Hoffmann, and M. Maggio, "Automated design of self-adaptive software with control-theoretical formal guarantees", *Proc. of ICSE'14—the 36th International Conference on Software Engineering*, ACM, 2014.

[Fil15a] A. Filieri, L. Grunske, and A. Leva, "Lightweight adaptive filtering for efficient learning and updating of probabilistic models", *Proc. of ICSE'15—the 37th International Conference on Software Engineering*, IEEE, 2015.

[Fil15b] A. Filieri, M. Maggio, K. Angelopoulos, N. D'Ippolito, I. Gerostathopoulos, A.B. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein *et al.*, "Software engineering meets control theory", *Proc. of SEAMS'15—the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, IEEE, 2015.

[Fil15c] A. Filieri, H. Hoffmann, and M. Maggio, "Automated multi-objective control for self-adaptive software design", *Proc. of FSE'15—the 10th Joint Meeting on Foundations of software engineering*, ACM, 2015.

[Fil16] A. Filieri, G. Tamburrelli, and C. Ghezzi, "Supporting self-adaptation via quantitative verification and sensitivity analysis at run time", *IEEE Transactions on Software Engineering*, Vol. 42, No. 1, 2016, 75-99.

[Fin96] A. Finkelstein and J. Dowell, "A Comedy of Errors: The London Ambulance Service Case Study", *Proc. of IWSSD'96—the 8th International Workshop on Software Specification and Design*, 1996.

[Fra03] X. Franch and N.A. Maiden, "Modelling component dependencies to inform their selection", *Lecture Notes in Computer Science*, Vol. 2580, 2003, 81-91.

[Fra04] X. Franch, G. Grau, and C. Quer, "A framework for the definition of metrics for actor-dependency models", *Proc. of RE'04—the 12th International Requirements Engineering Conference*, IEEE, 2004.

[Fra06] X. Franch, "On the quantitative analysis of agent-oriented models", *Proc. of CAiSE'06—the 18th International Conference on Advanced Information Systems Engineering*, Springer, 2006.

[Fra91] M.D. Fraser, K. Kumar, and V.K. Vaishnavi, "Informal and Formal Requirements Specification Languages: Bridging the Gap", *IEEE Transactions on Software Engineering*, Vol. 17, No. 5, 1991, 454-466.

[Fre14] E.M. Fredericks, B. DeVries, and B.H. Cheng, "AutoRELAX: automatically RELAXing a goal model to address uncertainty", *Empirical Software Engineering*, Vol. 19, No. 5, 2014, 1466-1501.

[Fri01] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, Springer, 2001.

[Gar03] D. Garlan, SW. Cheng, and B. Schmerl, "Increasing system dependability through architecture-based self-repair", *Architecting dependable systems*, Springer, 2003, 61-89.

[Gar04] D. Garlan, SW. Cheng, AC. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure", *Computer*, Vol. 37, No. 10, 2004, 46-54.

[Gar09] D. Garlan, B.R. Schmerl, and SW. Cheng, "Software Architecture-Based Self-Adaptation.", *Autonomic computing and networking*, Vol. 1, 2009, 31-55.

[Ghe09] C. Ghezzi and G. Tamburrelli, "Reasoning on non-functional requirements for integrated services", *Proc. of RE'09—the 17th International Requirements Engineering Conference*, IEEE, 2009.

[Ghe12] C. Ghezzi, J. Greenyer, and V.P. La Manna, "Synthesizing dynamically updating controllers from changes in scenario-based specifications", *Proc. of SEAMS'12—the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, IEEE, 2012.

[Ghe13] C. Ghezzi and A.M. Sharifloo, "Dealing with non-functional requirements for adaptive systems via dynamic software product-lines", *Software Engineering for Self-Adaptive Systems II*, Springer, 2013, 191-213.

[Gia01] D. Giannakopoulou and K. Havelund, "Runtime analysis of linear temporal logic specifications", *Proc. of ASE'01—the 16th International Conference on Automated Software Engineering*, 2001.

[Gie13] H. Giese, H. Müller, M. Shaw, and R. De Lemos, *Software Engineering for Self-Adaptive Systems II*, Springer, 2013.

[Gil09] A. Gil and J. Araujo, "AspectKAOS: integrating early-aspects into KAOS", *Proc. of the 15th Workshop on Early Aspects*, ACM, 2009.

[Gio02] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Reasoning with goal models", *Proc. of ER'02—the 21st International Conference on Conceptual Modeling*, Springer, 2002.

[Gio03] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Formal reasoning techniques for goal models", *Journal on Data Semantics*, Vol. 1, No. 1, 2003, 1-20.

[Gio05a] P. Giorgini, J. Mylopoulos, and R. Sebastiani, "Goal-oriented requirements analysis and reasoning in the tropos methodology", *Engineering Applications of Artificial Intelligence*, Vol. 18, No. 2, 2005, 159-171.

[Gio05b] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, "Modeling security requirements through ownership, permission and delegation", *Proc. of RE'05—the 13th International Requirements Engineering Conference*, IEEE, 2005.

[Gol08] H.J. Goldsby, P. Sawyer, N. Bencomo, B.H. Cheng, and D. Hughes, "Goal-based modeling of dynamically adaptive system requirements", *Proc. of ECBS'08—the 15th International Conference and Workshop on the Engineering of Computer Based Systems*, IEEE, 2008.

[Gra07] J.E. Gray, *Textmate: Power Editing for Everyone*, Pragmatic Bookshelf, 2007.

[Gru08] L. Grunske, "Specification patterns for probabilistic quality properties", *Proc. of ICSE'08—the 30th international conference on Software engineering*, IEEE, 2008.

[Gui15] F.P. Guimaraes, G.N. Rodrigues, D.M. Batista, and R. Ali, "Pragmatic requirements for adaptive systems: A goal-driven modeling and analysis approach", *Proc. of ER'15—the 34th International Conference on Conceptual Modeling*, Springer, 2015.

[Gui17] F.P. Guimarães, G.N. Rodrigues, R. Ali, and D.M. Batista, "Planning runtime software adaptation through pragmatic goal model", *Data & Knowledge Engineering*, Vol. 109, 2017, 25-40.

[Haa02] P. Haapanen and A. Helminen, *Failure mode and effects analysis of software-based automation systems*, Säteilyturvakeskus, 2002.

[Han04] K.M. Hansen, L. Wells, and T. Maier, "HAZOP analysis of UML-based software architecture descriptions of safety-critical systems", *NWUML'04—the 2nd Nordic Workshop on the Unified Modeling Language*, 2004, 59-78.

[Han94] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability", *Formal aspects of computing*, Vol. 6, No. 5, 1994, 512-535.

[Hav04] K. Havelund and G. Roşu, "Efficient monitoring of safety properties", *International Journal on Software Tools for Technology Transfer*, Vol. 6, No. 2, 2004, 158-173.

[Hea11] W. Heaven and E. Letier, "Simulating and optimising design decisions in quantitative goal models", *Proc. of RE'11—the 19th International Requirements Engineering Conference*, IEEE, 2011.

[Hel02] G. Helmer, J. Wong, M. Slagell, V. Honavar, L. Miller, and R. Lutz, "A software fault tree approach to requirements analysis of an intrusion detection system", *Requirements Engineering*, Vol. 7, No. 4, 2002, 207-220.

[Hor10] J. Horkoff and S. Eric, "Finding Solutions in Goal Models: An Interactive Backward Reasoning Approach.", *Proc. of ER'10—the 29th International Conference on Conceptual Modeling*, Springer, 2010.

[Hor11] J. Horkoff and E. Yu, "Analyzing goal models: different approaches and how to choose among them", *Proc. of SAC'11—the 26th Symposium on Applied Computing*, ACM, 2011.

[Hor14] J. Horkoff, R. Salay, M. Chechik, and A. Di Sandro, "Supporting early decision-making in the presence of uncertainty", *Proc. of RE'14—the 22d International Requirements Engineering Conference*, IEEE, 2014.

[Hor16] J. Horkoff and E. Yu, "Interactive goal model analysis for early requirements engineering", *Requirements Engineering*, Vol. 21, No. 1, 2016, 29-61.

[Hug06a] D. Hughes, P. Greenwood, G. Coulson, and G. Blair, "Gridstix: Supporting flood prediction using embedded hardware and next generation grid middleware", *Proc. of WoWMoM'06—the 7th International Symposium on on World of Wireless, Mobile and Multimedia Networks*, IEEE, 2006.

[Hug06b] D. Hughes, P. Greenwood, G. Blair, G. Coulson, F. Pappenberger, P. Smith, and K. Beven, "An intelligent and adaptable grid-based flood monitoring and warning system", *Proc. of the UK eScience All Hands Meeting*, 2006.

[IAEA12] International Atomic Energy Agency, *Storage of Spent Nuclear Fuel*, no. SSG-15, International Atomic Energy Agency, 2012.

[IEC61882] I.E. Commission *et al.*, *IEC 61882: Hazard and operability studies (HAZOP studies)-Application guide*, IEC, 2001.

[Jeg12] C. Jegourel, A. Legay, and S. Sedwards, "Cross-Entropy Optimisation of Importance Sampling Parameters for Statistical Model Checking", *Proc. of CAV'12—the 24th International Conference on Computer Aided Verification*, Springer, 2012.

[Jeg13] C. Jegourel, A. Legay, and S. Sedwards, "Importance splitting for statistical model checking rare properties", *Proc. of CAV'13—the 25th International Conference on Computer Aided Verification*, Springer, 2013.

[Jon08] C. Jones, *Applied software measurement: global analysis of productivity and quality*, McGraw-Hill Education Group, 2008.

[Jon94] C. Jones, *Assessment and control of software risks*, Yourdon Press, 1994.

[Jur01] J. Jürjens, "Towards development of secure systems using UMLsec", *Fundamental approaches to software engineering*, 2001, 187-200.

[Jur02] J. Jürjens, "UMLsec: Extending UML for Secure Systems Development", *Proc. of UML'02—the 5th International Conference on Unified Modeling Language: Model Engineering, Concepts, and Tools*, 2002.

[Jur15] I.J. Jureta, A. Borgida, N.A. Ernst, and J. Mylopoulos, "The requirements problem for adaptive systems", *ACM Transactions on Management Information Systems*, Vol. 5, No. 3, 2015, 17.

[Kai02] H. Kaiya, H. Horai, and M. Saeki, "AGORA: Attributed goal-oriented requirements analysis method", *Proc. of RE'02—the 10th International Requirements Engineering Conference*, Ieee, 2002.

[Kep03] J.O. Kephart and D.M. Chess, "The vision of autonomic computing", *Computer*, Vol. 36, No. 1, 2003, 41-50.

[Kle99] T.A. Kletz, *HAZOP and HAZAN: identifying and assessing process industry hazards*, IChemE, 1999.

[Klo11] J. Kloos, T. Hussain, and R. Eschbach, "Risk-based testing of safety-critical embedded systems driven by fault tree analysis", *Proc. of ICSTW'11—the 4th International Conference on Software Testing, Verification and Validation Workshops*, IEEE, 2011.

[Koy92] R. Koymans, *Specifying message passing and time-critical systems with temporal logic*, Vol. 651, Springer, 1992.

[Kra07] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge", *Proc. of FOSE'07—Future of Software Engineering*, IEEE, 2007.

[Kun75] HT. Kung, F. Luccio, and F.P. Preparata, "On finding the maxima of a set of vectors", *Journal of the ACM*, Vol. 22, No. 4, 1975, 469-476.

[Lam00] A. Van Lamsweerde and E. Letier, "Handling obstacles in goal-oriented requirements engineering", *IEEE Transactions on software engineering*, Vol. 26, No. 10, 2000, 978-1005.

[Lam03] A. van Lamsweerde, S. Brohez, R. De Landtsheer, and D. Janssens, "From system goals to intruder anti-goals: attack generation and resolution for security requirements engineering", *RHAS'03—Workshop on Requirements for High Assurance Systems*, 2003, 49-56.

[Lam04] A. van Lamsweerde, "Elaborating security requirements by construction of intentional anti-models", *Proc. of ICSE'04—the 26th International Conference on Software Engineering*, IEEE, 2004.

[Lam09] A. Van Lamsweerde, *Requirements engineering: From system goals to UML models to software*, John Wiley & Sons, 2009.

[Lam98a] A. Van Lamsweerde and E. Letier, "Integrating obstacles in goal-driven requirements engineering", *Proc. of ICSE'98—the 20th International Conference on Software Engineering*, IEEE, 1998.

[Lam98b] A. van Lamsweerde, R. Darimont, and E. Letier, "Managing conflicts in goal-driven requirements engineering", *IEEE transactions on Software engineering*, Vol. 24, No. 11, 1998, 908-926.

[Lee07] I. Lee, O. Sokolsky, J. Regehr *et al.*, "Statistical runtime checking of probabilistic properties", *Proc. of RV'17—the 17th International Conference on Runtime Verification*, Springer, 2007.

[Lem10] R. de Lemos, H. Giese, H.A. Müller, and M. Shaw, *Software Engineering for Self-Adaptive Systems II*, Springer, 2010.

[Let02] E. Letier, *Reasoning about agents in goal-oriented requirements engineering,* PhD Thesis, Université catholique de Louvain, 2002.

[Let04] E. Letier and A. Van Lamsweerde, "Reasoning about partial goal satisfaction for requirements and design engineering", *Proc. of FSE'04—the 12th International Symposium on Foundations of Software Engineering*, ACM, 2004.

[Let08] E. Letier, J. Kramer, J. Magee, and S. Uchitel, "Deriving event-based transition systems from goal-oriented requirements models", *Automated Software Engineering*, Vol. 15, No. 2, 2008, 175-206.

[Let14] E. Letier, D. Stefan, and E.T. Barr, "Uncertainty, risk, and information value in software requirements and architecture", *Proc. of ICSE'14—the 36th International Conference on Software Engineering*, ACM, 2014.

[Lev01] N.G. Leveson, *Safeware: system safety and computers*,, 2001.

[Lev02] N.G. Leveson, "An approach to designing safe embedded software", *Proc. of EMSOFT'02—the 2nd International Workshop on Embedded Software*, Springer, 2002.

[Lev11] N. Leveson, *Engineering a Safer World: Systems Thinking Applied to Safety*, MIT Press, 2011.

[Lev83] N.G. Leveson and P.R. Harvey, "Software fault tree analysis", *Journal of Systems and Software*, Vol. 3, No. 2, 1983, 173-181.

[Lin11] U. Linz, *Ion Beam Therapy: Fundamentals, Technology, Clinical Applications*, Springer, 2011.

[Lip00] M. Lippert and C.V. Lopes, "A study on exception detection and handling using aspect-oriented programming", *Proc. of ICSE'00—the 22nd International Conference on Software Engineering*, ACM, 2000.

[Liu05] YQ. Liu, YW. Chen, F. Gao, and GP. Jiang, "Risk evaluation using evidence reasoning theory", *Proc. of ICMLC'05—the 4th International Conference on Machine Learning and Cybernetics*, IEEE, 2005.

[Liu13] S. Liu, M. Yamada, N. Collier, and M. Sugiyama, "Change-point detection in time-series data by relative density-ratio estimation", *Neural Networks*, Vol. 43, 2013, 72-83.

[Lod02] T. Lodderstedt, D. Basin, and J. Doser, "SecureUML: A UML-based modeling language for model-driven security", *UML 2002-The Unified Modeling Language*, 2002, 426-441.

[Lun10] M.S. Lund, B. Solhaug, and K. Stølen, *Model-driven risk analysis: the CORAS approach*, Springer, 2010.

[Lut03] R.R. Lutz and I.C. Mikulski, "Requirements discovery during the testing of safety-critical software", *Proc. of ICSE'03—the 25th International Conference on Software Engineering*, IEEE, 2003.

[Lut07] R. Lutz, A. Patterson-Hine, S. Nelson, C.R. Frost, D. Tal, and R. Harris, "Using obstacle analysis to identify contingency requirements on an unpiloted aerial vehicle", *Requirements Engineering*, Vol. 12, No. 1, 2007, 41-54.

[Lut12a] R.R. Lutz, J.H. Lutz, J.I. Lathrop, T.H. Klinge, D. Mathur, D.M. Stull, T.G. Bergquist, and E.R. Henderson, "Requirements analysis for a product family of DNA nanodevices", *Proc. of RE'12—the 20th International Requirements Engineering Conference*, IEEE, 2012.

[Lut12b] R. Lutz, J. Lutz, J. Lathrop, T. Klinge, E. Henderson, D. Mathur, and D.A. Sheasha, "Engineering and verifying requirements for programmable self-assembling nanomachines", *Proc. of ICSE'12—the 34th International Conference on Software Engineering*, IEEE, 2012.

[Man92] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurrent systems: Specification*, Springer, 1992.

[McD02] J. McDermid, "Software Hazard and Safety Analysis", *Proc. of FTRTFT'02—the 7th Formal Techniques in Real-Time and Fault-Tolerant Systems*, 2002.

[McD94] J. McDermid and D. Pumfrey, "A development of hazard analysis to aid software design", *Proc. of COMPASS'94—the 9th Conference on Safety, Reliability, Fault Tolerance, Concurrency and Real Time, Security*, 1994.

[McD95] J.A. McDermid, M. Nicholson, D.J. Pumfrey, and P. Fenelon, "Experience with the application of HAZOP to computer-based systems", *Proc. of COMPASS'95—the 10th Annual Conference on Systems Integrity, Software Safety and Process Security*, IEEE, 1995.

[McD99] J. McDermott and C. Fox, "Using abuse case models for security requirements analysis", *Proc. of ACSAC'99—the 15th Computer Security Applications Conference*, IEEE, 1999.

[Med10] N. Medvidovic and R.N. Taylor, "Software architecture: foundations, theory, and practice", *Proc. of ICSE'10—the 32nd International Conference on Software Engineering*, ACM, 2010.

[Mei12] C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design: OBDD-foundations and applications*, Springer, 2012.

[Men03] T. Menzies, E. Chiang, M. Feather, Y. Hu, and J.D. Kiper, "Condensing Uncertainty via Incremental Treatment Learning", *Software Engineering with Computational Intelligence*, Springer, 2003, 319-361.

[Men14] D.F. Mendonça, R. Ali, and G.N. Rodrigues, "Modelling and analysing contextual failures for dependability requirements", *Proc. of SEAMS'14—the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ACM, 2014.

[Men16] D.F. Mendonça, G.N. Rodrigues, R. Ali, V. Alves, and L. Baresi, "GODA: A goal-oriented requirements engineering framework for runtime dependability analysis", *Information and Software Technology*, Vol. 80, 2016, 245-264.

[Men89] M.B. Mendel and T.B. Sheridan, "Filtering information from human experts", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 19, No. 1, 1989, 6-16.

[Mes04] L. Meshkat, S. Cornford, and M. Feather, "Requirements based system level risk modeling", *Proc. of RAMS'04—the 50th Symposium Reliability and Maintainability*, 2004.

[Mil92] US Military, *Reliability prediction of electronic equipment*, Technical report, MIL-HDBK-217F Notice 1, 1992.

[Mor08] M. Morandini, L. Penserini, and A. Perini, "Towards goal-oriented development of self-adaptive systems", *Proc. of SEAMS'08—the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ACM, 2008.

[Mor13] A. Moreira, R. Chitchyan, J. Araújo, and A. Rashid, *Aspect-oriented requirements engineering*, Springer, 2013.

[Mor16] G.A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Efficient Decision-Making under Uncertainty for Proactive Self-Adaptation", *Proc. of ICAC'16—the 13th International Conference on Autonomic Computing*, 2016.

[Mor17a] G.A. Moreno, O. Strichman, S. Chaki, and R. Vaisman, "Decision-making with cross-entropy for self-adaptation", *Proc. of SEAMS'17—the 12th International Symposium on Self-adaptation and self-managing systems*, IEEE, 2017.

[Mor17b] M. Morandini, L. Penserini, A. Perini, and A. Marchetto, "Engineering requirements for adaptive systems", *Requirements Engineering*, Vol. 22, No. 1, 2017, 77-103.

[Mus07] G. Mussbacher, D. Amyot, J. Araújo, A. Moreira, and M. Weiss, "Visualizing aspect-oriented goal models with aogrl", *Proc. of REV'07—the 2nd International Workshop on Requirements Engineering Visualization*, IEEE, 2007.

[Myl92] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach", *IEEE Transactions on software engineering*, Vol. 18, No. 6, 1992, 483-497.

[Myl99] J. Mylopoulos, L. Chung, and E. Yu, "From object-oriented to goal-oriented requirements analysis", *Communications of the ACM*, Vol. 42, No. 1, 1999, 31-37.

[Nah16] L. Nahabedian, V. Braberman, N. D'Ippolito, S. Honiden, J. Kramer, K. Tei, and S. Uchitel, "Assured and correct dynamic update of controllers", *Proc. of SEAMS'16—the 11th International Workshop on Self-Adaptation and Self-Managing Systems*, IEEE, 2016.

[NHS17] U.K. Department of Health, *Addendum to the Handbook to the NHS Constitution for England*, 2017

[Nie94] J. Nielsen, *Usability engineering*, Elsevier, 1994.

[NuGet] "NuGet Gallery", https://www.nuget.org/.

[Nus01] B. Nuseibeh, "Weaving together requirements and architectures", *Computer*, Vol. 34, No. 3, 2001, 115-119.

[OHa06] A. O'Hagan, C.E. Buck, A. Daneshkhah, J.R. Eiser, P.H. Garthwaite, D.J. Jenkinson, J.E. Oakley, and T. Rakow, *Uncertain judgements: eliciting experts' probabilities*, John Wiley & Sons, 2006.

[Ols10] R. Olsen, *OmniGraffle 5 Diagramming Essentials: Create Better Diagrams with Less Effort Using OmniGraffle*, Packt Publishing Ltd, 2010.

[Ori12] M. Oriol, N.A. Qureshi, X. Franch, A. Perini, and J. Marco, "Requirements Monitoring for Adaptive Service-Based Applications.", *Proc. of REFSQ'12—the 18th International Working Conference on Requirements Engineering: Foundation for Software Quality*, Springer, 2012.

[Otw92] H. Otway and D. von Winterfeldt, "Expert Judgment in Risk Analysis and Management: Process, Context, and Pitfalls", *Risk Analysis*, Vol. 12, No. 1, 1992, 83-93.

[Par07] GY. Park, JS. Lee, SW. Cheon, KC. Kwon, E. Jee, and K.Y. Koh, "Safety analysis of safety-critical software for nuclear digital protection system", *Proc. of SAFECOMP'07—the 26th International Conference on Computer Safety, Reliability, and Security*, Springer, 2007.

[Pas14] L. Pasquale, C. Ghezzi, C. Menghi, C. Tsigkanos, and B. Nuseibeh, "Topology aware adaptive security", *Proc. of SEAMS'14—the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ACM, 2014.

[Pas16] L. Pasquale, P. Spoletini, M. Salehie, L. Cavallaro, and B. Nuseibeh, "Automating trade-off analysis of security requirements", *Requirements Engineering*, Vol. 21, No. 4, 2016, 481-504.

[Per14] D. Perez-Palacin and R. Mirandola, "Uncertainties in the modeling of self-adaptive systems: a taxonomy and an example of availability evaluation", *Proc. of ICPE'14—the 5th International Conference on Performance Engineering*, ACM, 2014.

[Pie13] J. Pieprzyk, T. Hardjono, and J. Seberry, *Fundamentals of computer security*, Springer, 2013.

[Polo7] V. Poladian, D. Garlan, M. Shaw, M. Satyanarayanan, B. Schmerl, and J. Sousa, "Leveraging resource prediction for anticipatory dynamic configuration", *Proc. of SASO'07—the 1st International Conference on Self-Adaptive and Self-Organizing Systems*, IEEE, 2007.

[Pono7] C. Ponsard, P. Massonet, JF. Molderez, A. Rifaut, A. van Lamsweerde, and H.T. Van, "Early verification and validation of mission critical systems", *Formal Methods in System Design*, Vol. 30, No. 3, 2007, 233.

[Pon17] C. Ponsard and R. Darimont, "Quantitative Assessment of Goal Models within and beyond the Requirements Engineering Tool: A Case Study in the Accessibility Domain", *Proc. of iStar'14—the 10th International i* Workshop*, 2017.

[Pot95] C. Potts, "Using schematic scenarios to understand user needs", *Proc. of DIS'95—the 1st conference on Designing interactive systems: processes, practices, methods, & techniques*, ACM, 1995.

[Pub13] SPF Santé Publique, "Manuel belge de la régulation médicale", https://www.health.belgium.be/fr/manuel-belge-de-la-regulation-medicale.

[Quro9] N.A. Qureshi and A. Perini, "Engineering adaptive requirements", *Proc. of SEAMS'09—the 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, IEEE, 2009.

[Qur10a] N.A. Qureshi, A. Perini, N.A. Ernst, and J. Mylopoulos, "Towards a continuous requirements engineering framework for self-adaptive systems", *Proc. of RE@RunTime'10—the 1st International Workshop on Requirements @ Run Time*, IEEE, 2010.

[Qur10b] N.A. Qureshi and A. Perini, "Requirements engineering for adaptive service based applications", *Proc. of RE'10—the 18th International Requirements Engineering Conference*, IEEE, 2010.

[Qur11a] N.A. Qureshi, S. Liaskos, and A. Perini, "Reasoning about adaptive requirements for self-adaptive systems at runtime", *Proc. of RE@RunTime'11—the 2nd International Workshop on Requirements @ Run Time*, IEEE, 2011.

[Qur11b] N. Qureshi, I. Jureta, and A. Perini, "Requirements engineering for self-adaptive systems: Core ontology and problem statement", *Proc. of CAiSE'11—the 23th International Conference on Advanced Information Systems Engineering*, Springer, 2011.

[Rajo8] A. Rajan, M. Whalen, M. Staats, and M. Heimdahl, "Requirements coverage as an adequacy measure for conformance testing", *Formal Methods and Software Engineering*, 2008, 86-104.

[Ram10] A.J. Ramirez, B.H. Cheng, P.K. McKinley, and B.E. Beckmann, "Automatically generating adaptive logic to balance non-functional tradeoffs during reconfiguration", *Proc. of ICAC'10—the 7th International Conference on Autonomic Computing*, ACM, 2010.

[Ram11a] A.J. Ramirez, D.B. Knoester, B.H. Cheng, and P.K. McKinley, "Plato: a genetic algorithm approach to run-time reconfiguration in autonomic computing systems", *Cluster Computing*, Vol. 14, No. 3, 2011, 229-244.

[Ram11b] A. Ramirez and B. Cheng, "Automatic derivation of utility functions for monitoring software requirements", *Model Driven Engineering Languages and Systems*, 2011, 501-516.

[Ram12a] A. Ramirez, E. Fredericks, A. Jensen, and B. Cheng, "Automatically RELAXing a goal model to cope with uncertainty", *Search Based Software Engineering*, 2012, 198-212.

[Ram12b] A.J. Ramirez, B.H. Cheng, N. Bencomo, and P. Sawyer, "Relaxing claims: Coping with uncertainty while evaluating assumptions at run time", *Proc. of MODELS'12—the 15th International Conference on Model Driven Engineering Languages and Systems*, Springer, 2012.

[Rao09] K.D. Rao, V. Gopika, V.S. Rao, H. Kushwaha, A.K. Verma, and A. Srividya, "Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment", *Reliability Engineering & System Safety*, Vol. 94, No. 4, 2009, 872-883.

[Red05] F. Redmill, "Theory and practice of risk-based testing", *Software Testing, Verification and Reliability*, Vol. 15, No. 1, 2005, 3-20.

[Ref09] A. Refsdal and K. Stølen, "Employing key indicators to provide a dynamic risk picture with a notion of confidence", *Proc. of IFIPTM'09—the 3d International Conference on Trust Management*, Springer, 2009.

[Ref15] A. Refsdal, B. Solhaug, and K. Stølen, "Security risk analysis of system changes exemplified within the oil and gas domain", *International Journal on Software Tools for Technology Transfer*, Vol. 17, No. 3, 2015, 251-266.

[Rob03] W.N. Robinson, "Monitoring web service requirements", *Proc. of RE'03—the 9th International Requirements Engineering Conference*, IEEE, 2003.

[Rob05] W.N. Robinson, "Implementing rule-based monitors within a framework for continuous requirements monitoring", *Proc. of HICSS'05—the 38th International Conference on System Sciences*, IEEE, 2005.

[Rob06] W.N. Robinson, "A requirements monitoring framework for enterprise systems", *Requirements engineering*, Vol. 11, No. 1, 2006, 17-41.

[Rob07] W. Robinson, "Extended OCL for goal monitoring", *Electronic Communications of the EASST*, Vol. 9, 2007.

[Rob12] S. Robertson and J. Robertson, *Mastering the requirements process: Getting requirements right*, Addison-Wesley, 2012.

[Roh11] V.K. Rohatgi, "Semi-Variance in Finance", *International Encyclopedia of Statistical Science*, Springer, 2011, 1297-1298.

[Rui15] E. Ruijters and M. Stoelinga, "Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools", *Computer science review*, Vol. 15, 2015, 29-62.

[Rui17] E.J.J. Ruijters, D. Reijsbergen, PT. de Boer, and M.I.A. Stoelinga, "Rare event simulation for dynamic fault trees", *Proc. of SAFECOMP'17—the 36th International Conference on Computer Safety, Reliability, and Security*, 2017.

[Rya93] M. Ryan, "Defaults in specifications", *Proc. of RE'93—the first International Symposium on Requirements Engineering*, IEEE, 1993.

[Sab11] M. Sabetzadeh, D. Falessi, L. Briand, S. Di Alesio, D. McGeorge, V. hjem, and J. Borg, "Combining goal models, expert elicitation, and probabilistic simulation for qualification of new technology", *Proc. of HASE'11—the 13th International Symposium on High-Assurance Systems Engineering*, IEEE, 2011.

[Sal12] R. Salay, M. Famelis, and M. Chechik, "Language independent refinement using partial modeling", *Fundamental Approaches to Software Engineering*, 2012, 224-239.

[Sam05] U. Sammapun, I. Lee, and O. Sokolsky, "RT-MaC: Runtime monitoring and checking of quantitative and probabilistic properties", *Proc. of RTCSA'05—the 11th International Conference on Embedded and Real-Time Computing Systems and Applications*, IEEE, 2005.

[Saw10] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, "Requirements-aware systems: A research agenda for re for self-adaptive systems", *Proc. of RE'10—the 18th International Requirements Engineering Conference*, IEEE, 2010.

[Sch11] B. Schneier, *Secrets and lies: digital security in a networked world*, John Wiley & Sons, 2011.

[Sch93] PY. Schobbens, "Exceptions for algebraic specifications: on the meaning of 'but'", *Science of Computer Programming*, Vol. 20, No. 1-2, 1993, 73-111.

[Sch99] B. Schneier, "Attack trees", *Dr. Dobb's journal*, Vol. 24, No. 12, 1999, 21-29.

[Seb04] R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Simple and minimum-cost satisfiability for goal models", *Proc. of CAiSE'04—the 16th International Conference on Advanced Information Systems Engineering*, Springer, 2004.

[See12] F. Seehusen and B. Solhaug, "Tool-supported risk modeling and analysis of evolving critical infrastructures", *Proc. of ARES'12—the 7th International Conference on Availability, Reliability, and Security*, Springer, 2012.

[Sha06] A.A. Shapiro, G. Price, S. Cornford, Y. Gawdiak, M. Feather, and W.R. Ricks, "Planning a Large-Scale progression of R&D-a Pilot study in the Aerospace Domain", *Proc. of Aerospace Conference*, IEEE, 2006.

[Sha12] K.V. Sharma and P. Kumar, "An efficient risk analysis in requirement engineering", *Proc. of NUiCONE'12—Nirma University International Conference on Engineering*, IEEE, 2012.

[Sha13] K.V. Sharma and D.P. Kumar, "A Method to Risk Analysis in Requirement Engineering Using Tropos Goal Model with Optimized Candidate Solutions", *International Journal of Computer Science*, Vol. 10, No. 6, 2013, 250-259.

[Sie14] A. Siena, M. Morandini, and A. Susi, "Modelling risks in open source software component selection", *Proc. of ER'14—the 33d International Conference on Conceptual Modeling*, Springer, 2014.

[Sino5] G. Sindre and A.L. Opdahl, "Eliciting security requirements with misuse cases", *Requirements engineering*, Vol. 10, No. 1, 2005, 34-44.

[Sol11] M. Soldal Lund, B. Solhaug, and K. Stolen, *Model-Driven Risk Analysis*, Springer, 2011.

[Sol13a] B. Solhaug and F. Seehusen, "Model-driven risk analysis of evolving critical infrastructures", *Journal of Ambient Intelligence and Humanized Computing*, Vol. 5, No. 2, 2014, 187-204.

[Sol13b] B. Solhaug, K. Stølen *et al.*, "An approach to select cost-effective risk countermeasures", *Proc. of BSEC'13—the 27th Conference on Data and Applications Security and Privacy*, Springer, 2013.

[Sou11a] V.E.S. Souza and J. Mylopoulos, "From awareness requirements to adaptive systems: A control-theoretic approach", *Proc. of RE@RunTime'11—the 2nd International Workshop on Requirements @ Run Time*, IEEE, 2011.

[Sou11b] V.E.S. Souza, A. Lapouchnian, W.N. Robinson, and J. Mylopoulos, "Awareness requirements for adaptive systems", *Proc. of SEAMS'11—the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ACM, 2011.

[Sou12] V.E.S. Souza, A. Lapouchnian, and J. Mylopoulos, "(Requirement) evolution requirements for adaptive systems", *Proc. of SEAMS'12—the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, IEEE, 2012.

[Sou13] V.E.S. Souza, A. Lapouchnian, K. Angelopoulos, and J. Mylopoulos, "Requirements-driven software evolution", *Computer Science-Research and Development*, Vol. 28, No. 4, 2013, 311-329.

[Sta06] T. Stahl and M. Volter, *Model-driven software development: technology, engineering, management*, J. Wiley & Sons, 2006.

[Sta92] Bureau Labor Statistics, *Occupational injury and illness classification manual*, 1992

[Sut98] A.G. Sutcliffe, N.A. Maiden, S. Minocha, and D. Manuel, "Supporting scenario-based requirements engineering", *IEEE Transactions on software engineering*, Vol. 24, No. 12, 1998, 1072-1088.

[Tea00] R Core Team, "R language definition", https://cran.r-project.org/doc/manuals/R-lang.pdf.

[Tha05] P. Thati and G. Roşu, "Monitoring algorithms for metric temporal logic specifications", *Electronic Notes in Theoretical Computer Science*, Vol. 113, 2005, 145-162.

[Vai96] R. Vaidhyanathan and V. Venkatasubramanian, "A semi-quantitative reasoning methodology for filtering and ranking HAZOP results in HAZOPExpert", *Reliability Engineering & System Safety*, Vol. 53, No. 2, 1996, 185 - 203.

[Ves81] W.E. Vesely, F.F. Goldberg, N.H. Roberts, and D.F. Haasl, *Fault tree handbook*, Technical report, Nuclear Regulatory Commission, 1981.

[Vid12] A. Videla and J.J. Williams, *RabbitMQ in action: distributed messaging for everyone*, Manning, 2012.

[Vos08] D. Vose, *Risk analysis: a quantitative guide*, John Wiley & Sons, 2008.

[Wac07] D. Wackerly, W. Mendenhall, and R. Scheaffer, *Mathematical statistics with applications*, Nelson Education, 2007.

[Wel09] K. Welsh and P. Sawyer, "Requirements tracing to support change in dynamically adaptive systems", *Requirements Engineering: Foundation for Software Quality*, 2009, 59-73.

[Wel10] K. Welsh and P. Sawyer, "Understanding the scope of uncertainty in dynamically adaptive systems", *Requirements Engineering: Foundation for Software Quality*, 2010, 2-16.

[Wel11] K. Welsh, P. Sawyer, and N. Bencomo, "Towards requirements aware systems: Runtime resolution of design-time assumptions", *Proc. of ASE'11—the 26th International Conference on Automated Software Engineering*, IEEE, 2011.

[Wha06] M.W. Whalen, A. Rajan, M.P. Heimdahl, and S.P. Miller, "Coverage metrics for requirements-based testing", *Proc. of ISSTA'06—the International Symposium on Software Testing and Analysis*, ACM, 2006.

[Whi09] J. Whittle, P. Sawyer, N. Bencomo, B.H. Cheng, and JM. Bruel, "Relax: Incorporating uncertainty into the specification of self-adaptive systems", *Proc. of RE'09—the 17th International Requirements Engineering Conference*, IEEE, 2009.

[Whi10] J. Whittle, P. Sawyer, N. Bencomo, B.H. Cheng, and JM. Bruel, "RELAX: a language to address uncertainty in self-adaptive systems requirement", *Requirements Engineering*, Vol. 15, No. 2, 2010, 177-196.

[Yan14] Z. Yang, Z. Li, Z. Jin, and Y. Chen, "A systematic literature review of requirements modeling and analysis for self-adaptive systems", *Proc. of REFSQ'14—the 20th International Working Conference on Requirements Engineering: Foundation for Software Quality*, Springer, 2014.

[Yu97] E.S. Yu, "Towards modelling and reasoning support for early-phase requirements engineering", *Proc. of RE'97—the 3d International Symposium on Requirements Engineering*, IEEE, 1997.

[Zha06] J. Zhang and B.H. Cheng, "Using temporal logic to specify adaptive program semantics", *Journal of Systems and Software*, Vol. 79, No. 10, 2006, 1361-1369.

[Zow97] D. Zowghi and R. Offen, "A logical framework for modeling and reasoning about the evolution of requirements", *Proc. of RE'97—the 3d International Symposium on Requirements Engineering*, IEEE, 1997.

Appendix A

Case-studies

A.1 Case-Study: Brussels Ambulance Dispatching System

This section provides the complete specifications for the Brussels Ambulance Despatch System, presented in Section 9.3.3.

A.1.1 Goal specifications

This section provides the complete specifications for the goals of the Brussels Ambulance Despatch System, presented in Section 9.3.3.

Achieve [Incident Resolved]

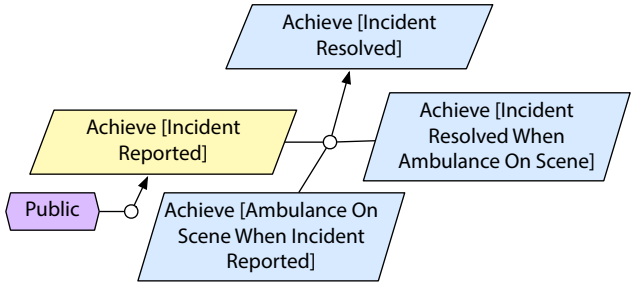


Figure A.1 Refinements for Achieve [Incident Resolved].

Goal Achieve [Incident Resolved]
Def All incidents shall be resolved. An incident is resolved if the victim is treated on the incident scene or transported to an hospital.
RSR 0.95
RefinedBy Achieve [Incident Reported], Achieve [Ambulance On Scene When Incident Reported], Achieve [Incident Resolved When Ambulance On Scene]

Goal Achieve [Incident Reported]
Def All incidents in the real-world shall be reported to the ambulance dispatching.
AssignedTo Public

Goal Achieve [Ambulance On Scene When Incident Reported]
Def An ambulance shall be on scene within 12 minutes for every reported incident.
RefinedBy Achieve [Ambulance Mobilized When Incident Reported], Achieve [Ambulance Left Station When Mobilized], Achieve [Mobilized Ambulance On Scene]

Goal Achieve [Incident Resolved When Ambulance On Scene]
Def The incident shall be resolved by the ambulance staff when the ambulance in on the incident scene.
RefinedBy Achieve [Incident Resolved On Scene When No Transport Required], Achieve [Patient Transported At Hospital When Required]

Achieve [Incident Resolved When Ambulance On Scene]

Goal Achieve [Patient Transported At Hospital When Required]
Def The patient shall be transported to the nearest and most appropriate hospital when required by the patient condition. The incident is resolved when the patient is at hospital.
RefinedBy Maintain [Appropriate Hospital Known], Achieve [Patient Transported At Hospital When Required]

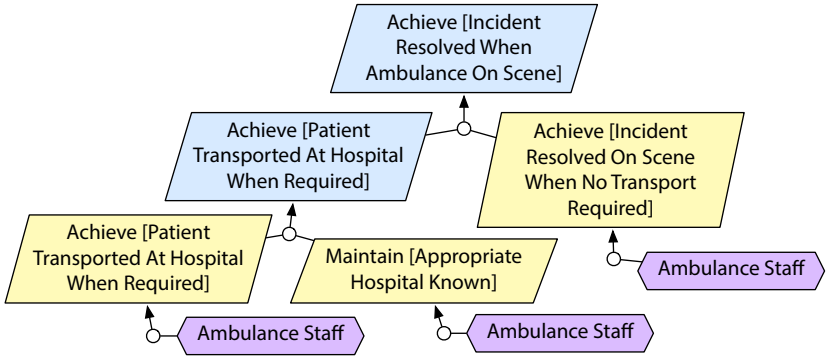


Figure A.2 Refinements for Achieve [Incident Resolved When Ambulance On Scene].

Goal Achieve [Incident Resolved On Scene When No Transport Required]
Def The incident shall be resolved on the incident scene if no transport of the victim towards an hospital is required.
ObstructedBy Patient Not Treated At Location
AssignedTo Ambulance Staff

Goal Achieve [Patient Transported At Hospital When Required]
Def The patient shall be transported to the indicated hospital when required by the patient condition.
ObstructedBy Patient Not Admitted At Or Transported To Hospital
AssignedTo Ambulance Staff

Goal Maintain [Appropriate Hospital Known]
Def The nearest and most appropriate hospital is known to the ambulance staff. Such hospital is indicated on the mobilization order.
AssignedTo Ambulance Staff

Achieve [Ambulance On Scene When Incident Reported]

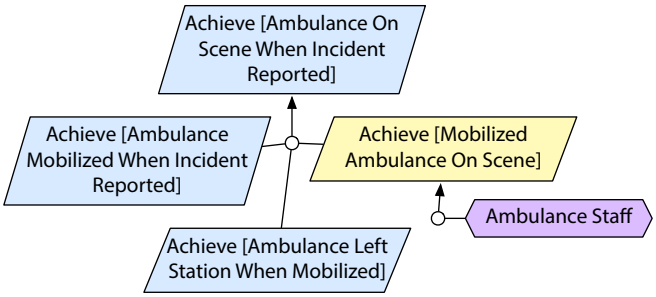


Figure A.3 Refinements for Achieve [Ambulance On Scene When Incident Reported].

Goal Achieve [Ambulance Mobilized When Incident Reported]
Def When an incident is reported, an ambulance shall be mobilized within 2 minutes.
RefinedBy Maintain [Accurate Incident Form], Achieve [Ambulance Mobilized Based On Incident Form]

Goal Achieve [Ambulance Left Station When Mobilized]
Def The mobilized ambulance shall leave the ambulance station within 2 minutes and the departure shall be confirmed by the staff to the dispatching software.
RefinedBy Achieve [Ambulance Physically Leaving Station When Mobilized], Achieve [Ambulance Leaving Status Known]

Goal Achieve [Mobilized Ambulance On Scene]
Def The mobilized ambulance shall be on the incident scene within 12 minutes.
ObstructedBy Mobilized Ambulance Not On Scene
AssignedTo Ambulance Staff

Achieve [Ambulance Mobilized When Incident Reported]

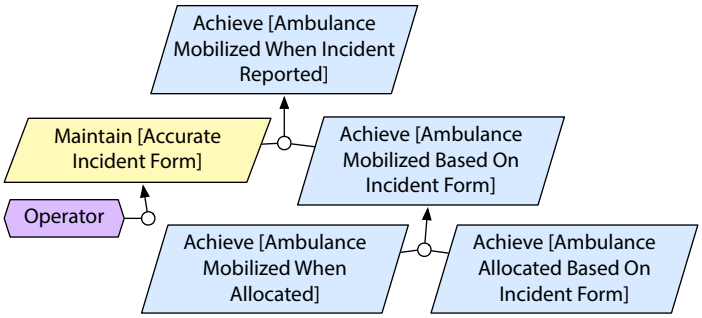


Figure A.4 Refinements for Achieve [Ambulance Mobilized When Incident Reported].

Goal Maintain [Accurate Incident Form]
Def The information about the incident shall be accurate. Informations about the incident include when it occurs and its position.
AssignedTo Operator

Goal Achieve [Ambulance Mobilized Based On Incident Form]
Def An ambulance shall be mobilized when an incident is reported within 2 minutes, based on the informations in the incident form.
RefinedBy Achieve [Ambulance Allocated Based On Incident Form], Achieve [Ambulance Mobilized When Allocated]

Goal Achieve [Ambulance Mobilized When Allocated]
Def The allocated ambulance shall be mobilized within 1 minute. An ambulance is mobilized when the ambulance staff confirmed the mobilization order.
RefinedBy Achieve [Ambulance Mobilized When Allocated At Station], Achieve [Ambulance Mobilized When Allocated On Road]

Goal Achieve [Ambulance Allocated Based On Incident Form]
Def An ambulance shall be allocated when an incident is reported within 1 minute, based on the informations in the incident form. An ambulance is allocated when the dispatching system decided which ambulance shall intervene on that incident.
RefinedBy Achieve [Ambulance Allocated Based On Incident Form When Ambulance Available], Ambulance Available

Achieve [Ambulance Mobilized When Allocated]

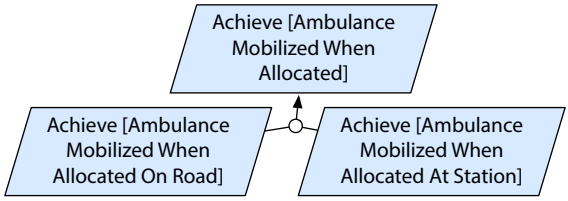


Figure A.5 Refinements for Achieve [Ambulance Mobilized When Allocated].

Goal Achieve [Ambulance Mobilized When Allocated On Road]
Def The allocated ambulance on road shall be mobilized within 1 minute.
RefinedBy Achieve [Allocated Ambulance Mobilization On Road Based On Location Info], Maintain [Accurate Ambulance Availability Known]

Goal Achieve [Ambulance Mobilized When Allocated At Station]
Def The allocated ambulance at station shall be mobilized within 1 minute.
RefinedBy Achieve [Allocated Ambulance Mobilization At Station Based On Location Info], Maintain [Accurate Ambulance Availability Known]

Achieve [Ambulance Mobilized When Allocated On Road]

Goal Maintain [Accurate Ambulance Availability Known]
Def Accurate information about the availability of the ambulance is known to the dispatching software.
RefinedBy Maintain [Accurate Status Encoded On MDT], Maintain [Status Known Based On Encoded Information]

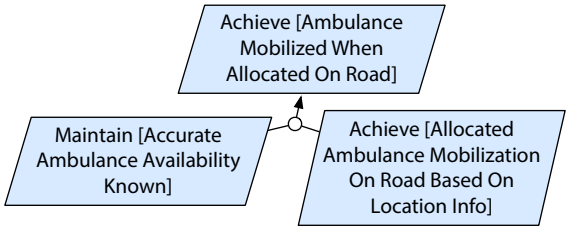


Figure A.6 Refinements for Achieve [Ambulance Mobilized When Allocated On Road].

Goal Achieve [Allocated Ambulance Mobilization On Road Based On Location Info]

Def The allocated ambulance on road shall be mobilized within 1 minute. Whether the ambulance is on road is determined by the location information.

RefinedBy Achieve [Mobilization Order Displayed On MDT], Achieve [Mobilization Order Printed At Station], Achieve [Mobilization Order Confirmed by Radio], Achieve [Allocated Ambulance Mobilization When Mobilization Order Displayed And Radio Contact]

Maintain [Accurate Ambulance Status Known]

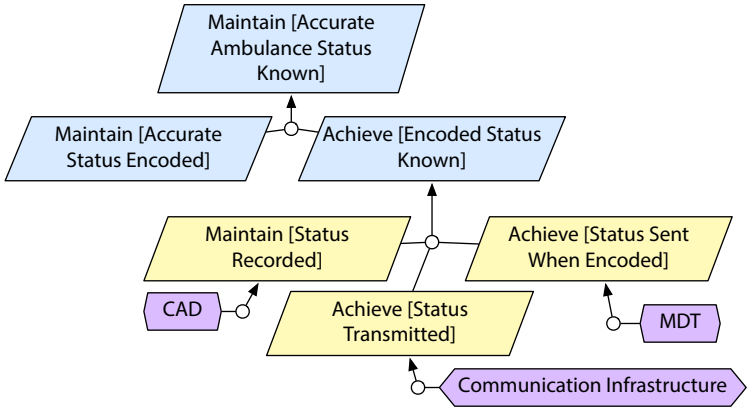


Figure A.7 Refinements for Maintain [Accurate Ambulance Status Known].

Goal Maintain [Accurate Ambulance Status Known]
Def The status of the ambulance shall be accurately known to the dispatching software. A status is accurate if the known status correspond to the status of the ambulance in the real-world. The status shall be known to the dispatching software within 3 minutes when encoded by the staff.
RefinedBy Maintain [Accurate Status Encoded], Achieve [Encoded Status Known]

Goal Maintain [Accurate Status Encoded]
Def The accurate status shall always be encoded.
RefinedBy Maintain [Status Leaving Encoded When Leaving Station], Maintain [Status OnScene Encoded When On Scene], Maintain [Status ToHospital Encoded When Leaving Towards Hospital], Maintain [Status AtHospital Encoded When At Hospital], Maintain [Status AvailableRadio Encoded When Available by Radio], Maintain [Status AvailableStation Encoded When Available at Station], Maintain [Status Unavailable Encoded When Unavailable]

Goal Achieve [Encoded Status Known]
Def The encoded availability information shall be known to the despatching software.
RefinedBy Achieve [Status Sent When Encoded], Achieve [Status Transmitted], Maintain [Status Recorded]

Goal Maintain [Status Recorded]
Def The encoded availability information shall be recorded when transmitted.
AssignedTo CAD

Goal Achieve [Status Transmitted]
Def The encoded availability information shall be transmitted to the dispatching software.
AssignedTo Communication Infrastructure

Goal Achieve [Status Sent When Encoded]
Def The encoded status information shall be sent to the dispatching software.
AssignedTo MDT

Maintain [Accurate Status Encoded]

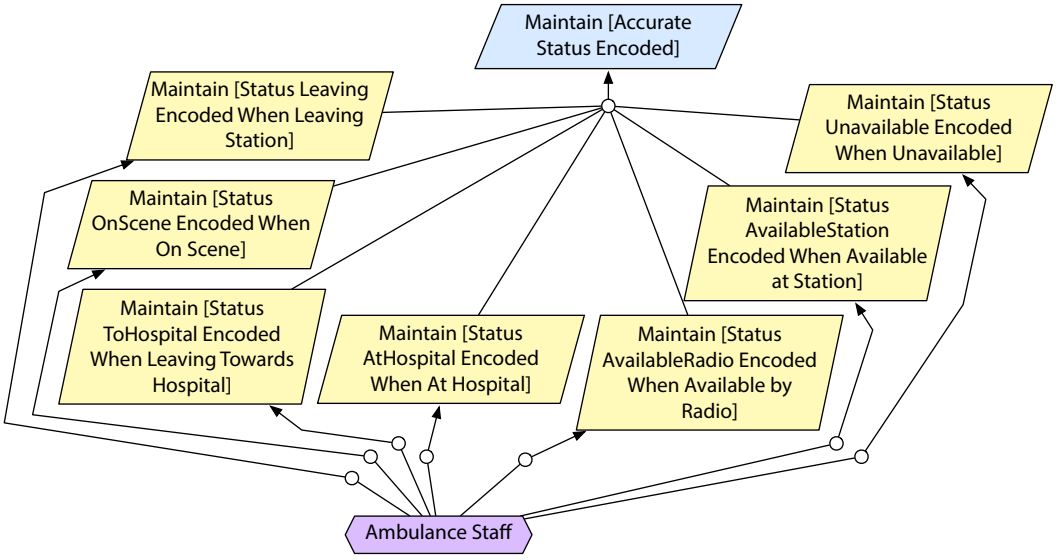


Figure A.8 Refinements for Maintain [Accurate Status Encoded].

Goal Maintain [Status Leaving Encoded When Leaving Station]
Def The Status Leaving is encoded when leaving the station.
ObstructedBy Accurate Leaving Status Not Encoded On MDT Or Innaccurate
AssignedTo Ambulance Staff

Goal Maintain [Status OnScene Encoded When On Scene]
Def The Status OnScene is encoded when arriving on scene.
ObstructedBy Accurate OnScene Status Not Encoded On MDT Or Innaccurate
AssignedTo Ambulance Staff

Goal Maintain [Status ToHospital Encoded When Leaving Towards Hospital]
Def The Status ToHospital is encoded when leaving the incident scene towards the hospital.
ObstructedBy Accurate ToHospital Status Not Encoded On MDT Or Innaccurate
AssignedTo Ambulance Staff

Goal Maintain [Status AtHospital Encoded When At Hospital]
Def The Status AtHospital is encoded when arriving at hospital.
ObstructedBy Accurate AtHospital Status Not Encoded On MDT Or Innaccurate
AssignedTo Ambulance Staff

Goal Maintain [Status AvailableRadio Encoded When Available by Radio]
Def The Status AvailableRadio is encoded when avaiable by radio.
ObstructedBy Accurate AvailableRadio Status Not Encoded On MDT Or Innaccurate
AssignedTo Ambulance Staff

Goal Maintain [Status AvailableStation Encoded When Available at Station]
Def The Status AvailableStation is encoded when available at station.
ObstructedBy Accurate AvailableStation Status Not Encoded On MDT Or Innaccurate
AssignedTo Ambulance Staff

Goal Maintain [Status Unavailable Encoded When Unavailable]
Def The Status Unavailable is encoded when unavailable.
ObstructedBy Accurate Unavailable Status Not Encoded On MDT Or Innaccurate
AssignedTo Ambulance Staff

Achieve [Allocated Ambulance Mobilization On Road Based On Location Info]

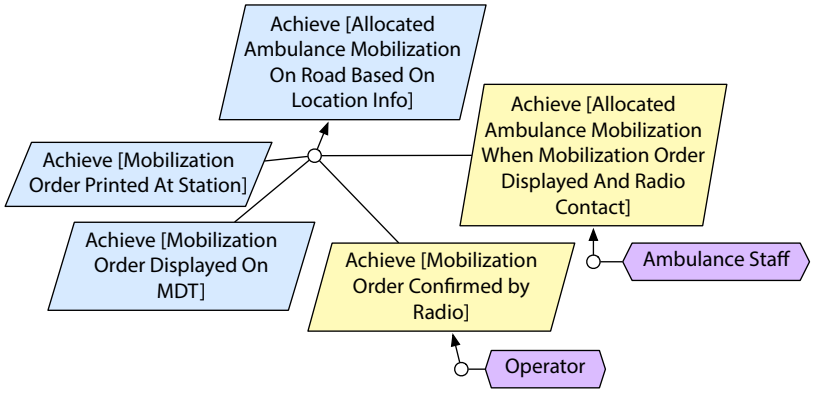


Figure A.9 Refinements for Achieve [Allocated Ambulance Mobilization On Road Based On Location Info].

Goal Achieve [Mobilization Order Printed At Station]
Def The mobilization order shall be printed at station when ambulance is allocated within 50 seconds.
RefinedBy Achieve [Mobilization Order Sent To Printer], Achieve [Mobilization Order Transmitted], Achieve [Mobilization Order Printed]

Goal Achieve [Mobilization Order Displayed On MDT]
Def The mobilization order shall be displayed on the Mobile Data Terminal within 40 seconds.
RefinedBy Achieve [Mobilization Order Sent To MDT], Achieve [Mobilization Order Transmitted], Achieve [Mobilization Order Displayed]

Goal Achieve [Mobilization Order Confirmed by Radio]
Def The mobilization order shall be confirmed orally by radio.
AssignedTo Operator

Goal Achieve [Allocated Ambulance Mobilization When Mobilization Order Displayed And Radio Contact]

Def The allocated ambulance shall be mobilized when the mobilization order is displayed on the MDT and the mobilization is confirmed by radio. The mobilization occurs within 10 seconds of the print or confirmation.

ObstructedBy Allocated Ambulance Not Mobilized And Mobilization Order Displayed And Confirmed

AssignedTo Ambulance Staff

Achieve [Mobilization Order Displayed On MDT]

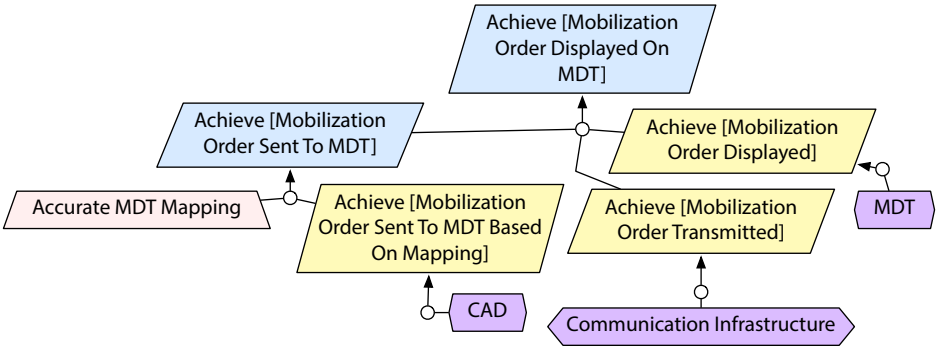


Figure A.10 Refinements for Achieve [Mobilization Order Displayed On MDT].

Goal Achieve [Mobilization Order Sent To MDT]

Def The mobilization order shall be sent to the MDT corresponding to the allocated ambulance.

RefinedBy Achieve [Mobilization Order Sent To MDT Based On Mapping], Accurate MDT Mapping

Goal Achieve [Mobilization Order Sent To MDT Based On Mapping]

Def The mobilization order shall be sent to the correspondign ambulance based on the mapping between ambulances and MDTs within 5 seconds.

AssignedTo CAD

Goal Achieve [Mobilization Order Transmitted]
Def The sent mobilization order shall be transmitted to the station printer.
AssignedTo Communication Infrastructure

Goal Achieve [Mobilization Order Displayed]
Def The transmitted mobilization order shall be displayed on the MDT within 5 seconds.
ObstructedBy Mobilization Order Not Displayed
AssignedTo MDT

Achieve [Mobilization Order Printed At Station]

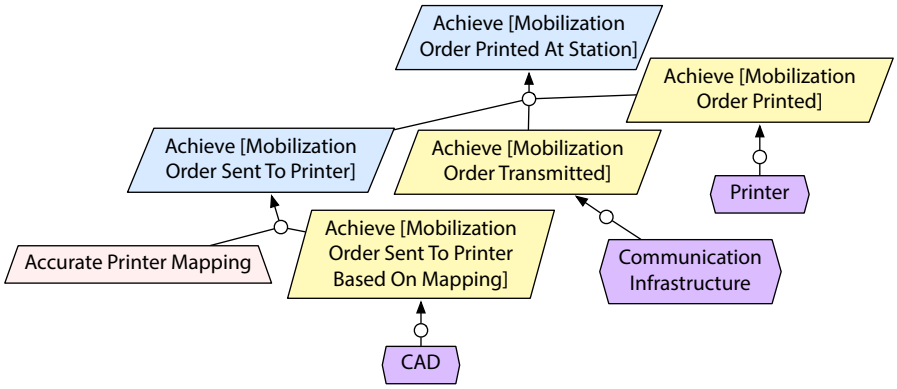


Figure A.11 Refinements for Achieve [Mobilization Order Printed At Station].

Goal Achieve [Mobilization Order Sent To Printer]
Def The mobilization order shall be printed on the station printer when the ambulance is allocated. The station printer is the printer at the station to which the ambulance is assigned.
RefinedBy Achieve [Mobilization Order Sent To Printer Based On Mapping], Accurate Printer Mapping

Goal Achieve [Mobilization Order Sent To Printer Based On Mapping]
Def The mobilization order shall be sent to the station printer when the ambulance is allocated.
AssignedTo CAD

Goal Achieve [Mobilization Order Transmitted]
Def The sent mobilization order shall be transmitted within 5 seconds.
ObstructedBy Mobilization Order Not Transmitted To MDT
AssignedTo Communication Infrastructure

Goal Achieve [Mobilization Order Printed]
Def The transmitted mobilization order shall be printed.
ObstructedBy Mobilization Order Not Printed
AssignedTo Printer

Achieve [Ambulance Mobilized When Allocated At Station]

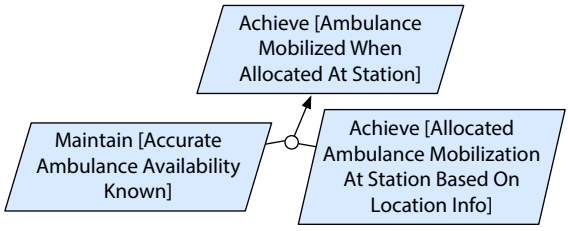


Figure A.12 Refinements for Achieve [Ambulance Mobilized When Allocated At Station].

Goal Achieve [Allocated Ambulance Mobilization At Station Based On Location Info]
Def The allocated ambulance at station shall be mobilized within 1 minute. Whether the ambulance is at station is determined by the location information.
RefinedBy Achieve [Mobilization Order Displayed On MDT], Achieve [Mobilization Order Printed At Station], Achieve [Mobilization Order Confirmed By Phone], Achieve [Allocated Ambulance Mobilization When Mobilization Order Printed And Phone Contact]

Achieve [Allocated Ambulance Mobilization At Station Based On Location Info]

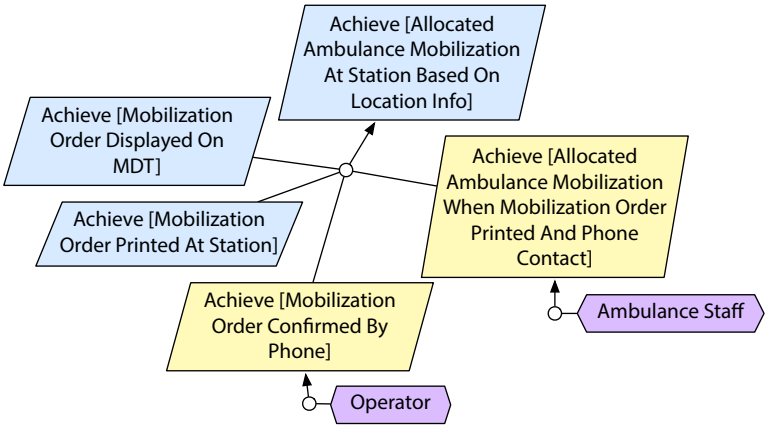


Figure A.13 Refinements for Achieve [Allocated Ambulance Mobilization At Station Based On Location Info].

Goal Achieve [Mobilization Order Confirmed By Phone]
Def The mobilization order shall be confirmed by a phone call within 50 seconds.
AssignedTo Operator

Goal Achieve [Allocated Ambulance Mobilization When Mobilization Order Printed And Phone Contact]
Def The allocated ambulance shall be mobilized when the mobilization order is printed on the station printer and the mobilization has been confirmed by a phone call. The mobilization occurs within 10 seconds of the print or confirmation.
ObstructedBy Allocated Ambulance Not Mobilized And Mobilization Order Printed And Confirmed
AssignedTo Ambulance Staff

Maintain [Accurate Ambulance Info Known]

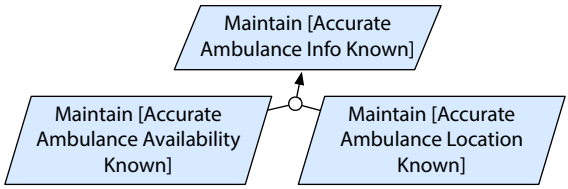


Figure A.14 Refinements for Maintain [Accurate Ambulance Info Known].

Goal Maintain [Accurate Ambulance Location Known]
Def Accurate location of the ambulance is known to the dispatching software. A location is accurate if the position of the ambulance in the real-world and the dispatching position do not differ by more than 250 meters.
RefinedBy Achieve [Ambulance Location Sent Every 3 Seconds], Maintain [Ambulance Location Known When Sent]

Maintain [Accurate Ambulance Availability Known]

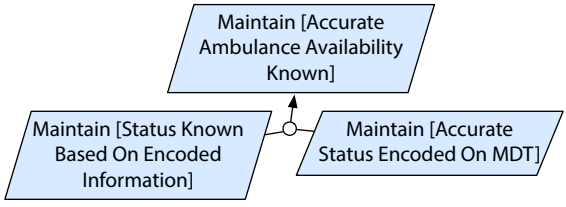


Figure A.15 Refinements for Maintain [Accurate Ambulance Availability Known].

Goal Maintain [Status Known Based On Encoded Information]
Def Availability of the ambulance shall be known to the dispatching software when encoded by the ambulance staff.
RefinedBy Maintain [Status Known Based On Encoded Status And Accurate Mapping], Accurate MDT Mapping

Goal Maintain [Accurate Status Encoded On MDT]

Def Accurate status information regarding the availability shall be encoded on the Mobile Data Terminal, i.e. staff is expected to press the button corresponding to their status.

RefinedBy Maintain [Status Leaving Encoded When Leaving Station], Maintain [Status AvailableRadio Encoded When Available by Radio], Maintain [Status AvailableStation Encoded When Available at Station], status__unavailable_encoded

Maintain [Accurate Status Encoded On MDT]

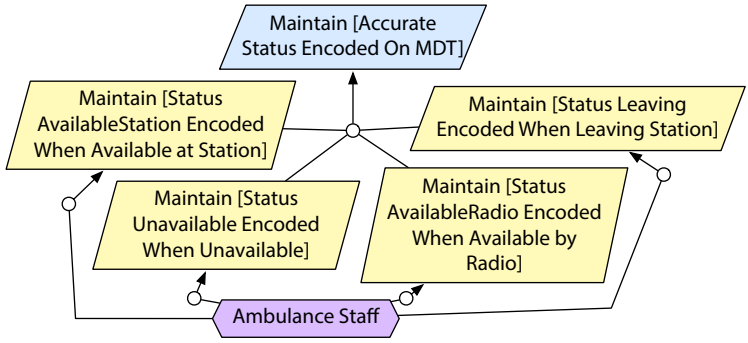


Figure A.16 Refinements for Maintain [Accurate Status Encoded On MDT].

Maintain [Status Known Based On Encoded Information]

Goal Maintain [Status Known Based On Encoded Status And Accurate Mapping]

Def Availability of the ambulance shall be known to the dispatching software when encoded by the ambulance staff, based on the mapping between the MDTs and the ambulances.

RefinedBy Achieve [Status Sent When Encoded], Achieve [Status Transmitted], Maintain [Status Recorded]

Maintain [Accurate Ambulance Location Known]

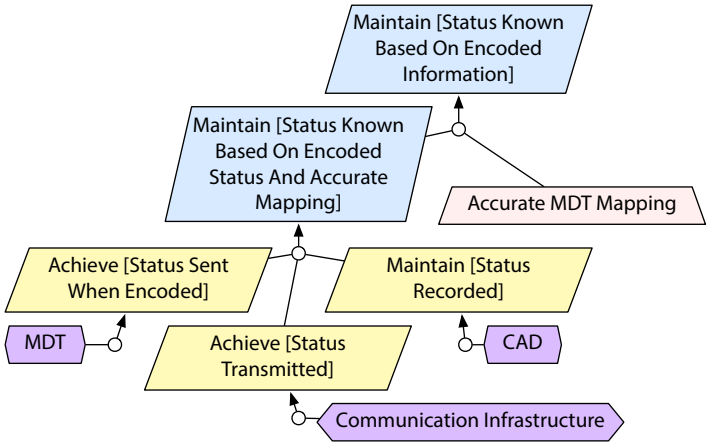


Figure A.17 Refinements for Maintain [Status Known Based On Encoded Information].

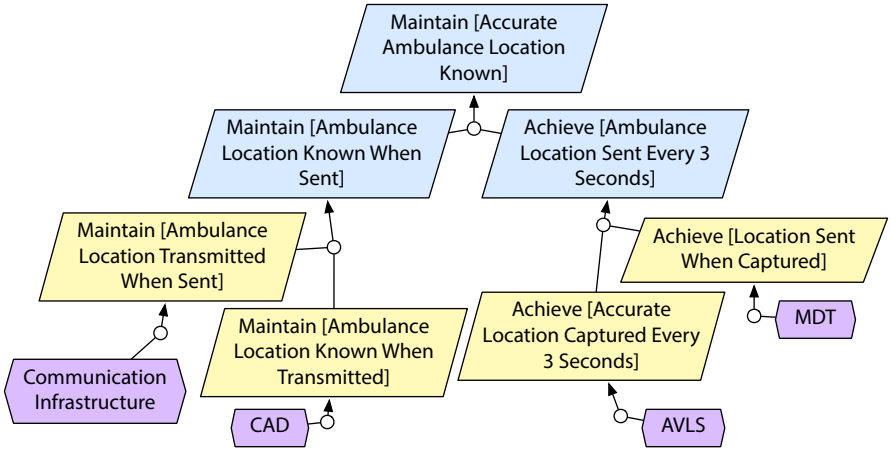


Figure A.18 Refinements for Maintain [Accurate Ambulance Location Known].

Goal Maintain [Ambulance Location Known When Sent]

Def The ambulance location is known to the dispatching software when sent.

RefinedBy Maintain [Ambulance Location Transmitted When Sent], Maintain [Ambulance Location Known When Transmitted]

Goal Achieve [Ambulance Location Sent Every 3 Seconds]
Def The ambulance location is sent every 3 seconds.
RefinedBy Achieve [Accurate Location Captured Every 3 Seconds], Achieve [Location Sent When Captured]

Goal Maintain [Ambulance Location Transmitted When Sent]
Def The ambulance location shall be timely transmitted to the despatching software when sent.
AssignedTo Communication Infrastructure

Goal Maintain [Ambulance Location Known When Transmitted]
Def The ambulance location shall be known when transmitted to the despatching software.
AssignedTo CAD

Goal Achieve [Accurate Location Captured Every 3 Seconds]
Def Accurate ambulance location is captured every 3 seconds. A captured location is accurate if it do not differ by more than 50 meters from the real position.
ObstructedBy Accurate Location Not Captured Every 3 Seconds
AssignedTo AVLS

Goal Achieve [Location Sent When Captured]
Def The ambulance location is sent when captured.
ObstructedBy Captured Ambulance Location Not Sent
AssignedTo MDT

Achieve [Ambulance Allocated Based On Incident Form]

Goal Achieve [Ambulance Allocated Based On Incident Form When Ambulance Available]
Def An available ambulance shall be allocated based on the informations on the incident form, within 1 minute.
RefinedBy Maintain [Accurate Ambulance Info Known], Achieve [Ambulance Allocation Based On Incident Form And Ambulance Info When Ambulance Available]

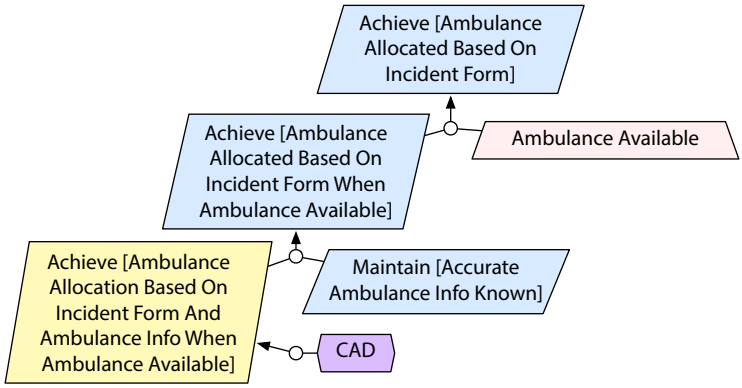


Figure A.19 Refinements for Maintain [Accurate Ambulance Location Known].

Goal Achieve [Ambulance Allocation Based On Incident Form And Ambulance Info When Ambulance Available]

Def An available ambulance shall be allocated within 1 minute when an incident is reported, based on the incident informations reported in the incident form and the ambulance availability information.

AssignedTo CAD

Goal Maintain [Accurate Ambulance Info Known]

Def Accurate availability and accurate location about the ambulance shall be known to the software system.

RefinedBy Maintain [Accurate Ambulance Location Known], Maintain [Accurate Ambulance Availability Known]

Achieve [Ambulance Left Station When Mobilized]

Goal Achieve [Ambulance Physically Leaving Station When Mobilized]

Def The mobilized ambulance shall leave the ambulance station within 2 minutes.

AssignedTo Ambulance Staff

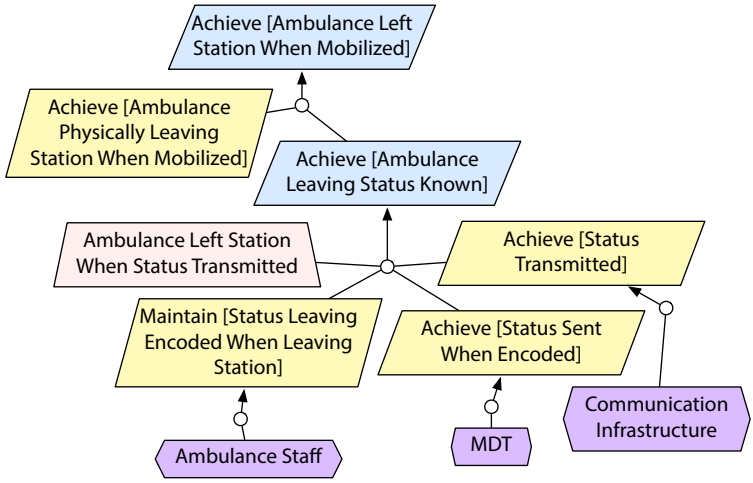


Figure A.20 Refinements for Maintain [Accurate Ambulance Location Known].

Goal Achieve [Ambulance Leaving Status Known]
Def The ambulance leaving status is known to the dispatching software when the mobilized ambulance left the ambulance station.
RefinedBy Maintain [Status Leaving Encoded When Leaving Station], Achieve [Status Sent When Encoded], Achieve [Status Transmitted], Ambulance Left Station When Status Transmitted

A.1.2 Obstacle specifications

This section provides the complete specifications for the obstacles of the Brussels Ambulance Despatch System, presented in Section 9.3.3.

Achieve [Location Sent When Captured]

Goal Captured Ambulance Location Not Sent
Def The captured ambulance location is not sent.
RefinedBy MDT Turned Off

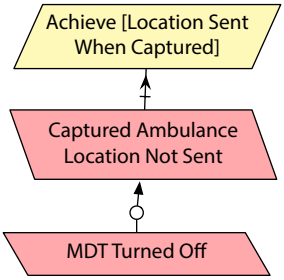


Figure A.21 Obstacles to Achieve [Location Sent When Captured].

Goal MDT Turned Off

Def The MDT is turned off for more than 5 seconds and mobilization order was transmitted.

Achieve [Allocated Ambulance Mobilization When Mobilization Order Displayed And Radio Contact]

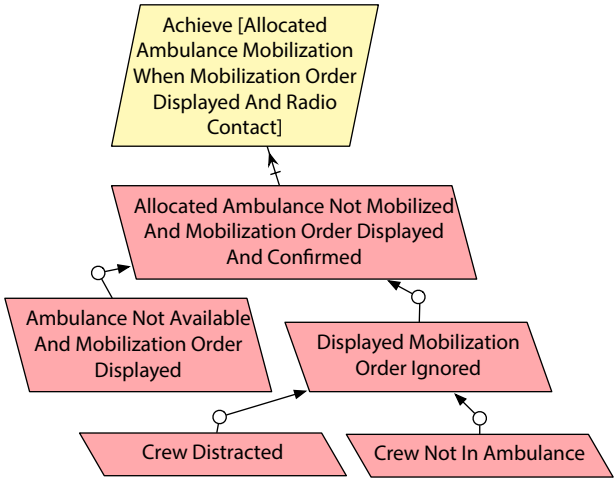


Figure A.22 Obstacles to Achieve [Allocated Ambulance Mobilization When Mobilization Order Displayed And Radio Contact].

Goal Allocated Ambulance Not Mobilized And Mobilization Order Displayed And Confirmed
Def The allocated ambulance is not mobilized when the mobilization order is displayed on the MDT and the mobilization is confirmed by radio. The mobilization did not occured within 10 seconds of the print or confirmation.
RefinedBy Ambulance Not Available And Mobilization Order Displayed
RefinedBy Displayed Mobilization Order Ignored

Goal Ambulance Not Available And Mobilization Order Displayed
Def The ambulance is not available even if the mobilizatin order is displayed and the mobilization order is confirmed.

Goal Displayed Mobilization Order Ignored
Def The displayed mobilization order is ignored by the ambulance staff.
RefinedBy Crew Distracted
RefinedBy Crew Not In Ambulance

Goal Crew Distracted
Def The ambulance staff is distracted and ignores the displayed mobilization order.

Goal Crew Not In Ambulance
Def The ambulance staff is not in the ambulance and is therefore not able to read the displayed mobilization order.

Achieve [Mobilization Order Transmitted]

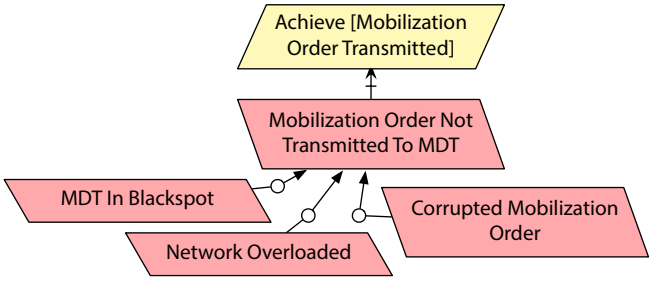


Figure A.23 Obstacles to Achieve [Mobilization Order Transmitted].

Goal Mobilization Order Not Transmitted To MDT
Def
RefinedBy Network Overloaded
RefinedBy MDT In Blackspot
RefinedBy Corrupted Mobilization Order

Goal MDT In Blackspot
Def The MDT that shall received the information is in a blackspot and no longer able to send and receive data.

Goal Corrupted Mobilization Order
Def The mobilization order is corrupted and cannot be transmitted reliably to the corresponding MDT.

Goal Network Overloaded
Def The network used to transmit information between the dispatching software and the MDTs is overloaded and not able to transmit information.

Achieve [Allocated Ambulance Mobilization When Mobilization Order Printed And Phone Contact]

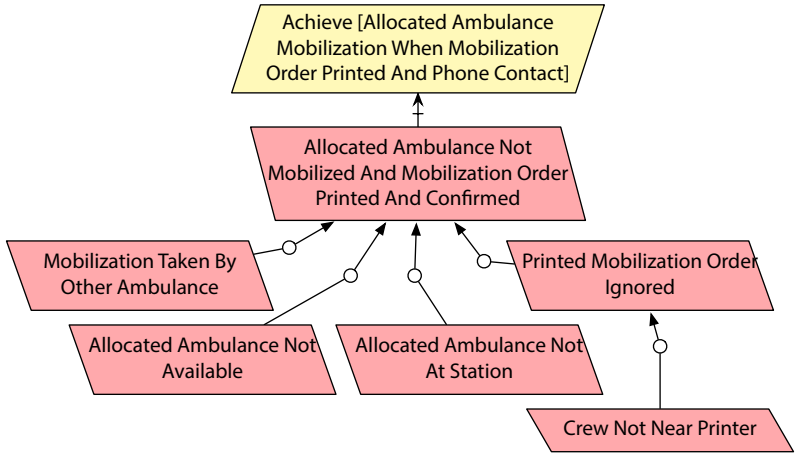


Figure A.24 Obstacles to Achieve [Allocated Ambulance Mobilization When Mobilization Order Printed And Phone Contact].

Goal Allocated Ambulance Not Mobilized And Mobilization Order Printed And Confirmed
Def The allocated ambulance is not mobilized but the mobilization order is printed and the mobilization is confirmed by the operator.
RefinedBy Allocated Ambulance Not At Station
RefinedBy Allocated Ambulance Not Available
RefinedBy Printed Mobilization Order Ignored
RefinedBy Mobilization Taken By Other Ambulance

Goal Allocated Ambulance Not At Station
Def The ambulance is not at the station when the mobilization order is printed and/or the call is made.

Goal Allocated Ambulance Not Available
Def The ambulance is at the station but not yet available (for example, cleaning and refurbishing the ambulance).

Goal Printed Mobilization Order Ignored
Def The printed mobilization order is ignored by the ambulance staff.
RefinedBy Crew Not Near Printer

Goal Crew Not Near Printer
Def The ambulance staff is not near the station printer.

Goal Mobilization Taken By Other Ambulance
Def The mobilization order is taken by an other ambulance staff.

Achieve [Patient Transported At Hospital When Required]

Goal Patient Not Admitted At Or Transported To Hospital
Def The patient is not admitted or transported to the nearest and most appropriate hospital.
RefinedBy Patient Not Transported
RefinedBy Patient Not Admitted At Hospital

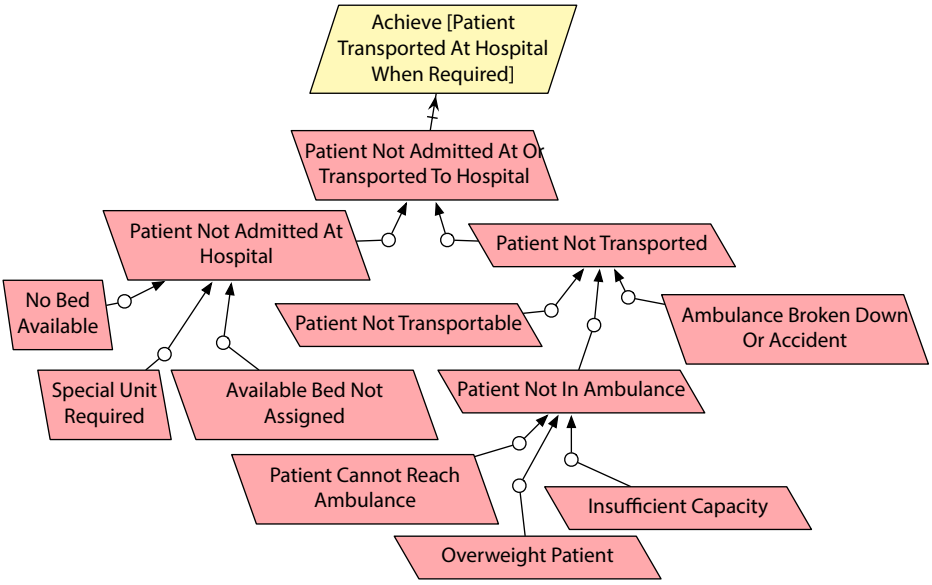


Figure A.25 Obstacles to Achieve [Patient Transported At Hospital When Required].

Goal Patient Not Transported
Def The patient is not transported to the nearest and most appropriate hospital.
RefinedBy Patient Not In Ambulance
RefinedBy Ambulance Broken Down Or Accident
RefinedBy Patient Not Transportable

Goal Patient Not In Ambulance
Def The patient is not in the ambulance but should be transported to the hospital.
RefinedBy Insufficient Capacity
RefinedBy Overweight Patient
RefinedBy Patient Cannot Reach Ambulance

Goal Patient Not Transportable
Def The patient is not safely transportable from the incident scene to the nearest and most appropriate hospital.

Goal Insufficient Capacity
Def The ambulance capacity is reached. Patient cannot be transported safely.

Goal Overweight Patient
Def The weight of the patient is over the maximal weight allowed for safe transportation by the ambulance.

Goal Patient Cannot Reach Ambulance
Def The patient cannot safely reach the ambulance.

Goal Patient Not Admitted At Hospital
Def The patient is not admitted at the nearest and most appropriate hospital.
RefinedBy No Bed Available
RefinedBy Available Bed Not Assigned
RefinedBy Special Unit Required

Goal Available Bed Not Assigned
Def The patient is not admitted at the nearest and most appropriate hospital, even if beds are available.

Goal Special Unit Required
Def A special equipment or specific staff is required to admit the patient at the hospital and such equipment or staff is not available at nearest and most appropriate hospital.

Goal No Bed Available
Def The nearest and most appropriate hospital has no available beds for the patient.

Achieve [Mobilized Ambulance On Scene]

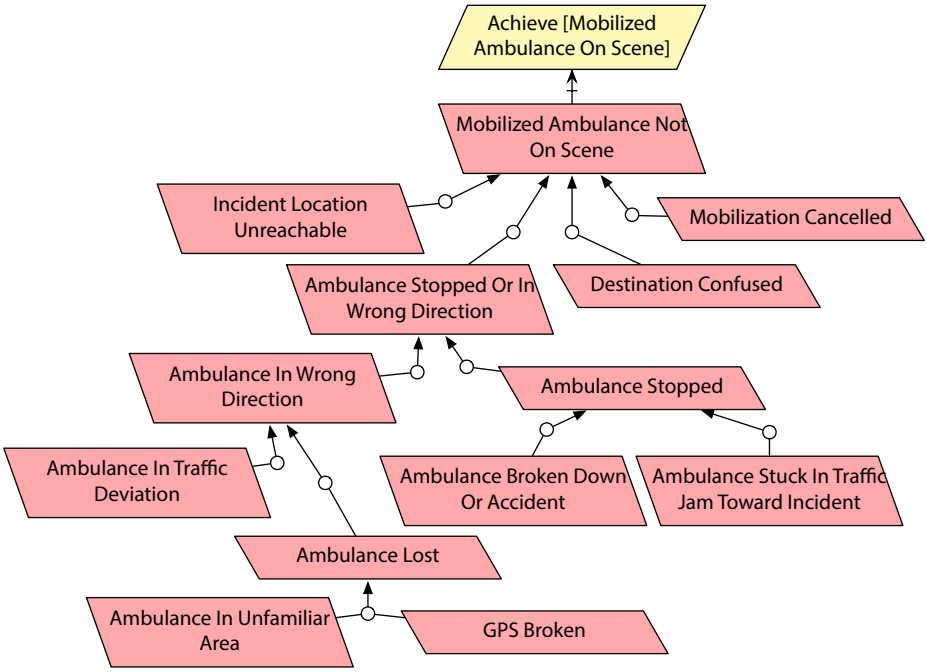


Figure A.26 Obstacles to Achieve [Mobilized Ambulance On Scene].

Goal Mobilized Ambulance Not On Scene
Def The mobilized ambulance is not on the incident scene within 12 minutes.
RefinedBy Mobilization Cancelled
RefinedBy Destination Confused
RefinedBy Ambulance Stopped Or In Wrong Direction
RefinedBy Incident Location Unreachable

Goal Incident Location Unreachable
Def The incident location is not safely reachable by the ambulance staff.

Goal Destination Confused
Def The incident location is confused, and the ambulance is not on scene within 12 minutes.

Goal Mobilization Cancelled
Def The mobilization is cancelled.

Goal Ambulance Stopped Or In Wrong Direction
Def The ambulance is stopped or in the wrong direction and will not reach the incident scene within the time constraints.
RefinedBy Ambulance Stopped
RefinedBy Ambulance In Wrong Direction

Goal Ambulance Stopped
Def The ambulance is stopped and will not reach the incident scene within the time constraints.
RefinedBy Ambulance Broken Down Or Accident
RefinedBy Ambulance Stuck In Traffic Jam Toward Incident

Goal Ambulance Broken Down Or Accident
Def The ambulance is broken down or involved in an accident and will not reach the incident scene within the time constraints.

Goal Ambulance Stuck In Traffic Jam Toward Incident
Def The ambulance is stuck in a traffic jam and will not reach the incident scene within the time constraints.

Goal Ambulance In Wrong Direction
Def The ambulance is headed in the wrong direction and will not reach the incident scene within the time constraints.
RefinedBy Ambulance Lost
RefinedBy Ambulance In Traffic Deviation

Goal Ambulance Lost
Def The ambulance is lost and will not reach the incident scene within the time constraints.
RefinedBy Ambulance In Unfamiliar Area, GPS Broken

Goal Ambulance In Unfamiliar Area
Def The ambulance is in an unfamiliar area.

Goal GPS Broken
Def The GPS of the ambulance driver break down.

Goal Ambulance In Traffic Deviation
Def The ambulance is in a traffic deviation and will not reach the incident scene within the time constraints.

Achieve [Incident Resolved On Scene When No Transport Required]

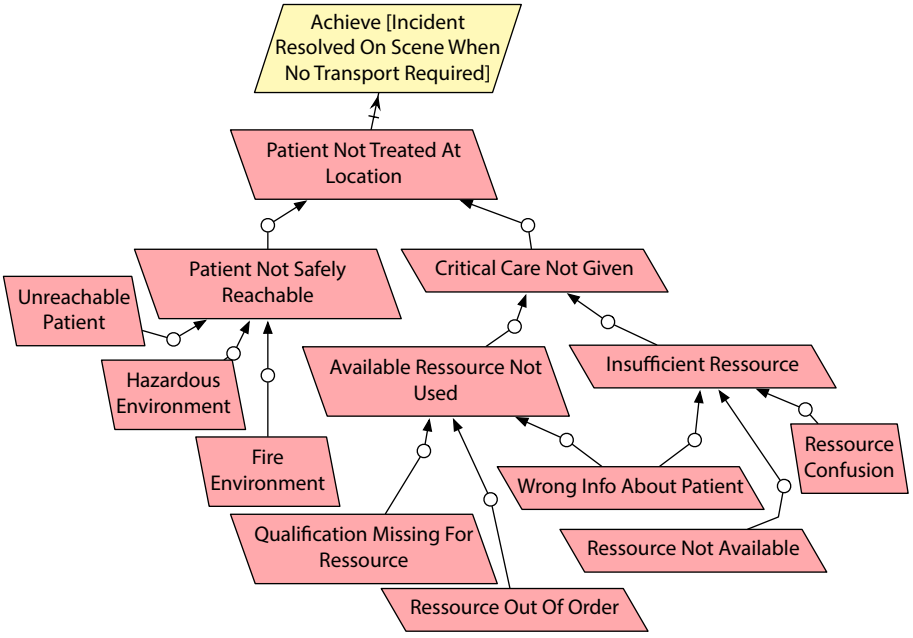


Figure A.27 Obstacles to Achieve [Incident Resolved On Scene When No Transport Required].

Goal Patient Not Treated At Location
Def The patient was not treated at the incident location.
RefinedBy Critical Care Not Given
RefinedBy Patient Not Safely Reachable

Goal Patient Not Safely Reachable
Def The patient was not reachable safely by the ambulance staff.
RefinedBy Fire Environment
RefinedBy Hazardous Environment
RefinedBy Unreachable Patient

Goal Fire Environment
Def The patient was not reachable safely by the ambulance staff due to a fiery environment.

Goal Hazardous Environment
Def The patient was not reachable safely by the ambulance staff due to a hazardous environment exluding fire.

Goal Unreachable Patient
Def The patient was not reachable safely by the ambulance staff but patient in a safe environment (e.g. locked in a lift).

Goal Critical Care Not Given
Def Critical care at location is not given to the patient.
RefinedBy Insufficient Ressource
RefinedBy Available Ressource Not Used

Goal Available Ressource Not Used
Def Available ressource required to care for the patient is not used.
RefinedBy Wrong Info About Patient
RefinedBy Ressource Out Of Order
RefinedBy Qualification Missing For Ressource

Goal Ressource Out Of Order
Def Available ressource required to care for the patient is out of order.

Goal Qualification Missing For Ressource
Def The staff does not have the required qualifications to provide the critical care required by the patient condition. For example, the ambulance staff is not allowed to deliver a drug but has the drug at disposal.

Goal Insufficient Ressource
Def Ressource required by the patient condition is not available in sufficient quantities.
RefinedBy Ressource Confusion
RefinedBy Ressource Not Available
RefinedBy Wrong Info About Patient

Goal Ressource Confusion
Def Ressource required by the patient condition is not used due to confusion.

Goal Ressource Not Available
Def Ressource required by the patient condition is not available.

Goal Wrong Info About Patient
Def Critical care is not provided to the patient as information is missing on his condition. This includes interventions were additional care would have been given if, for example, medical history, allergies, drug prescriptions, etc. was known.

Maintain [Status Unavailable Encoded When Unavailable]

Goal Accurate Unavailable Status Not Encoded On MDT Or Innaccurate
Def Accurate Unavailable status information is not encoded on the Mobile Data Terminal when the status changes; or the encoded status is not accurate.
RefinedBy Forget To Encode Unavailable Status
RefinedBy Other Status Than Unavailable Encoded

Goal Forget To Encode Unavailable Status
Def The ambulance staff forgets to encode the Unavailable status.

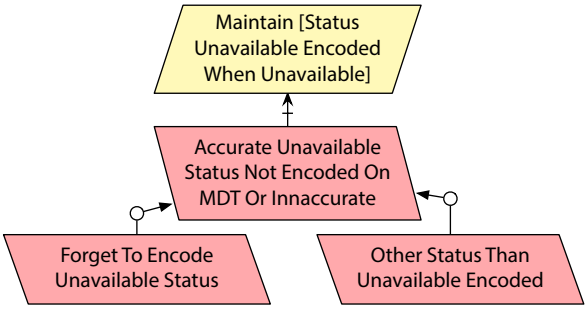


Figure A.28 Obstacles to Maintain [Status Unavailable Encoded When Unavailable].

Goal Other Status Than Unavailable Encoded

Def The ambulance staff encodes an other status than the Unavailable status when required.

Maintain [Status AvailableStation Encoded When Available at Station]

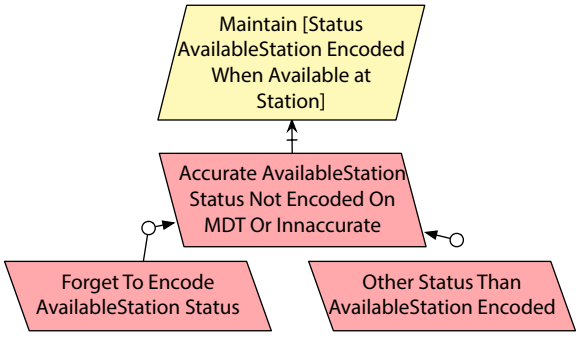


Figure A.29 Obstacles to Maintain [Status AvailableStation Encoded When Available at Station].

Goal Accurate AvailableStation Status Not Encoded On MDT Or Innaccurate
Def Accurate AvailableStation status information is not encoded on the Mobile Data Terminal when the status changes; or the encoded status is not accurate.
RefinedBy Forget To Encode AvailableStation Status
RefinedBy Other Status Than AvailableStation Encoded

Goal Forget To Encode AvailableStation Status
Def The ambulance staff forgets to encode the AvailableStation status.

Goal Other Status Than AvailableStation Encoded
Def The ambulance staff encodes an other status than the AvailableStation status when required.

Maintain [Status AvailableRadio Encoded When Available by Radio]

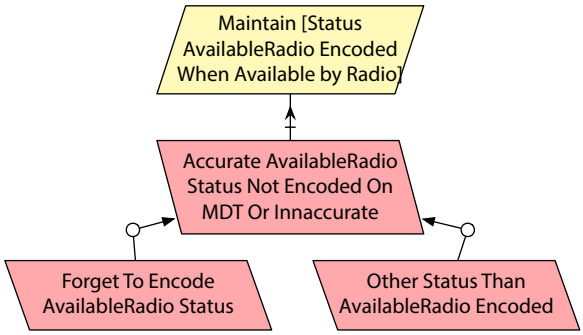


Figure A.30 Obstacles to Maintain [Status AvailableRadio Encoded When Available by Radio].

Goal Accurate AvailableRadio Status Not Encoded On MDT Or Innaccurate
Def Accurate AvailableRadio status information is not encoded on the Mobile Data Terminal when the status changes; or the encoded status is not accurate.
RefinedBy Forget To Encode AvailableRadio Status
RefinedBy Other Status Than AvailableRadio Encoded

Goal Forget To Encode AvailableRadio Status
Def The ambulance staff forgets to encode the AvailableRadio status.

Goal Other Status Than AvailableRadio Encoded
Def The ambulance staff encodes an other status than the AvailableRadio status when required.

Maintain [Status AtHospital Encoded When At Hospital]

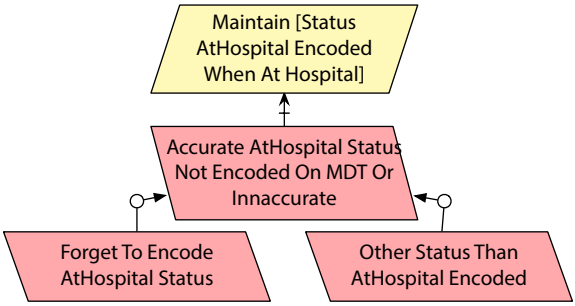


Figure A.31 Obstacles to Maintain [Status AtHospital Encoded When At Hospital].

Goal Accurate AtHospital Status Not Encoded On MDT Or Innaccurate
Def Accurate AtHospital status information is not encoded on the Mobile Data Terminal when the status changes; or the encoded status is not accurate.
RefinedBy Forget To Encode AtHospital Status
RefinedBy Other Status Than AtHospital Encoded

Goal Forget To Encode AtHospital Status
Def The ambulance staff forgets to encode the AtHospital status.

Goal Other Status Than AtHospital Encoded
Def The ambulance staff encodes an other status than the AtHospital status when required.

Maintain [Status ToHospital Encoded When Leaving Towards Hospital]

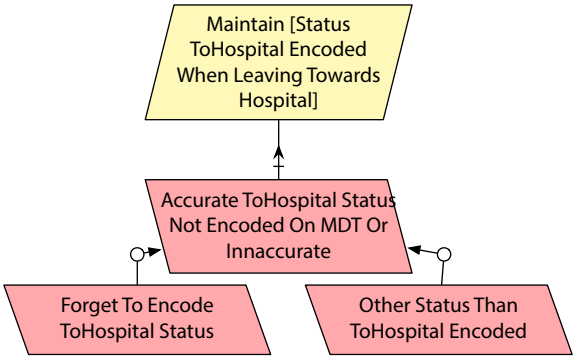


Figure A.32 Obstacles to Maintain [Status ToHospital Encoded When Leaving Towards Hospital].

Goal Accurate ToHospital Status Not Encoded On MDT Or Innaccurate
Def Accurate ToHospital status information is not encoded on the Mobile Data Terminal when the status changes; or the encoded status is not accurate.
RefinedBy Forget To Encode ToHospital Status
RefinedBy Other Status Than ToHospital Encoded

Goal Forget To Encode ToHospital Status
Def The ambulance staff forgets to encode the ToHospital status.

Goal Other Status Than ToHospital Encoded
Def The ambulance staff encodes an other status than the ToHospital status when required.

Maintain [Status OnScene Encoded When On Scene]

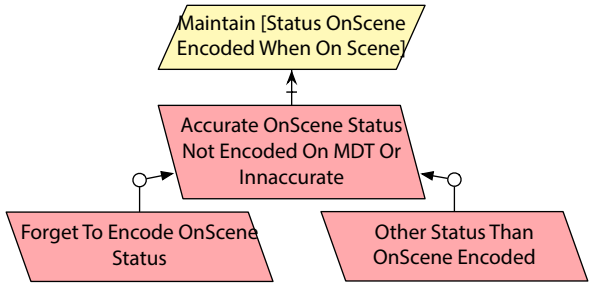


Figure A.33 Obstacles to Maintain [Status OnScene Encoded When On Scene].

Goal Accurate OnScene Status Not Encoded On MDT Or Innaccurate
Def Accurate OnScene status information is not encoded on the Mobile Data Terminal when the status changes; or the encoded status is not accurate.
RefinedBy Forget To Encode OnScene Status
RefinedBy Other Status Than OnScene Encoded

Goal Forget To Encode OnScene Status
Def The ambulance staff forgets to encode the OnScene status.

Goal Other Status Than OnScene Encoded
Def The ambulance staff encodes an other status than the OnScene status when required.

Maintain [Status Leaving Encoded When Leaving Station]

Goal Accurate Leaving Status Not Encoded On MDT Or Innaccurate
Def Accurate Leaving status information is not encoded on the Mobile Data Terminal when the status changes; or the encoded status is not accurate.
RefinedBy Forget To Encode Leaving Status
RefinedBy Other Status Than Leaving Encoded

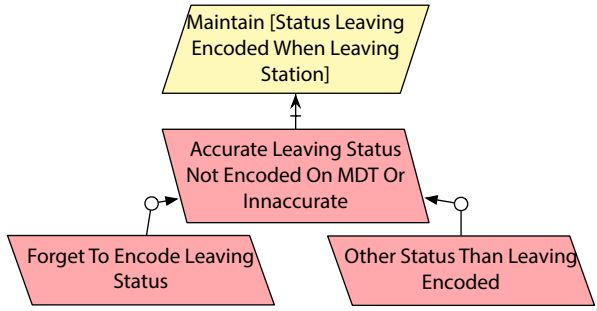


Figure A.34 Obstacles to Maintain [Status Leaving Encoded When Leaving Station].

Goal Forget To Encode Leaving Status
Def The ambulance staff forgets to encode the Leaving status.

Goal Other Status Than Leaving Encoded
Def The ambulance staff encodes an other status than the Leaving status when required.

Achieve [Accurate Location Captured Every 3 Seconds]

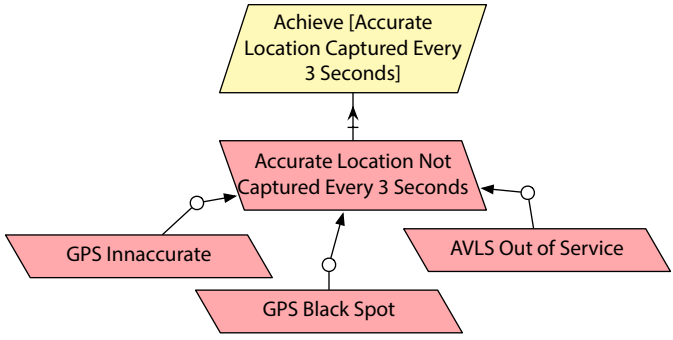


Figure A.35 Obstacles to Achieve [Accurate Location Captured Every 3 Seconds].

Goal Accurate Location Not Captured Every 3 Seconds
Def Accurate ambulance location was not captured every 3 seconds.
RefinedBy GPS Black Spot
RefinedBy GPS Innaccurate
RefinedBy AVLS Out of Service

Goal AVLS Out of Service
Def The AVLS component in the ambulance is out of service.

Goal GPS Black Spot
Def The ambulance is in a black spot for more than 3 seconds.

Goal GPS Innaccurate
Def The GPS position differs by more than 50 meters for more than 3 seconds.

Achieve [Mobilization Order Printed]

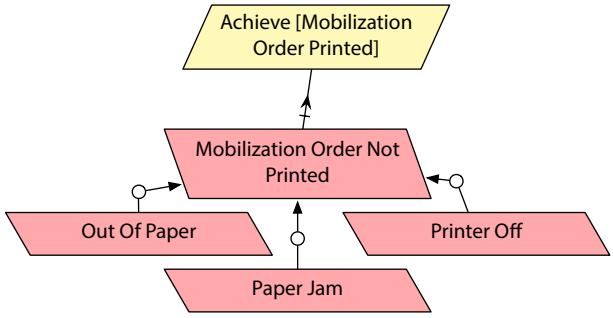


Figure A.36 Obstacles to Achieve [Mobilization Order Printed].

Goal Mobilization Order Not Printed
Def The mobilization order is not printed within 50 seconds at the allocated ambulance station.
RefinedBy Paper Jam
RefinedBy Out Of Paper
RefinedBy Printer Off

Goal Paper Jam
Def There is a paper jam for more than 50 seconds in the allocated ambulance station printer.

Goal Out Of Paper
Def There is no paper for more than 50 seconds in the allocated ambulance station printer.

Goal Printer Off
Def The allocated ambulance station printer is turned off for more than 50 seconds.

Achieve [Mobilization Order Displayed]

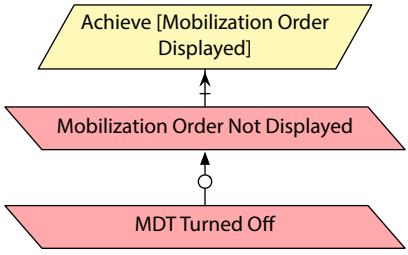


Figure A.37 Obstacles to Achieve [Mobilization Order Displayed].

Goal Mobilization Order Not Displayed
Def The transmitted mobilization order is not displayed within 5 seconds.
RefinedBy MDT Turned Off

A.2 Case-Study: Car Pooling System

This section provides the complete specifications for the Car Pooling System, presented in Section 9.3.1.

A.2.1 Goal specifications

This section provides the complete specifications for the goals of the Car Pooling System, presented in Section 9.3.1.

Achieve [Ride Proposal Eventually Served]

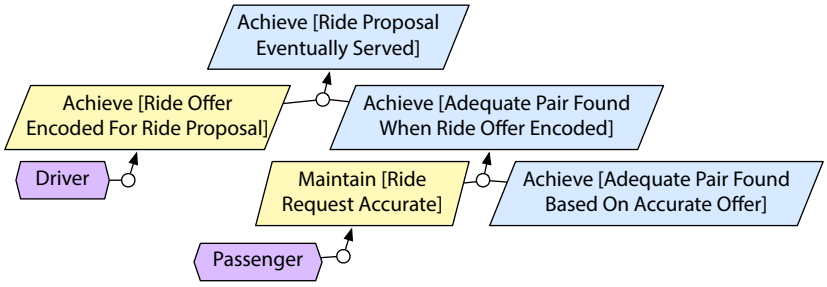


Figure A.38 Refinements for Achieve [Ride Proposal Eventually Served].

Goal Achieve [Ride Proposal Eventually Served]
Def Rides proposed by drivers shall be granted within time constraints. By granted, we mean that the driver did not travel alone.
RSR 0.95
RefinedBy Achieve [Ride Offer Encoded For Ride Proposal], Achieve [Adequate Pair Found When Ride Offer Encoded]

Goal Achieve [Ride Offer Encoded For Ride Proposal]
Def For any needed request, the request shall eventually be encoded in the system.
AssignedTo Driver

Goal Achieve [Adequate Pair Found When Ride Offer Encoded]
Def A compatible pair containing a request and an offer shall eventually be found for any encoded offer.
RefinedBy Maintain [Ride Request Accurate], Achieve [Adequate Pair Found Based On Accurate Offer]

Goal Maintain [Ride Request Accurate]
Def Ride request shall always be accurate and known by the system.
ObstructedBy Ride Request Not Accurate
AssignedTo Passenger

Goal Achieve [Adequate Pair Found Based On Accurate Offer]
Def Based on accurate offers known by the system, an offer compatible with the request shall eventually be found when the request has been encoded.
RefinedBy Achieve [Matching Suggestion Selected], Achieve [Suggestion Elected Based On Selection], Pair Found IIf Pair Elected

Achieve [Ride Need Eventually Served]

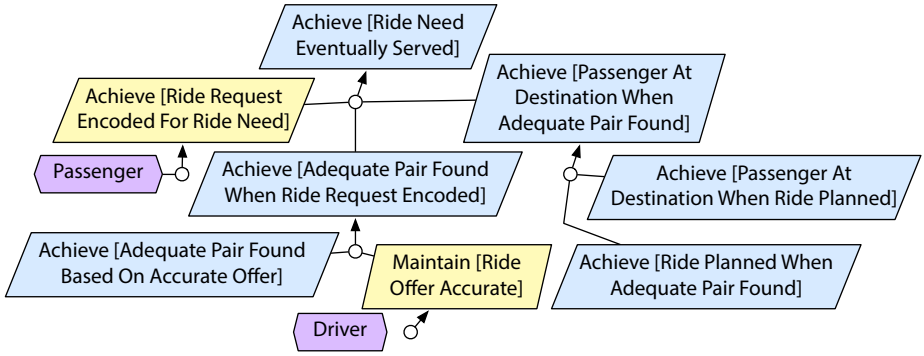


Figure A.39 Refinements for Achieve [Ride Need Eventually Served].

Goal Achieve [Ride Need Eventually Served]
Def Passengers that want to go to a destination shall arrive at that destination within the specified time constraints
RSR 0.95
RefinedBy Achieve [Ride Request Encoded For Ride Need], Achieve [Adequate Pair Found When Ride Request Encoded], Achieve [Passenger At Destination When Adequate Pair Found]

Goal Achieve [Ride Request Encoded For Ride Need]
Def For any needed request, the request shall eventually be encoded in the system.
AssignedTo Passenger

Goal Achieve [Adequate Pair Found When Ride Request Encoded]
Def A compatible pair containing a request and an offer shall eventually be found for any encoded request.
RefinedBy Maintain [Ride Offer Accurate], Achieve [Adequate Pair Found Based On Accurate Offer]

Goal Achieve [Passenger At Destination When Adequate Pair Found]
Def Passengers shall eventually arrive at destination when a compatible pair of request and offer has been found.
RefinedBy Achieve [Ride Planned When Adequate Pair Found], Achieve [Passenger At Destination When Ride Planned]

Goal Achieve [Adequate Pair Found Based On Accurate Offer]
Def Based on accurate offers known by the system, an offer compatible with the request shall eventually be found when the request has been encoded.
RefinedBy Achieve [Matching Suggestion Selected], Achieve [Suggestion Elected Based On Selection], Pair Found IIf Pair Elected

Goal Maintain [Ride Offer Accurate]
Def Ride offers proposed by the drivers shall always be as accurate as possible.
ObstructedBy Ride Offer Not Accurate
AssignedTo Driver

Goal Achieve [Ride Planned When Adequate Pair Found]
Def For any ride for which a pair was found, the ride shall eventually be planned.
RefinedBy Achieve [Ride Planned When Pair Elected], Pair Found IIf Pair Elected

Goal Achieve [Passenger At Destination When Ride Planned]
Def For any planned ride, the corresponding passenger is eventually at the destination specified in her request.
RefinedBy Achieve [Ride Instructions Known By Riders When Ride Planned], Achieve [Passenger At Destination When Instructions Known]

Achieve [Passenger At Destination When Ride Planned]

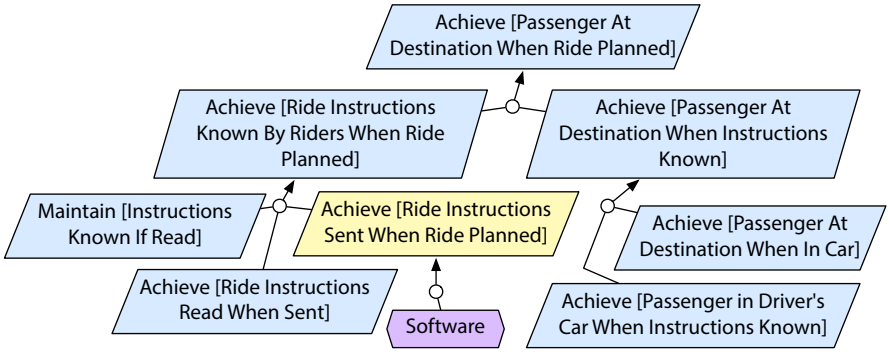


Figure A.40 Refinements for Achieve [Passenger At Destination When Ride Planned].

Goal Achieve [Passenger At Destination When Ride Planned]
Def For any planned ride, the corresponding passenger is eventually at the destination specified in her request.
RefinedBy Achieve [Ride Instructions Known By Riders When Ride Planned], Achieve [Passenger At Destination When Instructions Known]

Goal Achieve [Ride Instructions Known By Riders When Ride Planned]
Def For any planned ride, the instructions are known by both riders when the ride is planned.
RefinedBy Achieve [Ride Instructions Sent When Ride Planned], Achieve [Ride Instructions Read When Sent], Maintain [Instructions Known If Read]

Goal Achieve [Passenger At Destination When Instructions Known]
Def For any ride for which instructions are known by both driver and passenger, the passenger shall eventually arrive at the destination specified in her request.
RefinedBy Achieve [Passenger in Driver's Car When Instructions Known], Achieve [Passenger At Destination When In Car]

Goal Achieve [Ride Instructions Sent When Ride Planned]
Def For any planned ride, instructions are sent to corresponding passenger and driver.
ObstructedBy Ride Instructions Not Sent When Ride Planned
AssignedTo Software

Goal Achieve [Ride Instructions Read When Sent]
Def For any sent instructions about a ride involving the recipient of the instructions, the recipient shall eventually read these.
RefinedBy Achieve [Ride Instructions Read By Passenger], Achieve [Ride Instructions Read By Driver]

Goal Maintain [Instructions Known If Read]
Def For any read instructions about a ride involving the recipient of the instructions, these shall remain known forever by the rider.
RefinedBy Maintain [Instructions Known By Passenger], Maintain [Instructions Known By Driver]

Goal Achieve [Passenger in Driver's Car When Instructions Known]
Def For any ride for which instructions are known by both passenger and driver, the passenger shall eventually take a seat in driver's car.
RefinedBy Achieve [Riders At Pickup Point When Instructions Known], Achieve [Passenger In Driver's Car If Riders At Pickup Point]

Goal Achieve [Passenger At Destination When In Car]
Def For any ride for which the passenger is in driver's car, that passenger shall eventually arrive at destination specified in the corresponding request.
RefinedBy Achieve [Drop Point Reached When Passenger In Car], Achieve [Passenger At Destination When Drop Point Reached]

Achieve [Adequate Pair Found Based On Accurate Offer] and Achieve [Ride Planned When Adequate Pair Found]

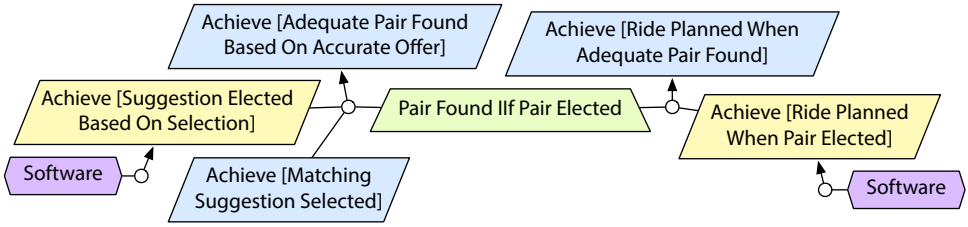


Figure A.41 Refinements for Achieve [Adequate Pair Found Based On Accurate Offer] and Achieve [Ride Planned When Adequate Pair Found].

Goal Achieve [Adequate Pair Found Based On Accurate Offer]
Def Based on accurate offers known by the system, an offer compatible with the request shall eventually be found when the request has been encoded.
RefinedBy Achieve [Matching Suggestion Selected], Achieve [Suggestion Elected Based On Selection], Pair Found IIf Pair Elected

Goal Achieve [Ride Planned When Adequate Pair Found]
Def For any ride for which a pair was found, the ride shall eventually be planned.
RefinedBy Achieve [Ride Planned When Pair Elected], Pair Found IIf Pair Elected

Goal Achieve [Suggestion Elected Based On Selection]
Def For any encoded need, when at least one suggestion has been selected, one among them shall be elected.
AssignedTo Software

Goal Achieve [Matching Suggestion Selected]
Def The passenger and a driver shall eventually select at least one suggestion matching the request when this request has been encoded.
RefinedBy Achieve [Suggestion Selected], Maintain [Suggestion Matching]

DomProp Pair Found IIf Pair Elected
Def A pair request/offer is considered as found when a suggestion serving both the request and the offer has been elected.

Goal Achieve [Ride Planned When Pair Elected]
Def For any ride for which a pair has been elected, the ride shall eventually be planned.
ObstructedBy Ride Not Planned And Pair Elected
AssignedTo Software

Maintain [Instructions Known If Read]

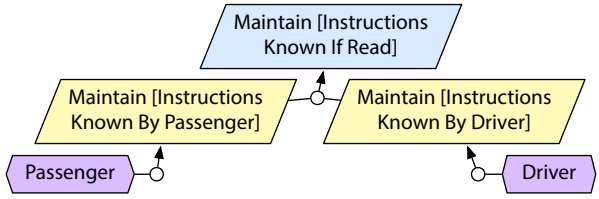


Figure A.42 Refinements for Maintain [Instructions Known If Read].

Goal Maintain [Instructions Known If Read]
Def For any read instructions about a ride involving the recipient of the instructions, these shall remain known forever by the rider.
RefinedBy Maintain [Instructions Known By Passenger], Maintain [Instructions Known By Driver]

Goal Maintain [Instructions Known By Passenger]
Def For any read instructions about a ride involving the passenger, instructions shall remain known forever by the passenger.
AssignedTo Passenger

Goal Maintain [Instructions Known By Driver]
Def For any read instructions about a ride involving the driver, these shall remain known forever by the driver.
AssignedTo Driver

Achieve [Ride Instructions Read When Sent]

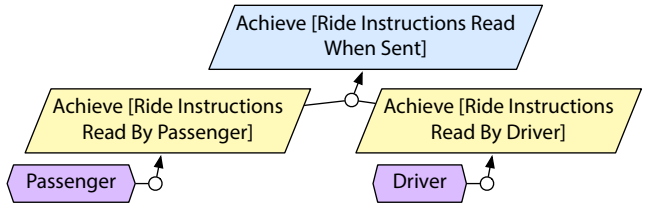


Figure A.43 Refinements for Achieve [Ride Instructions Read When Sent].

Goal Achieve [Ride Instructions Read When Sent]
Def For any sent instructions about a ride involving the recipient of the instructions, the recipient shall eventually read these.
RefinedBy Achieve [Ride Instructions Read By Passenger], Achieve [Ride Instructions Read By Driver]

Goal Achieve [Ride Instructions Read By Passenger]
Def For any sent instructions, the recipient shall eventually read the instructions.
ObstructedBy Ride Instructions Not Read By Passenger And Sent
AssignedTo Passenger

Goal Achieve [Ride Instructions Read By Driver]
Def For any sent instructions, the recipient shall eventually read the instructions.
ObstructedBy Ride Instructions Not Read By Driver And Sent
AssignedTo Driver

Achieve [Passenger At Destination When In Car]

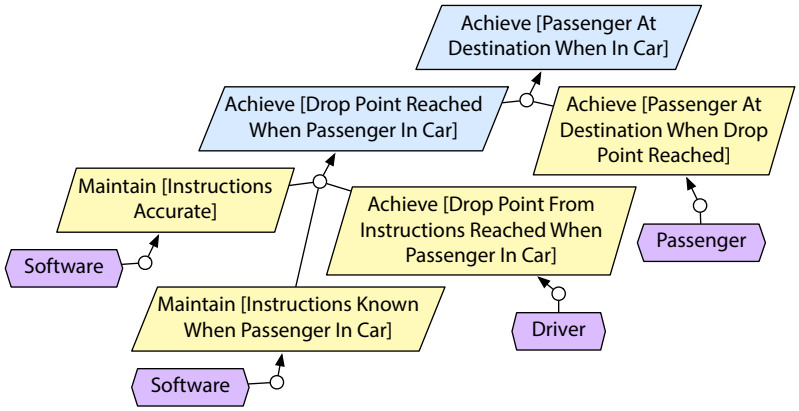


Figure A.44 Refinements for Achieve [Passenger At Destination When In Car].

Goal Achieve [Passenger At Destination When In Car]
Def For any ride for which the passenger is in driver's car, that passenger shall eventually arrive at destination specified in the corresponding request.
RefinedBy Achieve [Drop Point Reached When Passenger In Car], Achieve [Passenger At Destination When Drop Point Reached]

Goal Achieve [Drop Point Reached When Passenger In Car]
Def For any planned ride with a passenger and a driver, if the passenger is in driver's car then both shall eventually be at specified drop point.
RefinedBy Maintain [Instructions Accurate], Maintain [Instructions Known When Passenger In Car], Achieve [Drop Point From Instructions Reached When Passenger In Car]

Goal Achieve [Passenger At Destination When Drop Point Reached]
Def For any planned ride with a passenger at specified drop location, she shall eventually be at specified destination in her corresponding request.
AssignedTo Passenger

Goal Maintain [Instructions Accurate]
Def For any ride, the drop point specified in instructions equals the drop point computed for the ride.
AssignedTo Software

Goal Maintain [Instructions Known When Passenger In Car]
Def For any ride, instructions are known if the passenger is in driver's car.
AssignedTo Software

Goal Achieve [Drop Point From Instructions Reached When Passenger In Car]
Def For any ride, driver and passenger shall eventually arrive at drop point specified in corresponding instructions.
ObstructedBy Drop Point Not Reached When Passenger In Car
AssignedTo Driver

Achieve [Passenger in Driver's Car When Instructions Known]

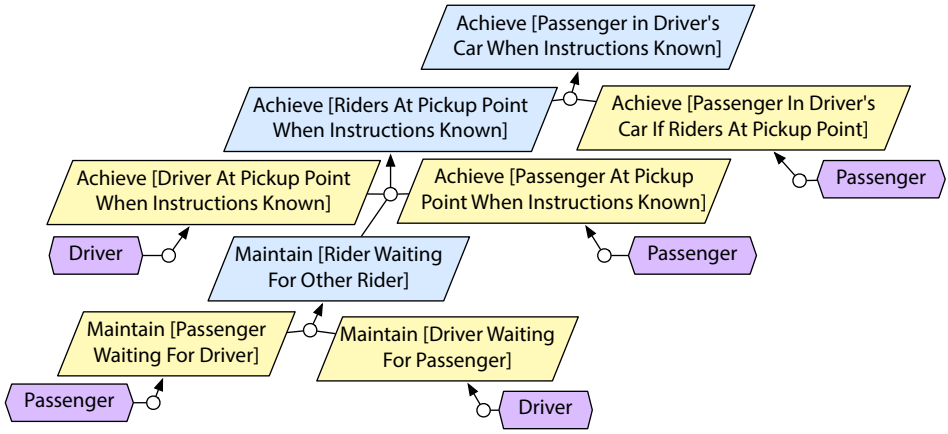


Figure A.45 Refinements for Achieve [Passenger in Driver's Car When Instructions Known].

Goal Achieve [Passenger in Driver's Car When Instructions Known]
Def For any ride for which instructions are known by both passenger and driver, the passenger shall eventually take a seat in driver's car.
RefinedBy Achieve [Riders At Pickup Point When Instructions Known], Achieve [Passenger In Driver's Car If Riders At Pickup Point]

Goal Achieve [Riders At Pickup Point When Instructions Known]
Def Riders shall eventually be at pickup point when instructions are known.
RefinedBy Achieve [Passenger At Pickup Point When Instructions Known], Achieve [Driver At Pickup Point When Instructions Known], Maintain [Rider Waiting For Other Rider]

Goal Achieve [Passenger In Driver's Car If Riders At Pickup Point]
Def For any ride of which passenger and driver are at pickup point, the passenger shall eventually take a seat in driver's car.
ObstructedBy Passenger Never In Driver's Car And Riders At Pickup Point
AssignedTo Passenger

Goal Achieve [Passenger At Pickup Point When Instructions Known]
Def Passenger shall eventually be at pickup point when instructions are known.
ObstructedBy Instructions Known And Passenger Not At Pickup Point
AssignedTo Passenger

Goal Achieve [Driver At Pickup Point When Instructions Known]
Def Driver shall eventually be at pickup point when instructions are known.
ObstructedBy Instructions Known And Driver Not At Pickup Point
AssignedTo Driver

Goal Maintain [Rider Waiting For Other Rider]
Def For any planned ride, the rider at pickup point shall wait for the other rider to arrive at specified pickup point.
RefinedBy Maintain [Passenger Waiting For Driver], Maintain [Driver Waiting For Passenger]

Goal Maintain [Passenger Waiting For Driver]
Def For any planned ride, the rider at pickup point shall wait for the other rider to arrive at specified pickup point.
AssignedTo Passenger

Goal Maintain [Driver Waiting For Passenger]
Def For any planned ride, the passenger at pickup point shall wait for the driver to arrive at specified pickup point.
AssignedTo Driver

Achieve [Matching Suggestion Selected]

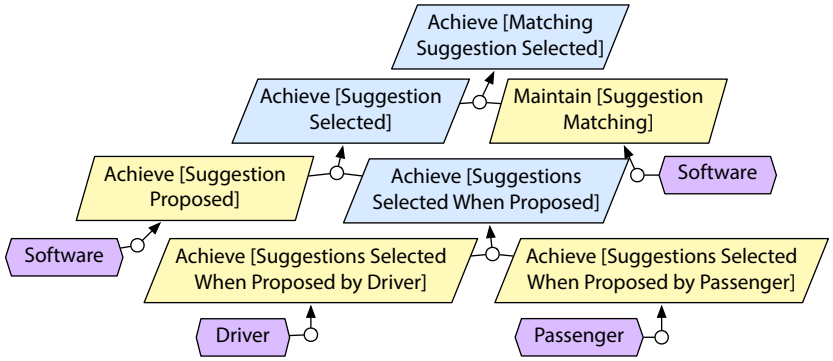


Figure A.46 Refinements for Achieve [Matching Suggestion Selected].

Goal Achieve [Matching Suggestion Selected]
Def The passenger and a driver shall eventually select at least one suggestion matching the request when this request has been encoded.
RefinedBy Achieve [Suggestion Selected], Maintain [Suggestion Matching]

Goal Achieve [Suggestion Selected]
Def For any encoded need, a proposition shall eventually be selected.
RefinedBy Achieve [Suggestion Proposed], Achieve [Suggestions Selected When Proposed]

Goal Maintain [Suggestion Matching]
Def For any selected suggestion, the request and offer from the suggestion matches.
ObstructedBy Proposition Not Matching When Selected
AssignedTo Software

Goal Achieve [Suggestion Proposed]
Def For any encoded request, a set of proposition shall eventually be suggested.
ObstructedBy No Pair Suggestion Proposed
AssignedTo Software

Goal Achieve [Suggestions Selected When Proposed]
Def For any encoded request, a suggested proposition shall eventually be selected.
RefinedBy Achieve [Suggestions Selected When Proposed by Driver], Achieve [Suggestions Selected When Proposed by Passenger]

Goal Achieve [Suggestions Selected When Proposed by Driver]
Def For any encoded request, a suggested proposition shall eventually be selected by the driver.
ObstructedBy No Passenger Selected
AssignedTo Driver

Goal Achieve [Suggestions Selected When Proposed by Passenger]
Def For any encoded request, a suggested proposition shall eventually be selected by the passenger.
ObstructedBy No Driver Selected
AssignedTo Passenger

A.2.2 Obstacle specifications

This section provides the complete specifications for the obstacles of the Car Pooling System, presented in Section 9.3.1.

Ride Offer Not Accurate

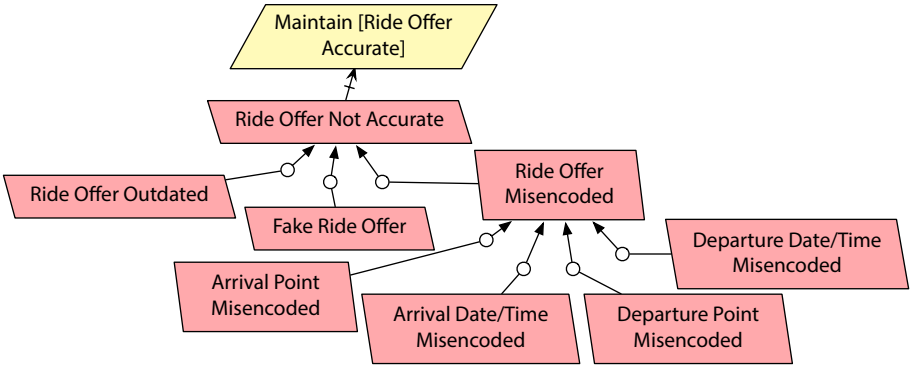


Figure A.47 Obstacle tree for Maintain [Ride Offer Accurate].

Obstacle Ride Offer Not Accurate
Def A ride offer encoded in the system does not accurately correspond to a ride proposal in the real world.
RefinedBy Fake Ride Offer
RefinedBy Ride Offer Outdated
RefinedBy Ride Offer Misencoded

Obstacle Fake Ride Offer
Def A ride offer encoded in the system does not correspond to any ride proposal in the real world.
ESR 0.10%

Obstacle Ride Offer Outdated
Def A ride offer is out-dated; the author of the offer did not update information yet.
ESR 1.00%

Obstacle Ride Offer Misencoded
Def Ride offer was Misencoded; some information may be missing or inaccurate.
RefinedBy Departure Date/Time Misencoded
RefinedBy Departure Point Misencoded
RefinedBy Arrival Date/Time Misencoded
RefinedBy Arrival Point Misencoded

Obstacle Departure Date/Time Misencoded
Def The departure date or time for a ride offer or request encoded in the system does not correspond to the expected date or time in the real world.
ESR 0.90%

Obstacle Departure Point Misencoded
Def The departure address for a ride offer or request encoded in the system does not correspond to the expect arrival point in the real world.
ESR 1.50%

Obstacle Arrival Date/Time Misencoded
Def The arrival date or time for a ride offer or request encoded in the system does not correspond to the expected date or time in the real world.
ESR 0.90%

Obstacle Arrival Point Misencoded
Def The arrival address for a ride offer or request encoded in the system does not correspond to the expect arrival point in the real world.
ESR 1.50%

Proposition Not Matching When Selected

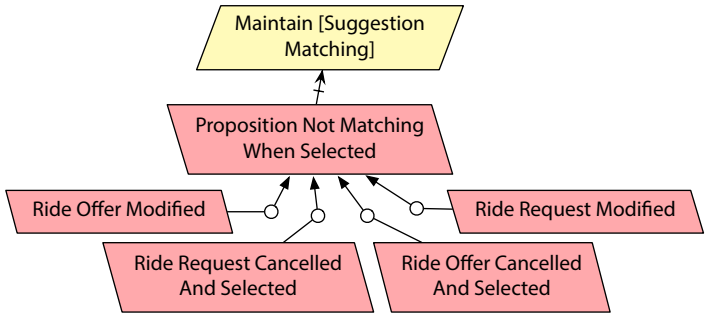


Figure A.48 Obstacle tree for Maintain [Suggestion Matching].

Obstacle Proposition Not Matching When Selected
Def When a proposal has been selected, the proposal does not remain matching.
RefinedBy Ride Offer Cancelled And Selected
RefinedBy Ride Request Cancelled And Selected
RefinedBy Ride Offer Modified
RefinedBy Ride Request Modified

Obstacle Ride Offer Modified
Def The ride offer has been selected and is modified such that the modification make the ride offer no longer matching the corresponding request.
ESR 1.00%

Obstacle Ride Request Modified
Def The ride request has been selected and is modified such that the modification make the ride request no longer matching the corresponding offer.
ESR 1.00%

Obstacle Ride Offer Cancelled And Selected
Def Ride offer is cancelled and selected.
ESR 1.00%

Obstacle Ride Request Cancelled And Selected
Def Ride request is cancelled and selected.
ESR 1.00%

No Pair Suggestion Proposed

Obstacle No Pair Suggestion Proposed
Def The software do not suggest compatible pairs.
RefinedBy No Request Matching Offer
RefinedBy No Offer Matching Request

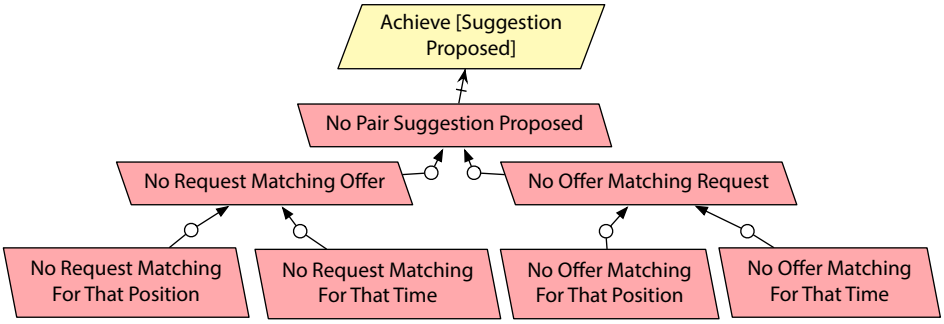


Figure A.49 Obstacle tree for Achieve [Suggestion Proposed].

Obstacle No Request Matching Offer
Def No request matches the encoded offer.
RefinedBy No Request Matching For That Time
RefinedBy No Request Matching For That Position

Obstacle No Offer Matching Request
Def No offer matches the encoded request.
RefinedBy No Offer Matching For That Time
RefinedBy No Offer Matching For That Position

Obstacle No Request Matching For That Time
Def No request satisfies the time constraints for the encoded offer.
ESR 2.00%

Obstacle No Request Matching For That Position
Def No request satisfies the geographic constraints for the encoded offer.
ESR 1.00%

Obstacle No Offer Matching For That Time
Def No offer satisfies the time constraints for the encoded request.
ESR 2.00%

Obstacle No Offer Matching For That Position
Def No offer satisfies the geographic constraints for the encoded request.
ESR 1.00%

No Passenger Selected and No Driver Selected

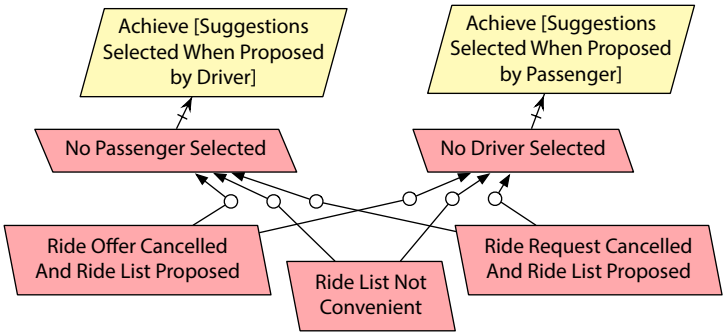


Figure A.50 Obstacle tree for Achieve [Suggestions Selected When Proposed by Driver] and Achieve [Suggestions Selected When Proposed by Passenger].

Obstacle No Driver Selected
Def No drivers are selected among the suggestions.
RefinedBy Ride List Not Convenient
RefinedBy Ride Offer Cancelled And Ride List Proposed
RefinedBy Ride Request Cancelled And Ride List Proposed

Obstacle No Passenger Selected
Def No passengers are selected among the suggestions.
RefinedBy Ride List Not Convenient
RefinedBy Ride Offer Cancelled And Ride List Proposed
RefinedBy Ride Request Cancelled And Ride List Proposed

Obstacle Ride Offer Cancelled And Ride List Proposed
Def Ride offer is cancelled and the ride list is proposed.
ESR 2.00%

Obstacle Ride Request Cancelled And Ride List Proposed
Def Ride request is cancelled and the ride list is proposed.
ESR 2.00%

Obstacle Ride List Not Convenient
Def The ride list is not convenient for the driver or the passenger.
ESR 1.00%

Ride Not Planned And Pair Elected

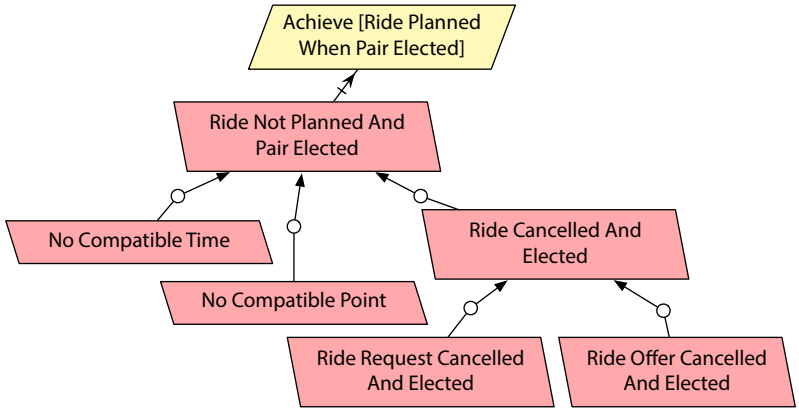


Figure A.51 Obstacle tree for Achieve [Ride Planned When Pair Elected].

Obstacle Ride Not Planned And Pair Elected
Def Ride is not planned but an adequate pair has been elected.
RefinedBy No Compatible Point
RefinedBy No Compatible Time
RefinedBy Ride Cancelled And Elected

Obstacle No Compatible Point
Def No pickup or drop point can be computed for the elected pair.
RefinedBy No Compatible Pickup Point Found
RefinedBy No Compatible Drop Point Found

Obstacle No Compatible Time
Def No pickup or drop time can be computed for the elected pair.
RefinedBy No Compatible Pickup Time Found
RefinedBy No Compatible Drop Time Found

Obstacle Ride Cancelled And Elected
Def Ride offer or request is cancelled.
RefinedBy Ride Offer Cancelled And Elected
RefinedBy Ride Request Cancelled And Elected

Obstacle Ride Offer Cancelled And Elected
Def Ride offer is cancelled and selected.
ESR 1.00%

Obstacle Ride Request Cancelled And Elected
Def Ride request is cancelled and selected.
ESR 1.00%

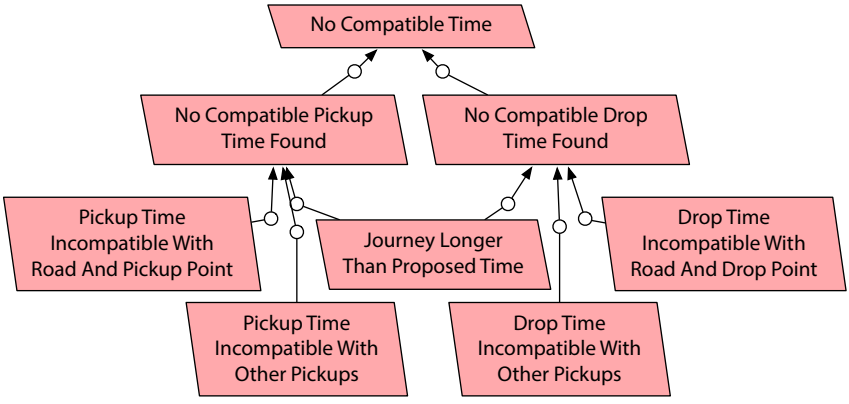


Figure A.52 Obstacle tree for No Compatible Time.

Obstacle No Compatible Pickup Time Found
Def No pickup time can be computed for the elected pair.
RefinedBy Pickup Time Incompatible With Road And Pickup Point
RefinedBy Pickup Time Incompatible With Other Pickups
RefinedBy Journey Longer Than Proposed Time

Obstacle No Compatible Drop Time Found
Def No drop time can be computed for the elected pair.
RefinedBy Drop Time Incompatible With Road And Drop Point
RefinedBy Drop Time Incompatible With Other Pickups
RefinedBy Journey Longer Than Proposed Time

Obstacle Pickup Time Incompatible With Road And Pickup Point
Def No compatible pickup time can be computed to reach the computed pickup point at that time.
ESR 0.50%

Obstacle Pickup Time Incompatible With Other Pickups
Def No compatible pickup time can be computed to reach the computed pickup point at that time due to other pickups.
ESR 0.50%

Obstacle Journey Longer Than Proposed Time
Def Journey is too long to pickup and drops the passenger within time constraints.
ESR 0.50%

Obstacle Drop Time Incompatible With Road And Drop Point
Def No compatible drop time can be computed to reach the computed drop point at that time.
ESR 0.50%

Obstacle Drop Time Incompatible With Other Pickups
Def No compatible drop time can be computed to reach the computed drop point at that time due to other pickups.
ESR 0.50%

Obstacle No Compatible Pickup Point Found
Def No pickup point can be computed for the elected pair.
RefinedBy No Pickup Point Accessible To Driver And Passenger
RefinedBy No Pickup Point Near Departure Point
RefinedBy Detour Too Important For Reaching Pickup Point

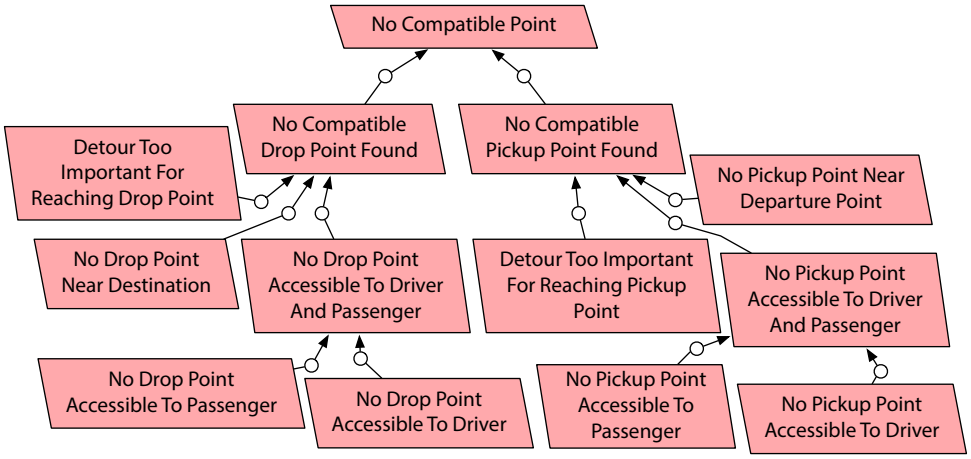


Figure A.53 Obstacle tree for No Compatible Point.

Obstacle No Compatible Drop Point Found
Def No drop point can be computed for the elected pair.
RefinedBy No Drop Point Accessible To Driver And Passenger
RefinedBy No Drop Point Near Destination
RefinedBy Detour Too Important For Reaching Drop Point

Obstacle No Pickup Point Near Departure Point
Def Given a compatible pair of request / offer, no pickup point is located within 1km from the departure point of the passenger.
ESR 0.50%

Obstacle Detour Too Important For Reaching Pickup Point
Def Detour made by driver to reach the pickup point from her route exceeds 3 km.
ESR 1.00%

Obstacle No Pickup Point Accessible To Driver And Passenger
Def No pickup point is accessible to both driver and passenger.
RefinedBy No Pickup Point Accessible To Driver
RefinedBy No Pickup Point Accessible To Passenger

Obstacle No Pickup Point Accessible To Driver
Def No pickup point is accessible to driver.
ESR 0.50%

Obstacle No Pickup Point Accessible To Passenger
Def No pickup point is accessible to passenger.
ESR 0.50%

Obstacle No Drop Point Near Destination
Def Given a compatible pair of request / offer, no drop point is located within 1km from the destination of the passenger.
ESR 1.00%

Obstacle Detour Too Important For Reaching Drop Point
Def Detour made by driver to reach the drop point from her route exceeds 3 km.
ESR 0.50%

Obstacle No Drop Point Accessible To Driver And Passenger
Def No drop point is accessible to both driver and passenger.
RefinedBy No Drop Point Accessible To Driver
RefinedBy No Drop Point Accessible To Passenger

Obstacle No Drop Point Accessible To Driver
Def No drop point is accessible to driver.
ESR 0.50%

Obstacle No Drop Point Accessible To Passenger
Def No drop point is accessible to passenger.
ESR 0.50%

Ride Instructions Not Sent When Ride Planned

Obstacle Ride Instructions Not Sent When Ride Planned
Def Ride instructions are not sent but the ride is planned
RefinedBy Wrong Contact Information
RefinedBy Failed Communication

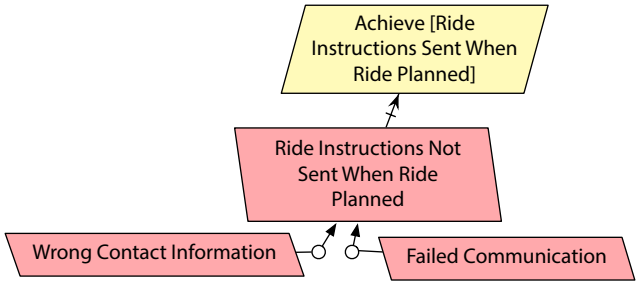


Figure A.54 Obstacle tree for Achieve [Ride Instructions Sent When Ride Planned].

Obstacle Wrong Contact Information
Def The contact information about the ride are not accurate.
ESR 7.00%

Obstacle Failed Communication
Def The communication with the ride fail.
ESR 3.00%

Ride Instructions Not Read By Passenger And Sent

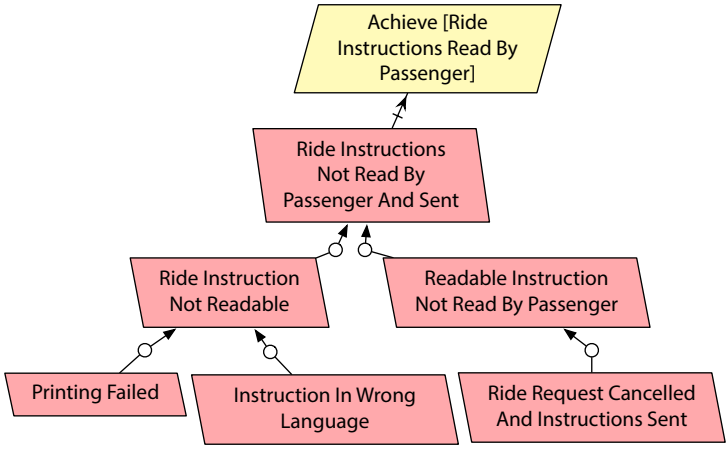


Figure A.55 Obstacle tree for Achieve [Ride Instructions Read By Passenger].

Obstacle Ride Instructions Not Read By Passenger And Sent
Def The sent instructions are not read by the passenger.
RefinedBy Ride Instruction Not Readable
RefinedBy Readable Instruction Not Read By Passenger

Obstacle Ride Instruction Not Readable
Def The ride instructions are not readable by the rider.
RefinedBy Instruction In Wrong Language
RefinedBy Printing Failed

Obstacle Readable Instruction Not Read By Passenger
Def The instructions are readable but not read by passenger.
RefinedBy Ride Request Cancelled And Instructions Sent

Obstacle Ride Request Cancelled And Instructions Sent
Def Ride request is cancelled and instructions are sent.
ESR 1.50%

Obstacle Instruction In Wrong Language
Def Language of the ride instructions is not understood by the rider.
ESR 0.50%

Obstacle Printing Failed
Def Printing the ride instructions failed.
ESR 1.00%

Ride Instructions Not Read By Driver And Sent

Obstacle Readable Instruction Not Read By Driver
Def The instructions are readable but not read by driver.
RefinedBy Ride Offer Cancelled And Instructions Sent

Obstacle Ride Offer Cancelled And Instructions Sent
Def Ride offer is cancelled and instructions are sent.
ESR 1.50%

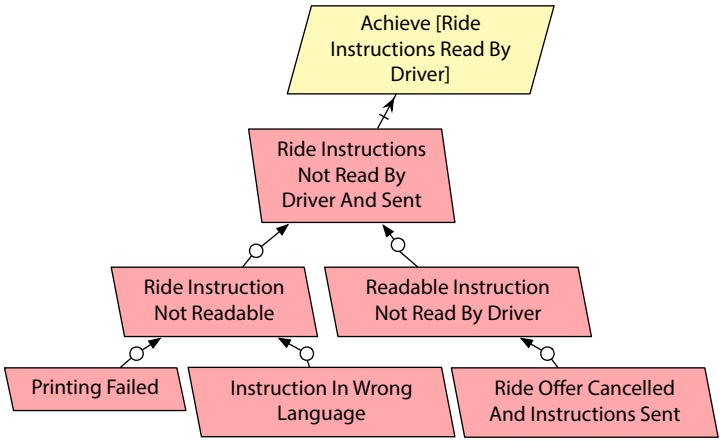


Figure A.56 Obstacle tree for Achieve [Ride Instructions Read By Driver].

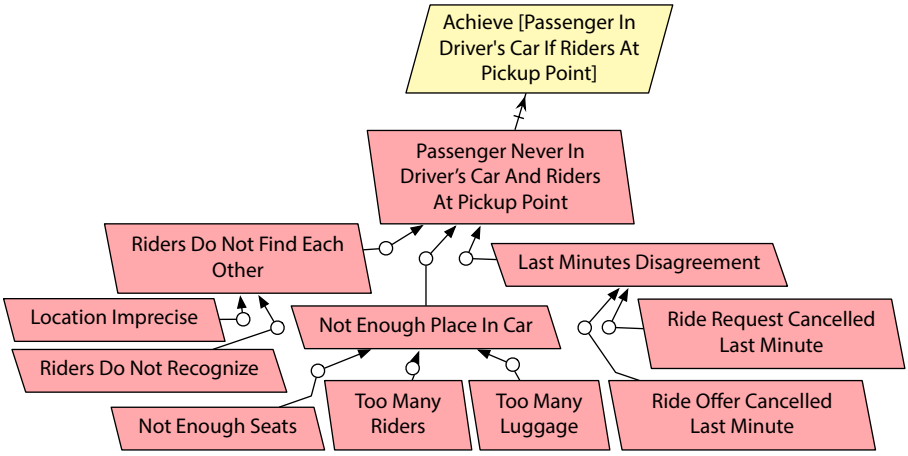


Figure A.57 Obstacle tree for Achieve [Passenger In Driver's Car If Riders At Pickup Point].

Passenger Never In Driver's Car And Riders At Pickup Point

Obstacle Passenger Never In Driver's Car And Riders At Pickup Point
Def The passenger is never in the driver's car but both riders are at the specified pickup point.
RefinedBy Not Enough Place In Car
RefinedBy Riders Do Not Find Each Other
RefinedBy Last Minutes Disagreement

Obstacle Not Enough Place In Car
Def There is not enough places in the driver's car.
RefinedBy Too Many Luggage
RefinedBy Too Many Riders
RefinedBy Not Enough Seats

Obstacle Riders Do Not Find Each Other
Def The riders do not find each other.
RefinedBy Riders Do Not Recognize
RefinedBy Location Imprecise

Obstacle Last Minutes Disagreement
Def Passenger and driver disagree at the last minutes.
RefinedBy Ride Request Cancelled Last Minute
RefinedBy Ride Offer Cancelled Last Minute

Obstacle Ride Offer Cancelled Last Minute
Def Ride offer is cancelled within the last minutes.
ESR 3.00%

Obstacle Ride Request Cancelled Last Minute
Def Ride request is cancelled within the last minutes.
ESR 2.00%

Obstacle Too Many Luggage
Def There is too many luggages in the driver's car, precluding the passenger to take a seat.
ESR 0.50%

Obstacle Too Many Riders
Def There is too many riders in the driver's car, precluding the passenger to take a seat.
ESR 0.50%

Obstacle Not Enough Seats
Def There is not enough available seats in the driver's car, precluding the passenger to take a seat.
ESR 0.50%

Obstacle Riders Do Not Recognize
Def Riders do not recognize each other.
ESR 1.00%

Obstacle Location Imprecise
Def Location is sufficiently imprecise so that the passenger and driver do not find each other.
ESR 2.00%

Instructions Known And Passenger Not At Pickup Point

Obstacle Instructions Known And Passenger Not At Pickup Point
Def Instructions are known by the passenger but the passenger is not at pickup point within time constraints.
RefinedBy Passenger Late At Pickup Point
RefinedBy Passenger Never At Pickup Point

Obstacle Passenger Late At Pickup Point
Def Passenger eventually reach the pickup point but not within time constraints.
ESR 6.00%

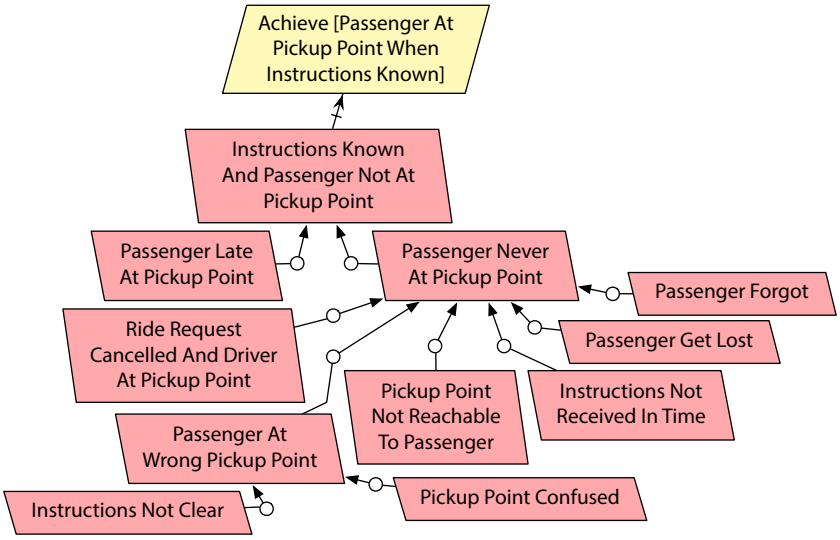


Figure A.58 Obstacle tree for Achieve [Matching Suggestion Selected].

Obstacle Passenger Never At Pickup Point
Def Passenger never reaches the pickup point.
RefinedBy Passenger Forgot
RefinedBy Passenger Get Lost
RefinedBy Instructions Not Received In Time
RefinedBy Pickup Point Not Reachable To Passenger
RefinedBy Ride Request Cancelled And Driver At Pickup Point
RefinedBy Passenger At Wrong Pickup Point

Obstacle Ride Request Cancelled And Driver At Pickup Point
Def The ride request is cancelled and the driver is at pickup point.
ESR 1.00%

Obstacle Passenger Forgot
Def Passenger forgot to go the specified pickup point
ESR 1.00%

Obstacle Passenger Get Lost
Def Passenger gets lost.
ESR 2.00%

Obstacle Instructions Not Received In Time
Def Instructions for reaching the pickup point arrives too late.
ESR 2.00%

Obstacle Pickup Point Not Reachable To Passenger
Def Pickup point is not reachable to passenger.
ESR 0.50%

Obstacle Passenger At Wrong Pickup Point
Def Passenger reaches an other, wrong, pickup point.
RefinedBy Pickup Point Confused
RefinedBy Instructions Not Clear

Obstacle Pickup Point Confused
Def Tow pickup points are confused.
ESR 1.00%

Obstacle Instructions Not Clear
Def Instructions are not clear.
ESR 0.50%

Instructions Known And Driver Not At Pickup Point

Obstacle Instructions Known And Driver Not At Pickup Point
Def Instructions are known by the driver but the driver is not at pickup point.
RefinedBy Driver Late At Pickup Point
RefinedBy Driver Never At Pickup Point

Obstacle Driver Late At Pickup Point
Def Driver eventually reach the pickup point but not within time constraints.
ESR 2.00%

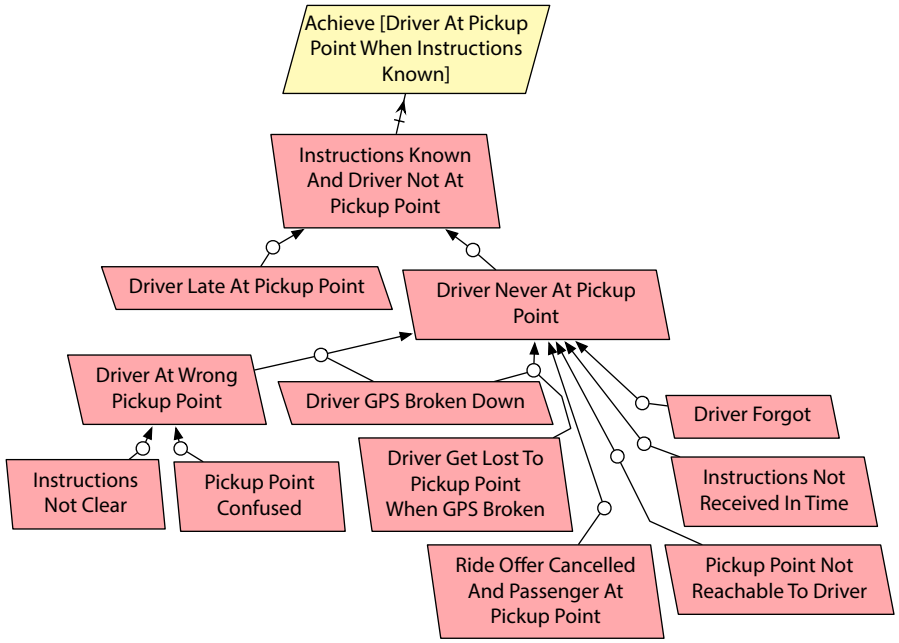


Figure A.59 Obstacle tree for Achieve [Passenger At Pickup Point When Instructions Known].

Obstacle Driver Never At Pickup Point
Def Driver never reaches the pickup point.
RefinedBy Driver Forgot
RefinedBy Driver Get Lost To Pickup Point When GPS Broken, Driver GPS Broken Down
RefinedBy Instructions Not Received In Time
RefinedBy Pickup Point Not Reachable To Driver
RefinedBy Ride Offer Cancelled And Passenger At Pickup Point
RefinedBy Driver At Wrong Pickup Point, Driver GPS Broken Down

Obstacle Ride Offer Cancelled And Passenger At Pickup Point
Def The ride offer is cancelled and the passenger is at pickup point.
ESR 1.00%

Obstacle Driver Forgot
Def Driver forgot to go the specified pickup point
ESR 2.00%

Obstacle Driver GPS Broken Down
Def The driver GPS is not working.
ESR 20.00%

Obstacle Driver Get Lost To Pickup Point When GPS Broken
Def Driver gets lost on its way to the pickup point when its GPS is broken.
ESR 30.00%

Obstacle Pickup Point Not Reachable To Driver
Def Pickup point is not reachable to driver.
ESR 0.50%

Obstacle Driver At Wrong Pickup Point
Def Driver reaches an other, wrong, pickup point.
RefinedBy Pickup Point Confused
RefinedBy Instructions Not Clear

Drop Point Not Reached When Passenger In Car

Obstacle Drop Point Not Reached When Passenger In Car
Def Drop point is not reached within time constraints when the passenger is in car.
RefinedBy Drop Point Not Reached In Time
RefinedBy Drop Point Never Reached

Obstacle Drop Point Not Reached In Time
Def Drop point is reached late when the passenger is in car.
RefinedBy Other Passenger Late
RefinedBy Detour On Planned Road
RefinedBy Stuck In Traffic Jams
RefinedBy Wrong instructions Sent

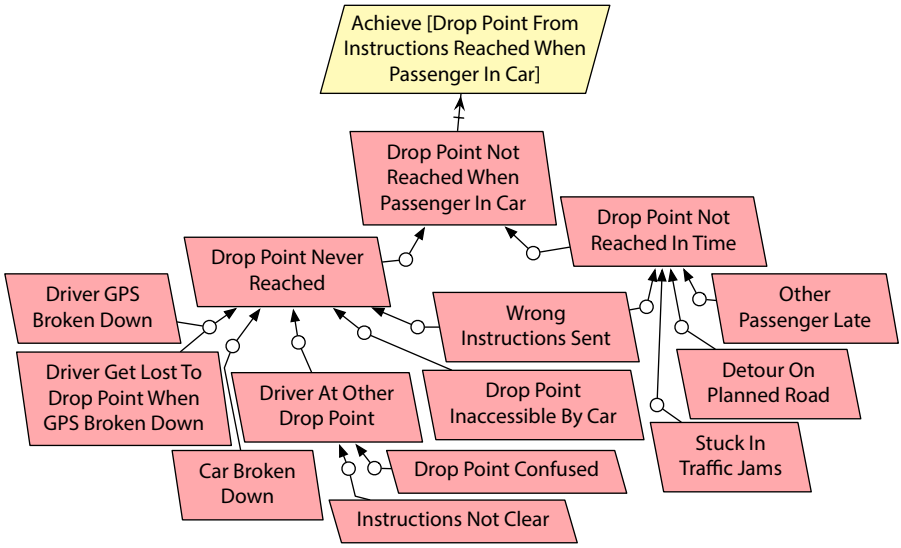


Figure A.60 Obstacle tree for Achieve [Drop Point From Instructions Reached When Passenger In Car].

Obstacle Drop Point Never Reached
Def Drop point is never reached when the passenger is in car.
RefinedBy Wrong instructions Sent
RefinedBy Car Broken Down
RefinedBy Driver Get Lost To Drop Point When GPS Broken Down, Driver GPS Broken Down
RefinedBy Drop Point Inaccessible By Car
RefinedBy Driver At Other Drop Point

Obstacle Other Passenger Late
Def Drop point for a given passenger is reached late due to other passengers pickuped late.
ESR 0.50%

Obstacle Detour On Planned Road
Def Unexpected and unplanned detour is required to reach the drop point.
ESR 1.50%

Obstacle Stuck In Traffic Jams
Def Driver's car is stuck, with passenger, in traffic jams.
ESR 10.00%

Obstacle Wrong instructions Sent
Def Sent instructions are inaccurate.
ESR 0.10%

Obstacle Car Broken Down
Def Driver's car broke down, when passenger is in car.
ESR 0.50%

Obstacle Driver Get Lost To Drop Point When GPS Broken Down
Def Driver gets lost on its way to the drop point.
ESR 20.00%

Obstacle Drop Point Inaccessible By Car
Def The planned drop point is inaccessible by car.
ESR 0.50%

Obstacle Driver At Other Drop Point
Def Driver reach another drop point than the specified one.
RefinedBy Drop Point Confused
RefinedBy Instructions Not Clear

Obstacle Drop Point Confused
Def Two plausible drop points are confused by both the driver and the passenger.
ESR 1.00%

Ride Request Not Accurate

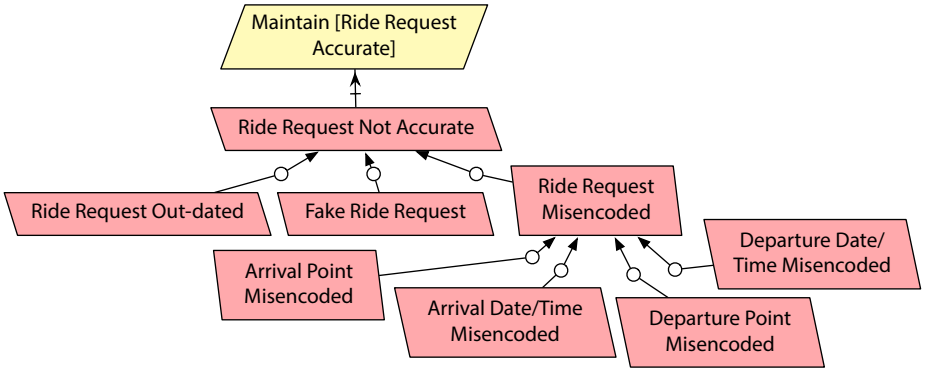


Figure A.61 Obstacle tree for Maintain [Ride Request Accurate].

Obstacle Ride Request Not Accurate
Def Ride request encoded in the system does not correspond accurately to a ride need in the real world.
RefinedBy Fake Ride Request
RefinedBy Ride Request Out-dated
RefinedBy Ride Request Misencoded

Obstacle Fake Ride Request
Def Ride request encoded in the system does not correspond to any ride need in the real world.
ESR 0.10%

Obstacle Ride Request Out-dated
Def Ride request is out-dated; the author of the request did not update information yet.
ESR 1.00%

Obstacle Ride Request Misencoded
Def Ride request origin or destination information is Misencoded.
RefinedBy Departure Date/Time Misencoded
RefinedBy Departure Point Misencoded
RefinedBy Arrival Date/Time Misencoded
RefinedBy Arrival Point Misencoded

Obstacle Ride Instructions Not Read By Driver And Sent
Def The sent instructions are not read by the driver.
RefinedBy Ride Instruction Not Readable
RefinedBy Readable Instruction Not Read By Driver

Appendix B

KAOSTools Formal Specification Grammar

```
<Formula> :=
    "forall" <Variable> ":" <Identifier>
             ("," <Variable> ":" <Identifier>)* "." <Formula>
    | "exists" <Variable> ":" <Identifier>
               ("," <Variable> ":" <Identifier>)* "." <Formula>
    | <StrongBinary>

# 'when A then B' is equivalent to 'always (if A then B)'
<StrongBinary> :=
    "when" <Binary> "then" <Formula>
    | <Binary>

# 'if A then B' is the implication
# 'A iff B' is the equivalence
<Binary> :=
    "if" <TemporalBinary> "then" <Formula>
    | <TemporalBinary> "iff" <Formula>
    | <TemporalBinary>

<TemporalBinary> :=
    <And> "until" <EventuallyTimeBoundEmphasis>? <Formula>
    | <And> "release" <Formula>
    | <And> <GloballyTimeBoundEmphasis>? "unless" <Formula>
    | <And>

<And> :=
    <Or> "and" <Formula>
    | <Or>

<Or> :=
    <Unary> "or" <Formula>
    | <Unary>
```

```
<Unary> :=
    "not" <Unary>
    | "next" <Unary>
    | ("sooner-or-later" | "eventually") <EventuallyTimeBoundEmphasis>?
<Unary>
    | ("always" | "globally") <GloballyTimeBoundEmphasis>? <Unary>
    | <Atom>

<Atom> :=
    <RelationReference>
    | <Comparison>
    | <AttributeReference>
    | <PredicateReference>
    | "(" <Formula> ")"

<Comparison> :=
    <ComparisonMember> <Comparator> <ComparisonMember>

<Comparator> :=
    ( "==" | "!=" | ">=" | "<=" | ">" | "<" )

<ComparisonMember> :=
   <AttributeReference> | <PredicateReference> | <VariableReference>
   | """ <String>? """ | <Float> | <Integer> | <Bool>

<AttributeReference> :=
   <Variable> "." <Identifier>

<RelationReference> :=
   "(" <Variable> ("," <Variable>)* ")" "in" <Identifier>

<PredicateReference> :=
   <Identifier> ( "(" (<Variable> ( "," <Variable> )*)? ")" )

<VariableReference> := <Variable>

<EventuallyTimeBoundEmphasis> :=
   "," <EventuallyTimeBound> ","

<GloballyTimeBoundEmphasis> :=
   "," <GloballyTimeBound> ","
```

```
<EventuallyTimeBound> :=
   ("strictly" )? "before" <TimeConstraint>
   | ("strictly" )? "after" <TimeConstraint>
   | "in" <TimeConstraint>

<GloballyTimeBound> :=
   "for" ("strictly" )? "more" "than" <TimeConstraint>
   | "for" ("strictly" )? "less" "than" <TimeConstraint>
   | "for" <TimeConstraint>

<TimeConstraint> :=
   (<Integer> <TimeUnit>) ( <Integer> <TimeUnit>)*

<TimeUnit> :=
   ( "day" | "days" | "d" )
   | ( "hour" | "hours" | "h" )
   | ( "minute" | "minutes" | "min" )
   | ( "second" | "seconds" | "s" )
   | ( "millisecond" | "milliseconds" | "ms" )

<LogicFormula> :=
   <LogicFormulaBinary>

<LogicFormulaBinary> :=
    "if" <LogicFormulaAnd> "then" <LogicFormula>
    | <LogicFormulaAnd> "iff" <LogicFormula>
    | <LogicFormulaAnd>

<LogicFormulaAnd> :=
    <LogicFormulaOr> "and" <LogicFormula>
    | <LogicFormulaOr>

<LogicFormulaOr> :=
    <LogicFormulaUnary> "or" <LogicFormula>
    | <LogicFormulaUnary>

<LogicFormulaUnary> :=
    "not" <LogicFormulaUnary>
    | <LogicFormulaAtom>

<LogicFormulaAtom> :=
   "true" | "false"
    | <RelationReference>
```

```
| <Comparison>
| <AttributeReference>
| <PredicateReference>
| "(" <LogicFormula> ")"
```