LOUVAIN
School of Management
RESEARCH INSTITUTE

WORKING PAPER

2016/23

User-Story Driven Development of Multi-Agent Systems

Soreangsey Kiv,
Louvain School of Management

Samedi Heng,
Louvain School of Management

Yves Wautelet,
KU Leuven, Belgium

Manuel Kolp,
Louvain School of Management

Louvain School of Management
Working Paper Series
Editor : Prof. Frank Janssen
(president-ilsm@uclouvain.be)

User-Story Driven Development of Multi-Agent Systems

Soreangsey Kiv, Louvain School of Management
Samedi Heng, Louvain School of Management
Yves Wautelet, KU Leuven, Belgium
Manuel Kolp, Louvain School of Management

Summary

Agile software development methodologies are mostly constituted of as a set of managerial guidelines and concepts not bounded to software development paradigms. Some of these like eXtreme Programming (XP) and the SCRUM are driven by operational requirements representation models/artifacts called User Stories (US). These US can be used as an anchoring point to agile methods; this means that we could use a US set to drive a software transformation approach to a particular development paradigm like agent-orientation. This paper presents a process plug-in for Multi-Agent System (MAS) development in agile methods based on US sets. The process plug-in takes advantage of an initial set of US to build a reasoning diagram (called the Rationale Diagram typically several are built for a single project) that document decompositions and means-end alternatives in scenarios for requirements realization. A Rationale Diagram can then be aligned with a MAS architectural design and implemented in an agent-oriented development language. The process plug-in is made out of three stages: Requirements
Analysis, Architectural Design and Implementation. Transformation from one stage to the other is overseen in this paper and illustrated on a case study.

Keywords: Agent Architecture, Agile Development, User Story, Agile Architecture, Multi-Agent System.

User-Story Driven Development of Multi-Agent Systems: A Process Fragment for Agile Methods

Abstract

Agile software development methods are mostly built as a set of managerial guidelines and development concepts on how to handle a software development but are not bounded to software development paradigms like object or agent orientation. Some methods, like eXtreme Programming and SCRUM are driven by operational requirements representation models called User Stories. These User Stories can be used as an anchoring point to agile methods; this means that we could take a User Stories set to drive a software transformation approach embedded in a particular development paradigm. This paper presents a process fragment for Multi-Agent Systems development with agile methods based on User Stories sets. The process fragment indeed takes advantage of an initial set of User Stories to build a reasoning model (called the Rationale Tree; typically several of these are built for a single project) that documents decompositions and means-end alternatives in scenarios for requirements realization. A Rationale Tree can then be aligned with a Multi-Agent design and implemented in an agent-oriented development language. In this paper the transformation is targeted to the JAVA Agent DEvelopment (JADE) framework. The process fragment (at least partially) covers the *Requirements Analysis*, *Multi-Agent System Design* and *Multi-Agent System Implementation* phases. Transformation from one phase to the other is overseen and illustrated on an example.

Keywords: Agent Software Engineering, Agile Development, User Story, Multi-Agent System, Process Fragment, Rationale Tree, JAVA Agent DEvelopment Framework, JADE, i*-based software process modeling

1. Introduction

1.1. Research Focus and Scope

Agile software development methods are nowadays well known and widely used in the software industry [16]. They are based on a set of principles and assumptions (described in [45, 44]) that work particularly well for the development of software applications for which innovation is a key aspect [27]. Such methods do not imply heavy requirements analysis and rigid software architecture; they rather encourage building flexible software that is continuously modified and refined during the project due to the strong involvement of end users and other stakeholders. They thus require the fast development of software prototypes or units to be tested in early phases of the project life cycle. Flexibility and change management of the software under development are key aspects to allow implementing modifications and increments rapidly and easily.

Agile methods like eXtreme Programming (XP) [3] and SCRUM [49] mostly use sets of *User Stories* (*US*) as main requirement artifacts. *User stories are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system* [13]. They dynamically describe actions performed by roles involved in the proper execution of the produced software; they are thus driven by run-time system behavior. US are, in this perspective, particularly well suited to describe software systems in which human and organizational aspects play a key role like for example nowadays' mobile apps.

We suggest to extend US-based agile software development methods with the adoption of *Multi-Agent Systems* (*MAS*) technology at the design and implementation stages. We will show that a precise study of US elements and interdependencies enriched with further domain knowledge allows to build a reasoning model that can ultimately be mapped to run-time MAS behavior. The transformation process of a consistent US set into a MAS design can then lead to the fast development of flexible prototypes; the so-developed software thus integrates the overall benefits of MAS technology. Therefore, this paper introduces a (software development) process fragment for agent-based development in agile methods. As an initial US set is used to build a graphical model that is then transformed into a MAS design and implementation, it can be characterized as *model-driven* (see [14]).

In the context of this paper, we adopt the definition of Seidita et al. [40] for a ***Process Fragment***, i.e. *a portion of design process adequately created*

and structured for being reused during the composition and enactment of new design processes both in the field of agent oriented software engineering and in other ones. Although we do not fully instantiate each concept defined in Seidita et al. [40], we rely on their terminology when defining our process fragment (see Section 4). When, in this paper, the concepts from [40] are firstly used and instanciated to our process fragment, we have put the concept in italic and bold (this way concepts from their process engineering framework are recognizable at first sight). We systematically invite the reader to refer to that source for further explanation of each concept; each instantiated concept definition is nevertheless transcripted in Section 4.

1.2. Contributions and Discussion

The paper constitutes a first attempt to integrate agent-technology with agile software development. The contribution of the present paper is the process fragment itself. Table 1 of Section 4 summarizes the meta-model process fragment elements of [40] that are instantiated to our particular case. All of these instances are necessary for implementing the process fragment and constitute a whole for its proper application.

A subset of these elements are depicted in the form of an i* Strategic Rationale diagram [57, 59, 58]. This allows to show the dependencies and place where each of these elements need to be used. The i* diagram also constitutes a reference for practitioners or researchers willing to apply our contribution.

More importantly, the application of the process fragment induces the transformation from requirements models to MAS design and implementation. To this end, rules (called ***Composition Guidelines***) to transform elements from the US unified model of [51] (i.e. the requirements meta-model) to the *JAVA Agent DEvelopment* (*JADE*) framework meta-model of [4] are defined in the paper. These two models are ***System Meta-Model Constructs*** (***SMMC***) of the process fragment and the ***Composition Guidelines*** constitute another of our main contributions.

For a proper integration of the process fragment within an agile method we only point to one prerequisite: *a US set expressed through a WHO, WHAT and WHY dimension describing the main aspects of the system to-be*. One might argue that we could have given up US in agile development and point to building other kinds of requirements artifacts specifically targeted to the development of MAS. Nevertheless, US writing remains a fundamental practice in agile methods [33] so we believe that it constitutes the adequate anchoring

point for further agent-based development. The prerequisite is met by most of agile developments based on XP and SCRUM.

Finally, a specific Computer-Aided Software Engineering (CASE) tool has also been developed to support the use of our process fragment.

1.3. Paper Structure

The paper is structured as follows. Section 2 depicts the research context. Section 3 the applied research method. Section 4 depicts the agent development process fragment for agile methods that constitutes the core contribution of the paper. Section 5 overviews the mapping rules – referred to as composition guidelines for the process fragment – of the Rationale Tree elements with the ones of the JADE framework as well as the specifically developed CASE-Tool. Section 6 illustrates the proposal on the development of a carpooling example. Section 7 describes the related work. Section 8 discusses the validity, the threats to the validity, as well as the scalability of the approach and future work. Finally, Section 9 concludes the paper.

2. Research Context

Currently, there is no unification in the use of US templates [51]. Indeed, the commonly used pattern (which is the one tackled in this paper with no further extensions) relates a WHO, a WHAT and possibly a WHY dimension[1] (Appendix A documents how we decompose US), but several concepts can be found in literature or blogs to represent these dimensions (e.g., Mike Cohn's *As a <type of user>, I want <some goal> so that <some reason>* [13]). Moreover, no definitions (i.e. semantics) are ever associated to these concepts (see Wautelet et al. [51]). Consequently, in 2013, Wautelet et al. [51] collected an exhaustive set of US templates used by practitioners classified them and related definitions to each concept. These definitions were found in various sources and frameworks, i.e. [57, 47, 36, 24]; some are based on Goal-Oriented Requirements Engineering (GORE, see [46]). After removing redundant concepts, a unified model for US templates was built (see Appendix A.2). Most of the definitions and concepts of this unified model come from the i* framework. The reader interested in the full research process to build the unified US model is invited to refer to Wautelet et al. [51].

[1] Examples are provided in Table 2.

A US tagged with the unified model evoked in previous section furnishes information on *the nature and granularity of the US element* in the WHAT and/or WHY dimensions. If the tagging is done by respecting the definitions associated to the concepts as well as the internal consistency of the US set, we can structure the set of US and use it to build a graphical model. Granularity identification of US has been identified in literature as an issue to be solved for an adequate structuring of requirements [31]. In an agile project, we can better support requirements engineering by enhancing the structure of the to-be software system when the exact granularity of a US element is determined. Through domain analysis, US elements can also be linked towards means-end and other decomposition links to depict scenarios of behaviors solving parts of the software problem. This was shown in [52] through the building of a graphical model of which the instance is called a Rationale Tree (see Appendix A.3 for more information). The graphical analysis of the US set using a Rationale Tree not only allows us to structure requirements but also allows to evaluate the consistency of the US set.

A consistent Rationale Tree built out of the requirements analysis describes (part of) a software solution behavior that can be aligned with the design of a MAS. Such a MAS is thus intended to map human behavior with system run-time behavior. The process fragment developed in this paper aims to first build the Rationale Tree for a specific project and map it to a MAS design compliant to the JADE framework.

3. Research Method

The present research is built as *design science* meaning that we do not try to understand reality as in social sciences but rather to build a framework that can serve human purposes [42]. We have thus proceeded to a problem identification namely *the integration of agent software development in agile and US-based development*; this constitutes the process fragment ***Goal***. The process fragment itself is a solution for it.

To such an extend, we have firstly defined an abstract view of the process fragment using an i* strategic rationale diagram [57]. This view has been built by identifying the principal stakeholders involved in a traditional agile development and enhancing it with the practices and models required for agent-oriented development. The process elements have been kept minimal and serve as a guidance for development. A specific software engineering meta-model like the *Software & Systems Process Engineering Metamodel*

(*SPEM*) [2] could have been used to make such a description (see for example [18, 28] for an application in the field of agent software development) but we wanted to keep the *Description* light and limited to essentials. Also, we wanted to explicitly show the dependencies of ***Work Products*** among the ***Roles*** which is easier with i* (by nature driven by elements dependencies and decompositions) then with meta-models like for example the SPEM.

Then, we have defined transformation ***Composition Guidelines*** from the Rationale Tree developed in previous works to a MAS architecture in JADE. These ***Composition Guidelines*** have been built by making a Cartesian product between the elements of the Rationale Tree as presented in Wautelet et al. [52] and the elements of the JADE framework as presented in Bellifemine et al. [4]. This allowed to determine the best possible mapping to implement a Rationale Tree and its reasoning abilities as a MAS in JADE.

As evoked in Section 2, the research to build a process for integrating MAS in agile development is based on previously validated works. The unified model for US templates as well as the related Rationale Tree are two previous contributions.

4. Agent Development Process Fragment: a Description

Table 1 documents all of the process fragment elements defined in Seidita et al. [40] that are instantiated onto our contribution.

Table 1: Instantiation of our Process Fragment.

Element	Definition (from [40])	Instantiation to our process fragment	i* Representation
Design Process	*Design process from which the fragment has been extracted.*	*User Story based Agile Methods like XP or SCRUM.*	N/A
Phase	*A specification of the fragment position in the design workflow. Usually referring to a taxonomy.*	*Requirements Analysis*, *Multi-Agent System Design*, *Multi-Agent System Implementation.*	Goal

Goal	*The process-oriented objective of the fragment.*	*The integration of agent software development in agile and US-based development.*	N/A
Activity	*A portion of work assignable to to a performer (role).*	*Build Rationale Tree*, *Tag User Story elements*, *Link User Story Elements*, *Remove redundant requirements*, *Identify missing requirements*, *Align Multi-Agent System Design with Rationale Tree*, *Define Multi-Agent System structure*, *Specify temporal exchange of messages*, *Implement software in an agent-oriented programming language.*	Task
Work Product	*The resulting product of the work done in the fragment; it can be realized in different ways also depending on the specific adopted notation.*	*Structured User Story*, *Initial User Story Set*, *Refined User Story Set*, *Consistent Rationale Tree*, *Multi-Agent System Design*, *Software Unit*, *Refined User Story Subset.*	Resource
Role	*The stakeholder performing the work in the process and responsible of producing a work product (or part of it).*	*Agile Product Owner*, *Software Analyst*, *Software Designer*, *Software Developer*, *System Tester*, *User.*	Actor
System Meta-Model Construct	*The concept of the fragment deals with, for instance a fragment aimaing at defining the system requirements has to defined and to identify the concept or requirements.*	*The Unified User Story Model* (from [51]), *The JADE Meta-Model* (from [4]).	N/A

Description	*It is the textual and pictural description of the fragment; it provides a bird-eye on the whole process the fragments come from and the fragment overview in terms of tasks to be performed, roles and work product kind to be delivered.*	The i* Process Model of Figure 1.	A Strategic Rationale Diagram
Composition Guideline	*A set of guidelines for assembling/composing the fragments with others.*	The transformation rules expressed in Section 5.	N/A

As already evoked, to show the ***Roles*** involved in the process fragment, their ***Work Product*** dependencies and their , we use a pictorial ***Description*** using the i* framework in Figure 1. For the sake of clarity, we insist that each of the ***Roles*** can, of course, be played by several individuals in a same project; similarly a same individual can play different ***Roles*** in a project.

We distinguish several types of stakeholders, all having various objectives and expectations, i.e.:

- The *Agile Product Owner* (*APO*) ***Role*** is a senior manager that is mainly in charge of developing a vision of the software system that needs to be built and propagate that vision over the development team; it is a key stakeholder of the project. His/her ***Activities*** are essentially outside the scope of our process fragment and are guided by the adopted agile method; we nevertheless identify and represent the *APO* ***Role*** here because the output of some of his/her activities are required by our process fragment. The *APO* ***Role*** indeed uses the product backlog (see [37, 8]) to store the *Initial User Story Set* ***Work Product***. The *Software Modeler* ***Role*** thus depends on the *APO* ***Role*** for obtaining the *Initial User Story Set* ***Work Product***; that is why it is

represented as a Resource dependency in the i* diagram of Figure 1. This *Initial User Story Set* is required for composition with our process fragment; it is typically collected over several future system (end) *Users* ***Role*** in the form of a *Structured User Story* ***Work Product***. The *APO* ***Role*** thus depends on the (end) *User* ***Role*** for obtaining the *Structured User Story* ***Work Product***; that is why it is represented as a Resource dependency in the i* diagram of Figure 1. At the end of the *Requirements Analysis* ***Phase***, the *Software Analyst* ***Role*** furnishes a *Refined User Story Set* ***Work Product*** to the *APO* ***Role*** (see next bullet);

- The *Software Analyst* ***Role*** is in charge of understanding the software problem by studying the application domain as well as to prepare a structured view and specification of user requirements. The first process fragment ***Activities*** are performed by this role. These are performed in the context of the *Requirements Analysis* ***Phase*** which is represented as an i* Goal in Figure 1. A means-end decomposition then allows to refine the i* Goal representing the ***Phase***. Indeed, to fulfill this i* Goal, the *Software Analyst* ***Role*** performs the ***Activity*** *Build Rationale Tree*. In itself, the latter ***Activity*** requires a set of other ***Activities*** to be achieved (as shown through the decompositions in Figure 1). The first one is to *Tag the user story elements* of the *Initial User Story Set* ***Work Product***). Then, the *Software Analyst* ***Role***, can *Link the User Story elements* in the Rationale View of the CASE-Tool (see Section 5). The purpose of this ***Activity*** is to link related US elements in the Rationale View through means-end and traditional decompositions (see Appendix A.3) to build a first version of the Rationale Tree. More domain knowledge is usually needed to fully perform this ***Activity*** so that the *Software Analyst* ***Role*** needs to discuss elements with the *APO* ***Role*** or immediately with (end) *Users*. Simultaneously, the *Software Analyst* ***Role*** needs to *Ensure consistency in the User Story set* and to *Identify missing requirements*; these are two other ***Activities*** of the process fragment represented as i* Tasks in Figure 1. Concretely, several US elements may express the same requirement and some elements are not expressed in US yet are needed to ensure a consistent system. The purpose of these two last ***Activities*** is to solve these issues. As an output, the *Software Analyst* ***Role*** sets at disposal of the *APO* ***Role*** a *Refined User Story Set*

Work Product and at disposal of the *Software Architect* a *Consistent Rationale Tree* (for both of these ***Work Products*** see the i* Resource dependencies between the two related roles in Figure 1);

- The *Software Designer* ***Role*** is in charge of transforming software specifications to a software architecture and design. ***Activities*** of the process fragment are performed by this role in the context of the *Multi-Agent System Design* ***Phase*** which is represented as an i* Goal in Figure 1. A means-end decomposition then allows to refine the i* Goal representing the ***Phase***. Indeed, to fulfill this i* Goal, the *Software Designer* ***Role*** performs the ***Activity*** *Align Multi-Agent System Design with Rationale Tree*. In itself, the latter ***Activity*** requires two other ***Activities*** to be achieved (as shown through the decompositions in Figure 1). The first one is to *Define the Multi-Agent System Structure*. In our case, we point to the use of the transformation rules depicted in Section 5; these are specific to the JADE framework but other agent-oriented languages could also be adopted. Then, the *Software Designer* ***Role***, can *Specify the temporal exchange of messages* in order to give further documentation on the execution of user (or business) processes to the *Software Developper*. Both of these atomic ***Activities*** are supported in the Design View of the CASE-Tool (also depicted in Section 5). The *Multi-Agent System Design* ***Work Product*** is transmitted to the *Software Developer* ***Role*** for further implementation;

- The *Software Developer* ***Role*** is in charge of associating code (JADE code in our case) to the *Multi-Agent System Design*. ***Activities*** of the process fragment are performed by this role in the context of the *Multi-Agent System Implementation* ***Phase*** which is represented as an i* Goal in Figure 1. A means-end decomposition then allows to refine the i* Goal representing the ***Phase***. Indeed, to fulfill this i* Goal, the *Software Developer* ***Role*** performs the ***Activity*** *Implement software in an agent-oriented programming language*. This ***Activity*** requires specific technical knowledge of the agent-oriented development language in use (JADE in our case). As output, the *Software Developer* ***Role*** furnishes a *Software Unit* ***Work Product*** to the *System Tester*. The latter ***Work Product*** can be a prototype of a defined module addressing specific requirements or a (partial) final implementation to be validated. In any case it must be an executable release on which

feedback can be collected;

- The *System Tester* is in charge of ensuring the system tests for the *Software Unit* ***Work Product*** is valid. We thus focus here on evaluation related to the (end) *User* only. The *System Tester* ***Activities*** are outside the scope of our process fragment and are guided by the adopted agile method;

- The (end) *User* ***Role*** uses the software application and is thus in charge of providing the *Feedback on Software Unit* ***Work Product***. The portion of the Rationale Tree covered by the *Software Unit* ***Work Product*** (implementation) is refined on the basis of the (end) *User* ***Role*** *Feedback on Software Unit* ***Work Product*** furnished by the *System Tester* ***Role*** to the *Software Analyst* ***Role*** in the form of a *Refined User Story Subset* ***Work Product***. The *Software Analyst* ***Role*** then refines the Rationale Tree on this basis (the *Build Rationale Tree* ***Activity***).

5. Composition Guidelines for Transformation and CASE-Tool

Figure 2 graphically illustrates the transformation *Composition Guidelines*. As already said, the meta-model for the requirements analysis phase has been developed in Wautelet et al. [51]. The meta-model for the MAS Design and MAS Implementation phases is taken from Bellifemine et al. [4]. The mapping has been built following the method evoked in the Section 3.

- The US unified model *Role* element represents the actions and social behavior taken by an individual or a system; it consequently aligns with the Agent concept. Roles distinguished in US (so in the Requirements Analysis ***Phase***) thus transform into JADE Agents in the *Multi-Agent System Design* ***Phase***. In the *Multi-Agent System Implementation* ***Phase***, each instance of the Role will be a new system Agent (e.g., each individual Driver instantiates a new Driver Agent);

- Hard-goals represent the most abstract and coarse-grained functional elements present in the US. We suggest as a good practice during requirements analysis to always define a counterpart to the Hard-goal in the form of a Task that realizes that particular US element Hard-goal. In other words, our point is that the Hard-goal is part of the problem

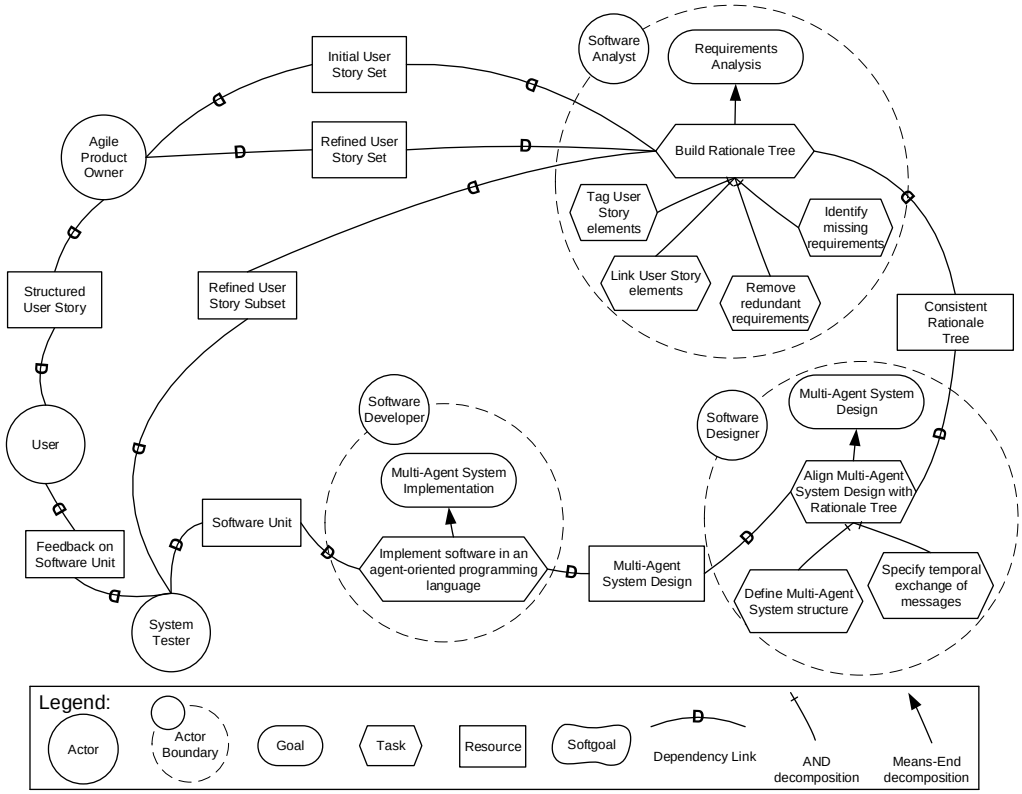


Figure 1: Process (fragment) for Integrating Agent-Oriented Development in Agile Projects.

domain so that we do not transform it as such to the MAS design; the Task corresponding to a solution to the Hard-goal and thus part of the solution domain will be transformed to the MAS design;

- Tasks and Capabilities represent (operational) functional elements of the software solution. In other words, they describe how a Role performs actions to achieve an Hard-goal or contribute to a Soft-goal. These elements constitute the core of the (future) Agent behavior; that is why they become *JADE Behaviors* in the MAS. There is nevertheless a difference between the transformation of Tasks and Capabilities to *JADE Behaviors*. Indeed, since the Task is, by nature, complex so not atomic, it is transformed into a *CompositeBehaviour* (itself composed of other behaviors) and the Capability – by nature atomic – is

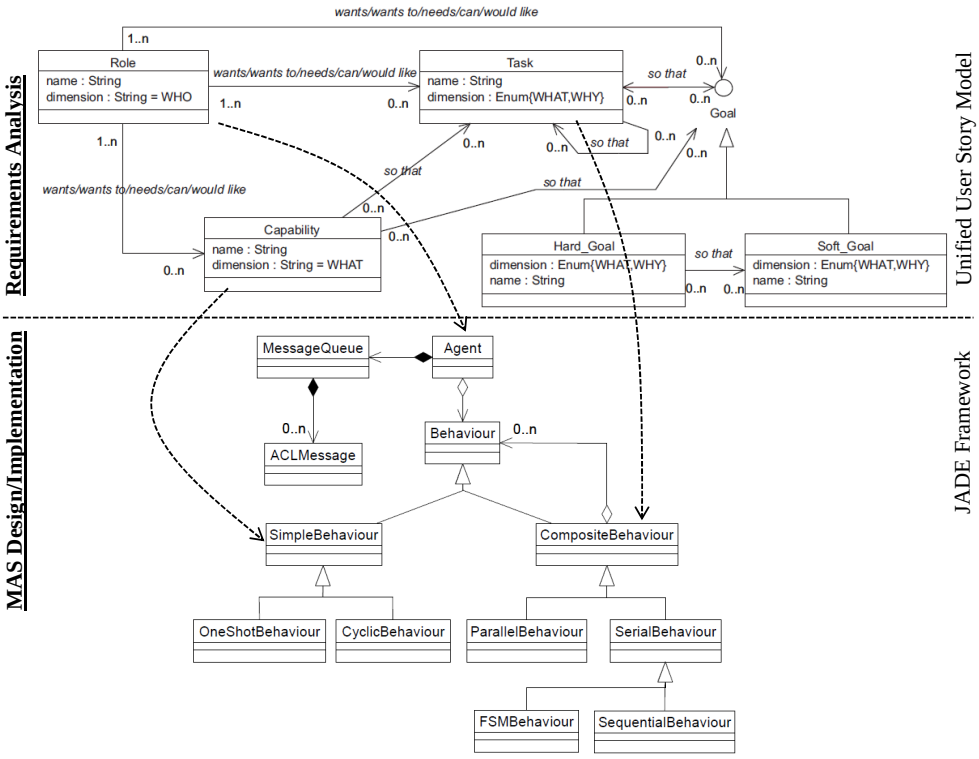


Figure 2: Transformation from US elements to MAS Design (adapted from [51, 4]).

transformed in a *SimpleBehaviour*;

- Soft-goals have no clear cut criteria to be achieved, they are considered as being "satisficed" if achieved to a degree considered acceptable [51]. The choices that will be made in the software solution and in the MAS design will thus have a positive or negative impact on the resolution of the Soft-goal itself. The Soft-goal is thus not transformed as such to the MAS design but the impact (positive or negative) of design choices could be traced from design elements like *JADE Behaviors* to the Soft-goal. This could for example be done using the Rationale tree in the fashion of the NFR framework (see [12]). This is, however, not currently supported and left for future work.

In order to support the editing of US sets on US cards as well as the Rationale Tree, we have built an add-on to the cloud version of our Descartes

Architect CASE-Tool [1]. Figure 3 illustrates two screens of the CASE-tool; for the purpose of the process fragment, three views are relevant:

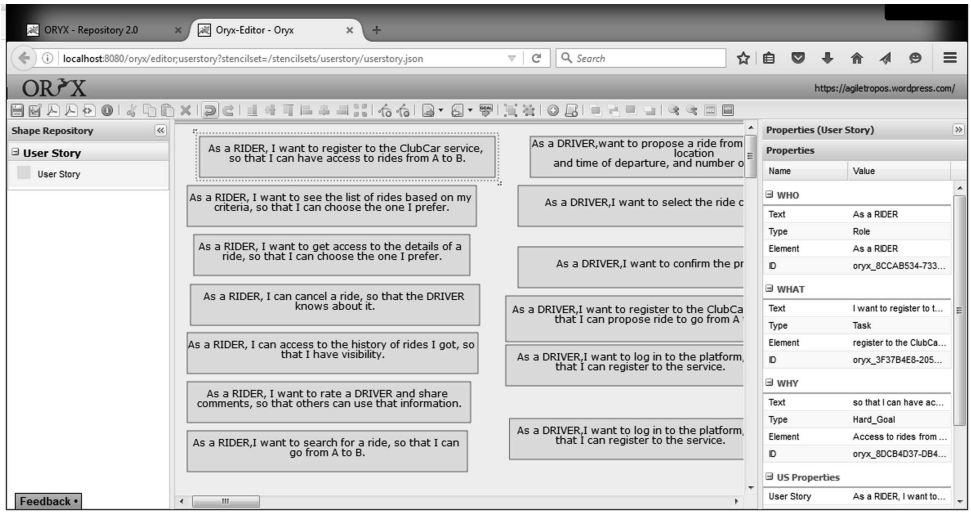
- the *User Story View* (*USV*) to edit US through virtual US cards. Each US element in a dimension must be tagged with a concept of the unified model;
- the *Rationale View* (*RV*) to edit Rationale Trees following the specification made in [52] and in this paper. The graphical elements can be automatically generated from the US defined in the USV; the modeler is then in charge of further editing the links between elements. When changes are made to graphical elements in the RV, the elements are automatically updated in the USV and vice-versa. These do indeed form a same logical element represented in different views;
- the *Design View* (*DV*) to edit a JADE Class diagrams;
- next to this, we can also edit classical UML diagrams.

Once again, as a prerequisite, the set of US needs to be tagged to start the transformation and round-trip between the views. The editing process is continuous and intensive over the *Requirements Analysis* ***Phase***[2] and to a certain extend over the entire project life cycle (see *Refined User Story* ***Work Product*** in Section 4). This is of course fully supported by the tool and leads to automatic updates of complementary views; consistency is ensured by separating the conceptual element in the CASE-Tool memory from its representation in a view so that if a change is made to an element in one view it is automatically impacted to the same element in the other view.

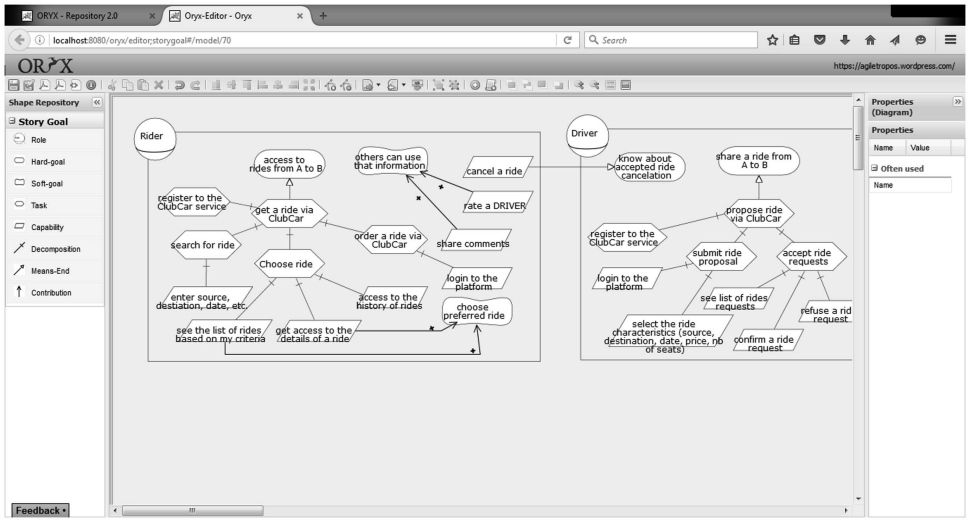
6. Illustrative Example

Our proposal will be illustrated using a running example about carpooling. Carpooling deals with sharing car journeys so that more than one person travels within a car. It, indeed, becomes increasingly important to save gas,

[2]In practice, during *Requirements Analysis* some US elements are "retagged" in an ad-hoc manner in later modeling stages. Indeed, the composition of the Rationale Tree mostly leads to reconsider the nature of some elements (of which the granularity and structure was hard to determine when only seen in an isolated manner in the first stages) like in most modeling approaches.



(a) User Story View



(b) Rationale View

Figure 3: The supporting CASE-Tool.

reduce traffic, save driving time and control pollution. *ClubCar* [41] is a multi-channel application available as an Android application, SMS service and IVR system. Users of ClubCar are riders and/or drivers that can register by SMS, voice or through an Android app. Roughly speaking, the software allows drivers to propose rides and submit their details with *dates*, *times*,

sources and *destinations* while riders can search for available rides.

The solution presented here is not considered as a full case study since the solution was designed *ex-post* of the solution built in [39]. This means that we took over the US set and made another development based on a MAS and that the MAS was not the original solution proposed for this problem. Nevertheless, this illustrative example contributes to showing the applicability and performance of the approach.

At the beginning of the *Requirements Analysis* ***Phase***, the responsibility and accountability (in the sense defined in [54]) of our process fragment starts up. The illustrative example consequently starts with that ***Phase***. Similarly, it finishes with the *Multi-Agent System Implementation* ***Phase***; ex-post activities are outside the responsibility and accountability of the process fragment and are dependent on the agile method the process fragment is embedded in.

6.1. Phase: Requirements Analysis

6.1.1. Activity: Tag User Story elements

Table 2 shows a sample of the *Initial User Story Set* ***Work Product*** that has been furnished by the *APO* ***Role***. To save some space, the US presented in the Table have already been associated with a tag; this was thus the first ***Activity*** of the *Software Analyst* ***Role***. The reader should keep in mind that there are much more US written for this project and that the scope of the project is much broader. We have made here an ad-hoc selection of a sample related to specific modules (and thus *Rationale Trees* decompositions). The sample of the US of the ClubCar application has indeed been selected to illustrate at best the research developed in this paper.

Table 2: US sample of the ClubCar application.

Dimension	*Element*	*D_C Type*
WHO	*As a RIDER,*	Role
WHAT	*I want to register to the ClubCar service,*	Task
WHY	*so that I can have access to rides from A to B.*	Hard-goal
WHO	*As a RIDER,*	Role
WHAT	*I want to see the list of rides based on my criteria,*	Capability
WHY	*so that I can choose the one I prefer.*	Soft-goal

WHO	*As a RIDER,*	Role
WHAT	*I want to get access to the details of a ride,*	Capability
WHY	*so that I can choose the one I prefer.*	Soft-goal
WHO	*As a RIDER,*	Role
WHAT	*I can cancel a ride,*	Capability
WHY	*so that the DRIVER knows about it.*	Hard-goal
WHO	*As a RIDER,*	Role
WHAT	*I can access to the history of rides I got,*	Capability
WHY	*so that I have visibility.*	Soft-goal
WHO	*As a RIDER,*	Role
WHAT	*I want to rate a DRIVER and share comments,*	Capability
WHY	*so that others can use that information.*	Soft-goal
WHO	*As a RIDER,*	Role
WHAT	*I want to search for a ride,*	Capability
WHY	*so that I can go from A to B.*	Soft-goal
WHO	*As a DRIVER,*	Role
WHAT	*I want to log in to the platform,*	Capability
WHY	*so that I can register to the service.*	Task
WHO	*As a DRIVER,*	Role
WHAT	*I want to propose a ride from A to B with the price, location and time of departure, and number of seats available.*	Task
WHO	*As a DRIVER,*	Role
WHAT	*I want to select the ride characteristics.*	Capability
WHO	*As a DRIVER,*	Role
WHAT	*I want to confirm the proposal.*	Capability
WHO	*As a DRIVER,*	Role
WHAT	*I want to register to the ClubCar service,*	Capability
WHY	*so that I can propose ride to go from A to B.*	Soft-goal

6.1.2. Activity: Link User Story elements

After the tagging of the *Initial User Story Set*, the first *Rationale Tree* can be produced. Out of the *Initial User Story Set* we generate the first *Rationale Tree* that is represented in Figure 4. At this preliminary stage, the different US elements are not yet interlinked.

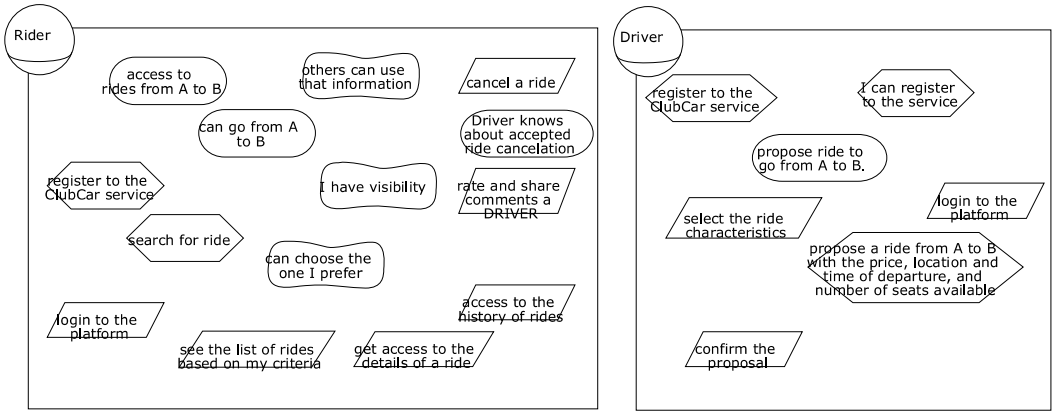


Figure 4: Initial Rationale Tree Built from the US set.

Figure 5 represents the Rationale Tree obtained after the US elements have been linked[3].). Further domain analysis is usually required to achieve such a stage, it includes more discussions with users, clients and other stakeholders. In practice, during this stage – when using the CASE-Tool – there is continuous round-trip between the US View and the Rationale View.

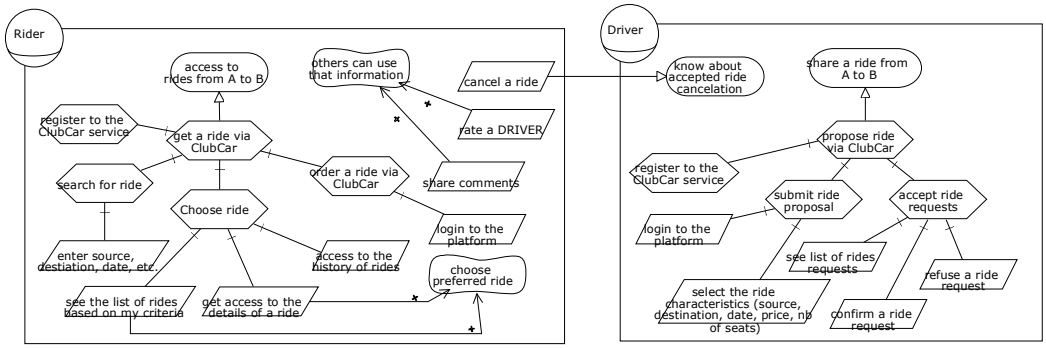


Figure 5: Refined Rationale Tree Built from the US set.

[3]In order to save space, the diagram also includes elements added during the ***Activities*** of Section 6.1.3

6.1.3. Activities: Remove redundant requirements & Identify missing requirements

Building a consistent Rationale Tree leads, for instance, to include the US containing the *Task* "propose ride via ClubCar" for the *Role* "Driver". This Task is required for linking the (more abstract) Hard-goal "share a ride from A to B" – that is part of the problem domain – to the solution domain in a means-end analysis fashion. Other US derived from elements added in the Rationale Tree are also included.

A *Consistent Rationale Tree* is the ***Work Product*** that will be used in the next stage for transformation to MAS design.

6.2. Phase: Multi-Agent System Design

6.2.1. Activity: Define Multi-Agent System Architecture

As discussed, the transformation process to the MAS Design takes roots in the Rationale Tree; elements are transformed following the rules seen previously in Section 5.

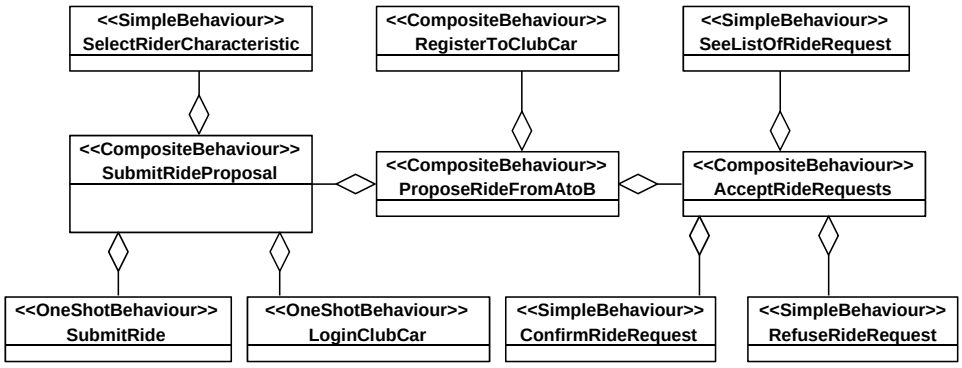


Figure 6: ClubCar JADE (Partial) Class Diagram.

The roles Driver and Rider are transformed in individual agents (i.e., Driver and Rider) in JADE. For illustrating our approach, we show here the transformed architecture for the agent Driver only. Figure 6 presents the internal architecture of the Driver agent transformed from the Rationale Tree. Typically, the Task `Propose ride via ClubCar` is transformed to a *CompositeBehaviour* `ProposeRideFromAtoB`. In addition, the latter behavior is composed of three other *CompositeBehaviours* `SubmitRideProposal`, `RegisterToClubCar`, and `AcceptRideRequest` which are respectively transformed from *Tasks* `SubmitRideProposal`, `Register to ClubCar service`

and `accept ride requests`. However, within our illustrative example we only focus on the `SubmitRideProposal` behavior. The `SubmitRideProposal` behavior is further composed of two *SimpleBehaviours* `SelectRideCharacteristic` and `LoginClubCar` which are respectively transformed from the Capabilities `Select the ride characteristics` and `login to the platform`.

The architecture obtained through the transformation process and illustrated in Figure 6 only provides the *signature* of the behaviors of the Driver agent; but not how this agent it effectively behaves (and thus reacts) when an `ACLMessage` is received; this is determined and described by the analyst on a case by case basis through domain knowledge (so not on the basis of the user stories themselves).

6.2.2. Activity: Specify temporal exchange of messages

Other diagrams like communication and dynamic diagrams (see [54]) allow to visualize the interaction between the agents when they send `ACLMessage`s. By doing so, we can we further discover the interaction between agents. This aspect remains out of the scope of the present paper because we focus on the software development process and transformation abilities.

6.3. Phase: Multi-Agent System Implementation

6.3.1. Activity: Implement software in an agent-oriented programming language

We have basically adopted the client-server architecture for implementing the ClubCar system as shown in Figure 7. The GUI constitutes the client side for interacting with the MAS implemented as a mobile application. The MAS constitutes the server side implemented with the JADE framework.

The JADE MAS, thus the server side constitutes the part transformed from the Rationale Tree. JADE uses the Java language for implementing agents and uses `ACLMessage`s for communicating between agents.

When a mobile application is launched by the end-user, an Agent is created within the JADE platform for handling all the requests from that (end) *User* (in our implementation we create a session for each application launched and the ID session has become the Agent ID for the corresponding Agent in the JADE platform). For an effective execution of the application we need to allow the mobile application to communicate directly with the Agents present in the JADE platform. When an action is performed at the level of the GUI, the Agent concerned with the transaction receives what the mobile application has sent. We propose in our implementation an encapsulation

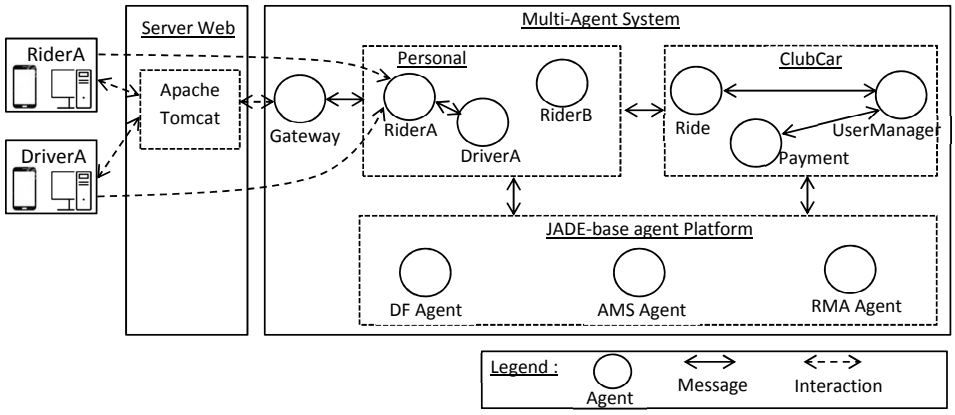


Figure 7: ClubCar Architecture.

mechanism that allows the mobile application to send or receive content to/from the concerned agent directly. This means that the agents send and receive the requests composed within the client (meaning at the level of the GUI) in real-time. We use JSON formats for communication between the mobile application and the MAS since this format is popular in web technologies and light weight compared to other formats like XML. In addition, it allows the mobile application to directly update its interface after getting the data from the corresponding agent. In order to achieve this, the request of the client is, as a first step, encapsulated in the HTTP request. When the Servlet receives any HTTP request from the client, it reads the content of this request in JSON data format and writes it into a Java Object. Then, as a second step, it sends that object to the JADE gateway.

Finally, the Gateway reads the JSON data from the Java Object sent and builds the `ACLMessage` using the JSON data. The `ACLMessage` is sent to the corresponding agent. Since JSON can also be operated in JADE, we do not re-translate the client content into the XML format that is normally used in the JADE platform. An example of the JSON format encapsulated in `ACLMessage` is given in Figure 8. The ontology-based communication is out of the scope of this paper and implementation.

7. Related Work

First of all, in line with [14, 10], our work can be said to be model-driven because the MAS implementation is partially built and deduced from a high

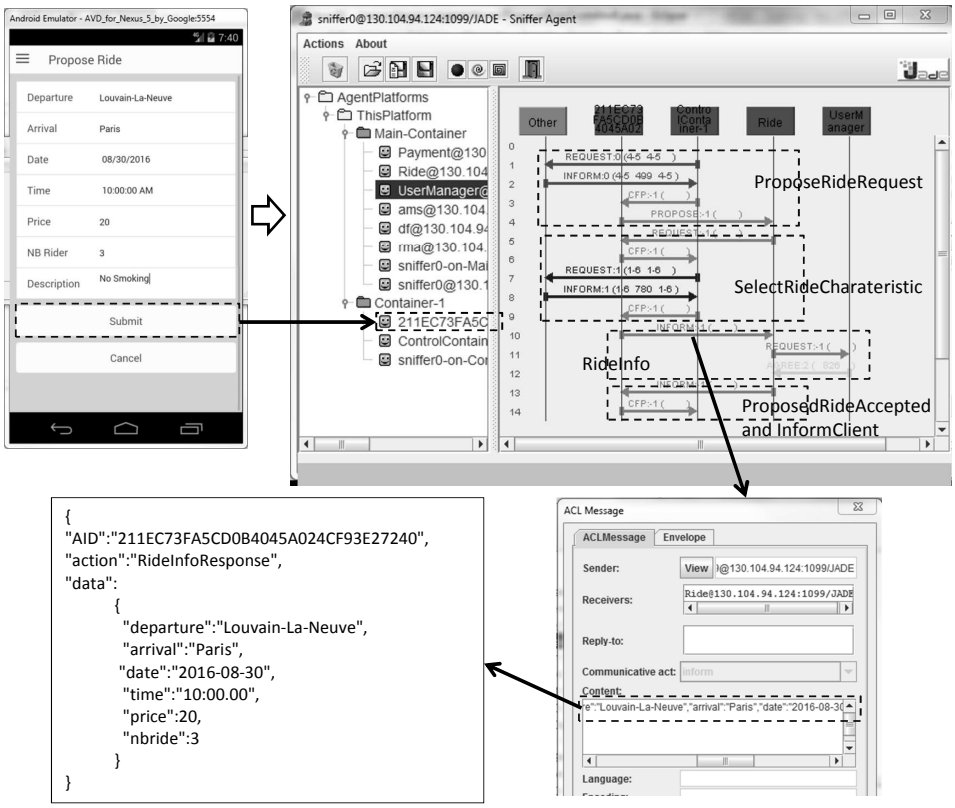


Figure 8: ClubCar Application.

level analysis model, the Rationale Tree. We thus start with further studying the model driven software development for MAS.

Hahn et al. [26] propose a meta-model for agent systems that abstracts from existing agent-oriented methodologies, programming languages and platforms. It defines the abstract syntax of a proposed domain specific modeling language for MAS; the latter is a base to generate code in agent-oriented languages like JACK [55] or JADE. Their approach is not immediately comparable to ours since we directly aim to generate code in a specific language without having an in-depth platform independent MAS design. The idea of such a middle layer has been studied in [53] but we aim to generate code as fast as possible to be complied with agile principles. Similarly, [9, 22, 10] have built a domain specific modeling language for abstracting and supporting the development of a MAS. Their approach is nevertheless deeply anchored in

the semantic web paradigm because it specifically takes into account the interactions of semantic web agents with semantic web services. It is thus more specific than for example [26]. Besides, [9, 22, 10] also depict MAS analysis in terms of high level structures like Roles, Goals, Plans or even Capabilities. Nevertheless they devote specific effort to design a semantic web solution. Once again, we do not spend time making an enhanced design stage and turn immediately to code.

Then, [19] proposes a meta-model for the integration of diverse agent-oriented modeling languages that use the powertype pattern (e.g., Tropos, Prometheus, Ingenias and AUML). Our unified model for user story based modeling could be an instance of the inter-methodology MAS metamodel proposed in [19]. This shows the suitability of the analysis stage to be aligned with a MAS implementation. Moreover, [25] extends the MAS-ML metamodel and enhance its tool to support the modeling of not only proactive agents but also several other architectures described in the literature. Their contribution is mainly located in the detailed design of MAS and goes deep in the possible agents description. We do not intend to do this to keep our practices as agile as possible and immediately target a specific implementation language. Also, their aim is the integration of heterogeneous sources; we essentially target collaborating agents in an homogeneous environment.

We can also focus on the use of US in agent methods. In Agile PASSI [11], US are used as requirement artifact for communication but it is their only usage. US, specifically in MaSE [15], is one source of requirements for capturing the goal of the agent [56] but without formal transformation. [21] uses fuzzy theory to provide a systematic and methodical approach to explore various dependencies such as goal, task, resource or Soft-goal from US. Again, the technique is limited to the analysis stage with no transformation to design. In contrast, Tenso et al .[43] adopt agent-oriented modeling in agile development. They provide a method for decomposing a goal into sub-goals and link them to US elements. Carrera et al. [6] use US as testing mechanism for agent-oriented software development. Only one US template is used and aligned to JBehave (http://jbehave.org).

Prometheus and INGENIAS [20] are two agent-oriented methods allowing to generate JADE code on the basis of a MAS design. We do not recommend to build a full generic MAS architecture like in these methods before generating code but rather to faster go to the basic code produced and complete it immediately in the JADE development language to be consistent with agile principles.

Finally, our work can also be compared to Tropos [5, 7] because Tropos builds a MAS out of an i* Strategic Rationale Diagram [57] (SRD). Our Rationale Tree is largely inspired by the SRD structure. Nevertheless, Tropos distinguishes explicitly a set of design artifacts that need to be built as a middle layer before implementing the MAS. We do not point to such a stage but rather immediately bridge our Rationale Tree to a specific MAS implementation language (JADE in our case) for fast implementation and agility.

8. Validity, Threats to the Validity, Scalability of the Approach and Future Work

As already evoked, the prerequisite to the use of our approach is to tag the US when setting them up. For this first specific task, in terms of time, the investment is limited: at most a few minutes per US, including the task to encode them in the CASE-Tool. More time would then be necessary to create and edit the links between US elements in the RV. This is, however, similar to classical software solution modeling. Moreover, we dispose of a clustering algorithm based on US syntax that allows to make clusters of relating US as a first step in the analysis process; our approach is comparable to [32]. This allows to start modeling on the basis of clusters of US that are rather similar which can save considerable time when compared to building Rationale Trees on the basis of a random organization of US. This allows a more consistent first basis for US elements structuring and grouping.

A few threats to validity could also be pointed out and should be clarified in later validation of the work:

- *Accuracy in US tagging could impact building a consistent Rationale Tree.* A study on the perception of US elements' granularity using the unified model has been performed in [48]. The study has distinguished different groups of users from students to software development professionals. The results were better with experienced modelers, but identifying granularity did not lead to major issues in any group with the condition that the set of US was taken as a consistent whole. This, indeed, allows to evaluate the relative links and hierarchy of US elements leading to adequate granularity identification and thus US elements tagging. As "stand alone" elements, granularity identification makes no sense and is almost random;

- *Accuracy of the Rationale Tree with respect to the understanding of user requirements.* The Rationale Tree is built out of the initial US set thus derived from the primary source of requirements in the agile project. As said, further domain analysis is made during the *Requirements Analysis* ***Phase*** in order to establish dependencies between US elements and identify gaps in the software solution. The Rationale Tree is not an adequate ***Work Product*** to be used for requirements validation with Users; nevertheless early prototypes generated on its basis can be quickly validated by users in order to evaluate its adequacy and possible changes to be made in the requirements and their understanding. The highly iterative nature of agile methods thus limit this second threat to validity.

The technique of transforming a set of US into Rationale Trees is virtually applicable to all sizes of projects. The complexity of the Rationale Trees may then vary from project to project and the larger the number of US, the larger the modeling effort required. The tricky question of scalability can thus legitimately be posed. The requirements analysis stage of our technique could be compared to *User Story Mapping* (*USM* [38]); the latter is applied to projects of any size. Splitting US through their three dimensions will induce more modeling effort but using the CASE-Tool will save effort by adding flexibility in model management comparing to build a USM on a board or on the ground as it is the case for larger projects. Building Rationale Trees requires time from the software development team so that the time spend during the requirements analysis stage will be increased. Nevertheless, the time for the design and implementation will be lowered thanks to this modeling and structuring effort.

Experience of the process application on multiple US sets showed us that the key to the successful application of the method is the distance between the *Initial User Story set* and reaching a *Consistent Rationale Tree*. The latter indeed constitutes the required artifact for be transformed into an intelligent system: if it cannot be reached, the transformation will not deliver a satisfying prototype. More work should then be made on the building of the *Consistent Rationale Tree*. So far we considered reaching consistency in the Rationale Tree through domain analysis, fast(er) prototyping should/could also be envisaged to reach such results. The Rationale Tree could then evolve from iteration to iteration in order to reach consistency. In any case, at least basic domain analysis is required to link elements to reduce the field

of possible organizations between US elements and rapidly converge to a relevant solution.

Future work also includes the application of the technique on more real-life case studies. We will notably proceed to a statistical analysis of the stakeholder's perception of the relevancy and contribution of the Rationale Tree for projects they have worked on as well as the pros and cons of the agent-oriented prototype. The cost of the approach in terms of time and resources and effort also still needs to be evaluated.

9. Conclusion

Agile development methodologies mostly focus on techniques to manage a software project in a very intuitive and down to earth way; they are not bound to a particular implementation technology or paradigm. Since requirements are expressed in a very informal and operational way through US, lots of elements relative to the software problem and solution are mixed into the project backlog. What, at first, may seem to be a huge drawback in the perspective of structuring and studying requirements can in fact also be seen as an interesting advantage. Indeed, US are expressed in a directive way close to the way users and other stakeholders behave in a real life organizational environment. If the US set's elements interdependency can be established, realization scenarios can be built. The completeness and redundancy of these scenarios can be overseen as well as means-end alternatives. This is precisely the way abstract intelligent reasoning systems work and, with the use of agent-technology, we can map organizational and system behavior. US sets, made consistent in the Rationale Tree, can thus be aligned with an MAS design and be implemented in an agent-oriented development language like JADE.

The process fragment overviewed in this paper takes advantage of the organizational and operational aspects of US to build a model-driven software development methodology. As evoked in the paper, the success of the method's application is mainly dependent on the quality of the first US set used as input; preliminary work is thus required from the *APO* ***Role*** to deliver a qualitative US set. Further work includes the evaluation of the maximum distance between the initial US set and a consistent Rationale Tree for the process fragment to deliver value successfully. Current findings have been illustrated on the development of an android app for carpooling.

Appendix A. Requirements Analysis: Building a Consistent Rationale Tree from a User Story Set

This section positions the use that we make in our agent process fragment of previous contributions at the level of the requirements analysis stage. First of all, Section Appendix A.1 positions the US concept and how we tackle US sets. The goal is to start from US sets and to develop a consistent requirements model providing reasoning abilities that can lead to the design (and implementation) of a MAS. To this end, we tag the US set using the unified model of US templates (see Section Appendix A.2) and generate a first Rationale Tree (see Section Appendix A.3) that provides the relationships between US elements. Further domain analysis is required to enhance the Rationale Tree since the initial US set cannot furnish all the required information for building a consistent Rationale Tree: round-tripping between the US-set view and the Rationale Tree view is then made (see Section 6.1).

Appendix A.1. Research Structure: Decomposing a User Story in Descriptive Concepts

Figure A.9 depicts a meta-model that describes the conceptual foundations of the requirements analysis stage of our process fragment. A full description of the meta-model can be found in [52], we only point here out the following elements.

When eliciting requirements, we do not take US as a whole but decompose it on the basis of their *WHO*, *WHAT* and – when available – *WHY* dimensions. These elements are all referred to as *Descriptive_Concepts* (*D_C*) in our research. When a US set is decomposed into *D_C*, possible (decomposition) relationships between various *D_C* are identified through domain analysis (see Appendix A.2). To represent the fact that various *D_C* can be linked, the *Link* class is introduced in the meta-model. The possible instances of the *Link* class are defined in Appendix A.3.

Appendix A.2. Unified-Model of User Stories' Descriptive_Concepts

As evoked, [51] suggests to build a unified model for designing US templates. The interested reader can refer to the latter reference for the research details and process, and we use this model as reference. Figure A.10 shows the meta-model of US templates. A US template can be designed taking an element from the WHO, WHAT and possibly WHY dimensions.

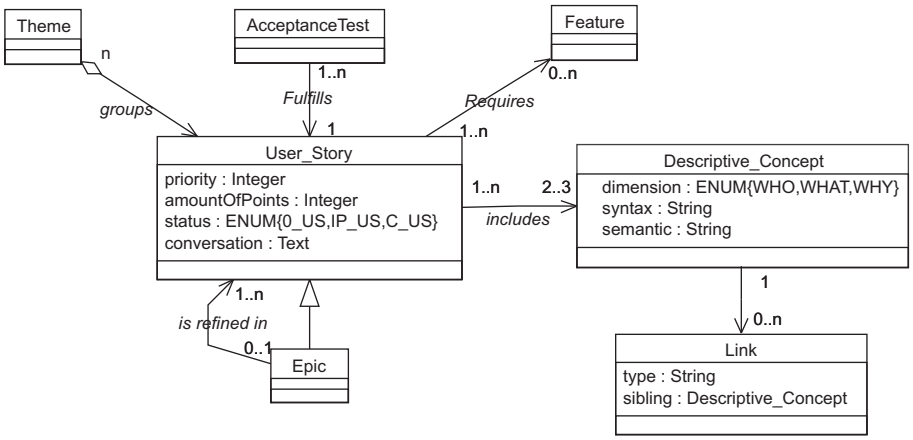


Figure A.9: A Meta-Model for User Story Structuring (from [52]).

The link between the classes conceptually represents the link from one dimension to the other. Concretely, the unidirectional association from the *Role* to one of the *Capability*, *Task* or *Goal* classes implies that the target class instantiates an element of the WHAT dimension (always tagged as *wants/wants to/needs/can/would like* in the model). This means that "As a *Role*, I want/want to/need/can/would a *Capability*/*Task*/*Goal*" are three possible instances of US template valid with respect to the unified US model; the *Role* is thus in the WHO dimension and the *Capability*, *Task* or *Goal* is in the WHAT dimension of the US. Then, the unidirectional association from one of these classes instantiating the WHAT dimension to one of the classes instantiating the WHY dimension (always tagged as *so that* into the model) implies that the target class possibly (since 0 is the minimal cardinality) instantiates an element of the WHY dimension. In other words, the templates "As a *Role*, I want/want to/need/can/would a *Capability*" and "As a *Role*, I want/want to/need/can/would a *Capability* so that *Goal*" are two valid templates of the US model. The second one includes a WHY dimension and the first one not. Another US template supported by our model is for instance: *As a <Role>, I would like <Task> so that <Hard-goal>*.

Each concept is associated with a particular syntax (identical to the name of the class in Figure A.10) and a semantic. The syntax and semantics of the model are summarized here. As a result of the research conducted in [51], the couples syntax/semantic are the following:

- *A Role is an abstract characterization of the behavior of a social actor*

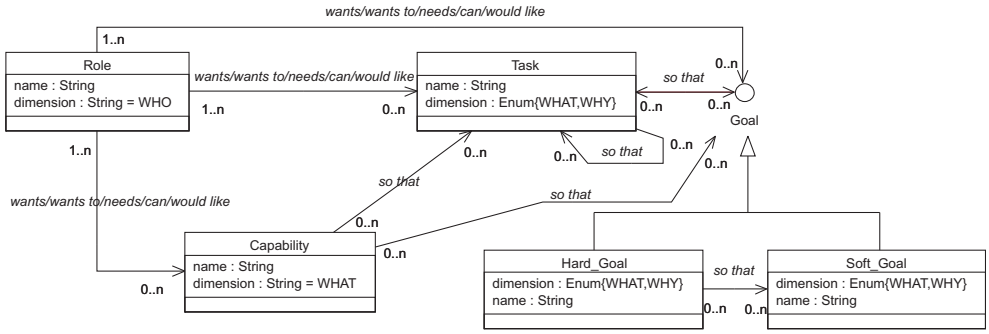


Figure A.10: Unified Model for User Story Descriptive Concepts (from [51]).

within some specialized context or domain of endeavor;

- *A Task specifies a particular way of attaining a goal*;
- *A Capability represents the ability of an actor to define, choose, and execute a plan for the fulfillment of a goal, given certain world conditions and in the presence of a specific event*;
- *A Hard-goal is a condition or state of affairs in the world that the stakeholders would like to achieve*;
- *A Soft-goal is a condition or state of affairs in the world that the actor would like to achieve. But unlike a hard-goal, there are no clear-cut criteria for whether the condition is achieved, and it is up to the developer to judge whether a particular state of affairs in fact achieves sufficiently the stated Soft-goal.*

More information about the elements, their granularity and relative position can be found in [51, 52, 50].

Appendix A.3. A Rationale Tree for User Story Sets Representation

[52] has developed a graphical representation in the form of a Rationale Tree for US sets tagged with the unified model for US templates depicted previously. These used icons as well as the possible links between the US elements are summarized in Figure A.11.

Further comments and explanations can be given about the different relationships (links) that we can have between the different US elements. These are links between elements of the WHAT and/or of the WHY dimension. [52] distinguishes 3 types of links:

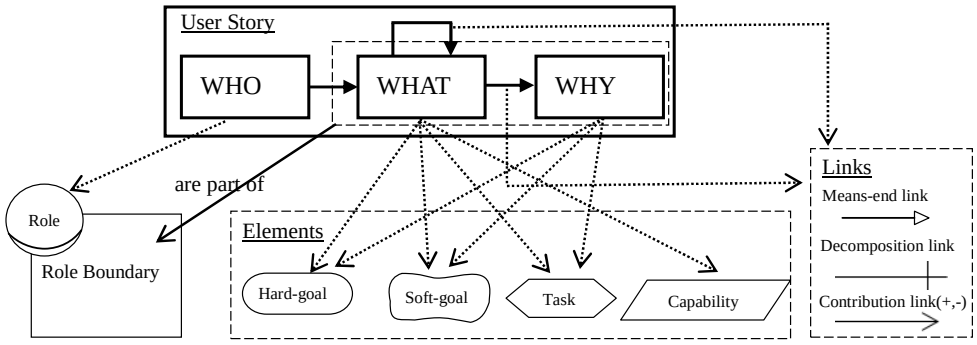


Figure A.11: Icons Used Within the Representation of the US Elements Using the Strategic Rationale Reasoning (from [52]).

- Means-end links which *indicate a relationship between an end, and a means for attaining it. The "means" is expressed in the form of a task, since the notion of task embodies how to do something, with the "end" is expressed as a goal. In the graphical notation, the arrowhead points from the means to the end* [57];

- Decomposition links are more specifically associated to tasks, indeed *a task element is linked to its component nodes by decomposition links* [57]. Moreover, *A task can be decomposed into four types of elements: a Sub-goal, a Sub-task, a Resource, and/or a Soft-goal - corresponding to the four types of elements. The task can be decomposed into one to many of these elements...* [57];

- Contribution links for contributions to Soft-goals, indeed *any of these Contribution Links can be used to link any of the elements to a Soft-goal to model the way any of these Elements contributes to the satisfaction or fulfillment of the Soft-goal* [57].

The modeler has to create the links between the D_C in function of the requirements/domain analysis. The study and linking of elements lead to a tree hierarchy. That way, an analysis (i) of the alternatives (means-end), (ii) of the possibly missing elements and (iii) of the possible redundant elements (in the decompositions) can be performed.

[1] The descartes architect case-tool, 2016.

[2] Anonymous. Software & systems process engineering meta-model specification. version 2.0. Technical report, Object Management Group, 2008.

[3] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2Nd Edition)*. Addison-Wesley Professional, 2004.

[4] F.L. Bellifemine, G. Caire, and D. Greenwood. *Developing multi-agent systems with JADE*, volume 5. Wiley, 2007.

[5] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[6] Álvaro Carrera, Carlos A Iglesias, and Mercedes Garijo. Beast methodology: An agile testing methodology for multi-agent systems based on behaviour driven development. *Info. Syst. Frontiers*, 16(2):169–182, 2014.

[7] Jaelson Castro, Manuel Kolp, and John Mylopoulos. Towards requirements-driven information systems engineering: the tropos project. *Inf. Syst.*, 27(6):365–389, 2002.

[8] H. Frank Cervone. Understanding agile project management methods using scrum. *OCLC Systems & Services*, 27(1):18–22, 2011.

[9] Moharram Challenger, Sebla Demirkol, Sinem Getir, Marjan Mernik, Geylani Kardas, and Tomaz Kosar. On the use of a domain-specific modeling language in the development of multiagent systems. *Eng. Appl. of AI*, 28:111–141, 2014.

[10] Moharram Challenger, Marjan Mernik, Geylani Kardas, and Tomaz Kosar. Declarative specifications for the development of multi-agent systems. *Computer Standards & Interfaces*, 43:91–115, 2016.

[11] Antonio Chella, Massimo Cossentino, Luca Sabatucci, and Valeria Seidita. Agile passi: An agile process for designing agents. *International Journal of Computer Systems Science & Engineering*, 21(2):133–144, 2006.

[12] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. On non-functional requirements in software engineering. In *Conceptual Modeling: Foundations and Applications - Essays in Honor of John Mylopoulos*, pages 363–379, 2009.

[13] Mike Cohn. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional, 1st edition, 2009.

[14] Alberto Rodrigues da Silva. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, 43:139–155, 2015.

[15] Scott A DeLoach, Eric T Matson, and Yonghua Li. Applying agent oriented software engineering to cooperative robotics. In *FLAIRS Conference*, pages 391–396, 2002.

[16] Torgeir Dingsøyr, Sridhar P. Nerur, Venugopal Balijepally, and Nils Brede Moe. A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85(6):1213–1221, 2012.

[17] T. Tung Do, Manuel Kolp, Stéphane Faulkner, and Alain Pirotte. Agent-oriented design patterns. In *ICEIS 2004, Proceedings of the 6th International Conference on Enterprise Information Systems, Porto, Portugal, April 14-17, 2004*, pages 48–53, 2004.

[18] Stéphane Faulkner, Manuel Kolp, Yves Wautelet, and Youssef Achbany. A formal description language for multi-agent architectures. In Manuel Kolp, Brian Henderson-Sellers, Haralambos Mouratidis, Alessandro Garcia, Aditya Ghose, and Paolo Bresciani, editors, *Agent-Oriented Information Systems IV, 8th International Bi-Conference Workshop, AOIS 2006, Hakodate, Japan, May 9, 2006 and Luxembourg, Luxembourg, June 6, 2006, Revised Selected Papers*, volume 4898 of *Lecture Notes in Computer Science*, pages 143–163. Springer, 2006.

[19] Iván García-Magariño. Towards the integration of the agent-oriented modeling diversity with a powertype-based language. *Computer Standards & Interfaces*, 36(6):941–952, 2014.

[20] José Manuel Gascueña and Antonio Fernández-Caballero. Prometheus and INGENIAS agent methodologies: A complementary approach. In

Michael Luck and Jorge J. Gómez-Sanz, editors, *Agent-Oriented Software Engineering IX, 9th International Workshop, AOSE 2008, Estoril, Portugal, May 12-13, 2008, Revised Selected Papers*, volume 5386 of *Lecture Notes in Computer Science*, pages 131–144. Springer, 2008.

[21] Vibha Gaur and Anuja Soni. A novel approach to explore inter agent dependencies from user requirements. *Procedia Technology*, 1:412–419, 2012.

[22] Sinem Getir, Moharram Challenger, and Geylani Kardas. The formal semantics of a domain-specific modeling language for semantic web enabled multi-agent systems. *Int. J. Cooperative Inf. Syst.*, 23(3), 2014.

[23] Paolo Giorgini, Manuel Kolp, and John Mylopoulos. Multi-agent and software architectures: A comparative case study. In *Agent-Oriented Software Engineering III, Third International Workshop, AOSE 2002, Bologna, Italy, July 15, 2002, Revised Papers and Invited Contributions*, pages 101–112, 2002.

[24] Martin Glinz. A glossary of requirements engineering terminology, version 1.4. 2012.

[25] Enyo José Tavares Gonçalves, Mariela I. Cortés, Gustavo Augusto Lima de Campos, Yrleyjânder Salmito Lopes, Emmanuel Sávio Silva Freire, Viviane Torres da Silva, Kleinner Silva Farias de Oliveira, and Marcos Antonio de Oliveira. MAS-ML 2.0: Supporting the modelling of multi-agent systems with different agent architectures. *Journal of Systems and Software*, 108:77–109, 2015.

[26] Christian Hahn, Cristián Madrigal-Mora, and Klaus Fischer. A platform-independent metamodel for multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 18(2):239–266, 2009.

[27] Jim Highsmith and Alistair Cockburn. Agile software development: The business of innovation. *IEEE Computer*, 34(9):120–122, 2001.

[28] Manuel Kolp, Stéphane Faulkner, and Yves Wautelet. Social structure based design patterns for agent-oriented software engineering. *IJIIT*, 4(2):1–23, 2008.

[29] Manuel Kolp, Paolo Giorgini, and John Mylopoulos. Organizational multi-agent architectures: a mobile robot example. In *The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings*, pages 94–95, 2002.

[30] Manuel Kolp and John Mylopoulos. Software architectures as organizational structures. In *Proc. ASERC Workshop on The Role of Software Architectures in the Construction, Evolution, and Reuse of Software Systems, Edmonton, Canada*, 2001.

[31] Olga Liskin, Raphael Pham, Stephan Kiesling, and Kurt Schneider. Why we need a granularity concept for user stories. In Giovanni Cantone and Michele Marchesi, editors, *Agile Processes in Software Engineering and Extreme Programming - 15th International Conference, XP 2014, Rome, Italy, May 26-30, 2014. Proceedings*, volume 179 of *LNBIP*, pages 110–125. Springer, 2014.

[32] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn E. M. van der Werf, and Sjaak Brinkkemper. AQUSA: the automatic quality user story artisan for agile software development. In *Joint Proceedings of REFSQ-2016 Workshops, Doctoral Symposium, Research Method Track, and Poster Track co-located with the 22nd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2016), Gothenburg, Sweden, March 14, 2016.*, 2016.

[33] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn E. M. van der Werf, and Sjaak Brinkkemper. The use and effectiveness of user stories in practice. In Maya Daneva and Oscar Pastor, editors, *Requirements Engineering: Foundation for Software Quality - 22nd International Working Conference, REFSQ 2016, Gothenburg, Sweden, March 14-17, 2016, Proceedings*, volume 9619 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2016.

[34] Haralambos Mouratidis, Manuel Kolp, Stéphane Faulkner, and Paolo Giorgini. A secure architectural description language for agent systems. In *4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25-29, 2005, Utrecht, The Netherlands*, pages 578–585, 2005.

[35] John Mylopoulos, Manuel Kolp, and Paolo Giorgini. Agent-oriented software development. In *Methods and Applications of Artificial Intelligence, Second Hellenic Conference on AI, SETN 2002. Thessaloniki, Greece, April 11-12, 2002, Proceedings*, pages 3–17, 2002.

[36] OMG. Business Process Model and Notation (BPMN). version 2.0.1. Technical report, Object Management Group, 2013.

[37] Frauke Paetsch, Armin Eberlein, and Frank Maurer. Requirements engineering and agile software development. In *12th IEEE International Workshops on Enabling Technologies (WETICE 2003), Infrastructure for Collaborative Enterprises, 9-11 June 2003, Linz, Austria*, pages 308–313. IEEE Computer Society, 2003.

[38] Jeff Patton and Peter Economy. *User Story Mapping: Discover the Whole Story, Build the Right Product*. O'Reilly Media, Inc., 1st edition, 2014.

[39] Christelle Scharff, Samedi Heng, and Vidya Kulkarni. On the difficulties for students to adhere to scrum on global software development projects: preliminary results. In Stuart R. Faulk, David M. Weiss, Michal Young, and Lian Yu, editors, *Proceedings of the Second International Workshop on Collaborative Teaching of Globally Distributed Software Development, CTGDSD 2012, Zurich, Switzerland, June 9, 2012*, pages 25–29. IEEE, 2012.

[40] Valeria Seidita, Massimo Cossentino, and Antonio Chella. A proposal of process fragment definition and documentation. In Massimo Cossentino, Michael Kaisers, Karl Tuyls, and Gerhard Weiss, editors, *Multi-Agent Systems - 9th European Workshop, EUMAS 2011, Maastricht, The Netherlands, November 14-15, 2011. Revised Selected Papers*, volume 7541 of *Lecture Notes in Computer Science*, pages 221–237. Springer, 2011.

[41] Maninder Pal Kaur Shergill and Christelle Scharff. Developing multichannel mobile solutions for a global audience: The case of a smarter energy solution. *SARNOFF'12, New Jersey*, 2012.

[42] Herbert A. Simon. *The Sciences of the Artificial (3rd Ed.)*. MIT Press, Cambridge, MA, USA, 1996.

[43] Tanel Tenso and Kuldar Taveter. Requirements engineering with agent-oriented models. In *ENASE*, pages 254–259, 2013.

[44] Dan Turk, Robert B. France, and Bernhard Rumpe. Assumptions underlying agile software development processes. *CoRR*, abs/1409.6610, 2014.

[45] Daniel E. Turk, Robert B. France, and Bernhard Rumpe. Assumptions underlying agile software-development processes. *J. Database Manag.*, 16(4):62–87, 2005.

[46] Axel van Lamsweerde. Goal-oriented requirements enginering: A roundtrip from research to practice. In *12th IEEE International Conference on Requirements Engineering (RE 2004), 6-10 September 2004, Kyoto, Japan*, pages 4–7. IEEE Computer Society, 2004.

[47] Axel Van Lamsweerde. *Requirements engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[48] Mattijs Velghe. Requirements engineering in agile methods: Contributions on user story models. Master's thesis, KU Leuven, Belgium, 2015.

[49] Kevin Vlaanderen, Slinger Jansen, Sjaak Brinkkemper, and Erik Jaspers. The agile requirements refinery: Applying scrum principles to software product management. *Information and Software Technology*, 53(1):58 – 70, 2011.

[50] Yves Wautelet, Samedi Heng, Diana Hintea, Manuel Kolp, and Stephan Poelmans. Bridging user story sets with the use case model. In Sebastian Link and Juan Trujillo, editors, *Advances in Conceptual Modeling - ER 2016 Workshops, AHA, MoBiD, MORE-BI, MReBA, QMMQ, SCME, and WM2SP, Gifu, Japan, November 14-17, 2016, Proceedings*, volume 9975 of *Lecture Notes in Computer Science*, pages 127–138, 2016.

[51] Yves Wautelet, Samedi Heng, Manuel Kolp, and Isabelle Mirbel. Unifying and extending user story models. In Matthias Jarke, John Mylopoulos, Christoph Quix, Colette Rolland, Yannis Manolopoulos, Haralambos Mouratidis, and Jennifer Horkoff, editors, *CAiSE 2014, Thessaloniki, Greece, June 16-20, 2014. Proceedings*, volume 8484 of *LNCS*, pages 211–225. Springer, 2014.

[52] Yves Wautelet, Samedi Heng, Manuel Kolp, Isabelle Mirbel, and Stephan Poelmans. Building a rationale diagram for evaluating user story sets. In *Tenth IEEE International Conference on Research Challenges in Information Science, RCIS 2016, Grenoble, France, June 1-3, 2016*, pages 1–12. IEEE, 2016.

[53] Yves Wautelet, Samedi Heng, Manuel Kolp, and Christelle Scharff. Towards an agent-driven software architecture aligned with user stories. In H. Jaap van den Herik and Joaquim Filipe, editors, *Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART 2016), Volume 2, Rome, Italy, February 24-26, 2016.*, pages 337–345. SciTePress, 2016.

[54] Yves Wautelet and Manuel Kolp. Business and model-driven development of BDI multi-agent systems. *Neurocomputing*, 182:304–321, 2016.

[55] Michael Winikoff. Jack$^{\text{tm}}$ intelligent agents: An industrial strength platform. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 175–193. Springer, 2005.

[56] Mark F Wood and Scott A DeLoach. An overview of the multiagent systems engineering methodology. In *Agent-Oriented Soft. Eng.*, pages 207–221. Springer, 2001.

[57] Eric Yu, Paolo Giorgini, Neil Maiden, and John Mylopoulos. *Social Modeling for Requirements Engineering*. MIT Press, 2011.

[58] Eric S. K. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *3rd IEEE International Symposium on Requirements Engineering (RE'97), January 5-8, 1997, Annapolis, MD, USA*, pages 226–235. IEEE Computer Society, 1997.

[59] Eric S. K. Yu. Social modeling and i*. In *Conceptual Modeling: Foundations and Applications - Essays in Honor of John Mylopoulos*, pages 99–121, 2009.