User requirements modeling and analysis of software-intensive systems

Michel dos Santos Soares a,∗, Jos Vrancken b, Alexander Verbraeck b

a Universidade Federal de Uberlândia, P.O. Box 593, 38400-902 Uberlândia, Brazil
b Delft University of Technology, P.O. Box 5015, NL 2600 GA, Delft, The Netherlands

ARTICLE INFO

Article history:
Received 20 July 2010
Received in revised form 7 October 2010
Accepted 14 October 2010
Available online xxx

Keywords:
Requirements
UML
SysML
Software-intensive systems

ABSTRACT

The increasing complexity of software systems makes Requirements Engineering activities both more important and more difficult. This article is about user requirements development, mainly the activities of documenting and analyzing user requirements for software-intensive systems. These are modeling activities that are useful for further Requirements Engineering activities. Current techniques for requirement modeling present a number of problems and limitations. Based on these shortcomings, a list of requirements for requirements modeling languages is proposed. The proposal of this article is to show how some extensions to SysML diagrams and tables can fulfill most of these requirements. The approach is illustrated by a list of user requirements for a Road Traffic Management System.

© 2010 Elsevier Inc. All rights reserved.

1. Introduction

Software-intensive systems (Wirsing et al., 2008; Tiako, 2008; Hinchey et al., 2008) are large, complex systems in which software is an essential component, interacting with other software, systems, devices, actuators, sensors and with people. Being an essential component, software influences the design, construction, deployment, and evolution of the system as a whole (ANSI/IEEE, 2000). These systems are in widespread use and their impact on society is increasing. Developments in engineering software-intensive systems have a large influence on the gains in productivity and prosperity that society has seen in recent years (Dedrick et al., 2003). Their complexity is increased due to the large number of elements and reliability factors. Thus, they must be decomposed into several smaller components in order to manage complexity and facilitate their implementation and verification. In addition, there is a need to increase the level of abstraction, hiding whenever possible unnecessary complexity, by the intense use of models. Examples of software-intensive systems can be found in many sectors, such as manufacturing plants, transportation, military, telecommunication and health care.

More specifically, the type of software-intensive systems that are investigated in this article are the Distributed Real-Time Systems. The term Real-Time System usually refers to systems with explicit timing constraints (Gomaa, 2000; Laplante, 2004). Dijkstra (2002) recognized that some applications are concurrent in nature. In concurrent problems, there is no way of predicting which system component will provide the next input, which increases design complexity. Moreover, system components, such as sensors and actuators, are often geographically distributed in a network and need to communicate according to specific timing constraints described in requirements documents.

Requirements for software are a collection of needs expressed by stakeholders respecting some constraints under which the software must operate (Pressman, 2009; Robertson and Robertson, 2006). Requirements can be classified in many ways. The first classification used in this article is related to the level of detail (the second classification is presented in Section 7.1). In this case, the two classes of requirements are user requirements and system requirements (Sommerville, 2010). User requirements are high-level abstract requirements based on end users’ and other stakeholders’ viewpoint. They are usually written using natural language, occasionally with the help of domain specific models such as mathematical equations, or even informal models not related to any method or language (Luisa et al., 2004). The fundamental purpose of user requirements specification is to document the needs and constraints gathered in order to later develop software based on those requirements.

Systems requirements are derived from user requirements but with a detailed description of what the system should do, and are usually modeled using formal or semi-formal methods and languages. This proposed classification allows the representation of different views for different stakeholders. This is good Software Engineering practice, as requirements should be written from different viewpoints because different stakeholders use them for various purposes.

* Corresponding author.
E-mail addresses: michel@facom.ufu.br, mics.soares@gmail.com (M.d.S. Soares).

0164-1212/$ – see front matter © 2010 Elsevier Inc. All rights reserved.

Please cite this article in press as: Soares, M.S., et al., User requirements modeling and analysis of software-intensive systems. J. Syst. Software (2010), doi:10.1016/j.jss.2010.10.020
The process by which requirements for systems and software products are gathered, analyzed, documented and managed throughout the development life cycle is called Requirements Engineering (Sommerville, 2010). Requirements Engineering is a very influential phase in the life cycle. According to the SWEBOK (Abran et al., 2004), it concerns Software Design, Software Testing, Software Maintenance, Software Configuration Management, Software Engineering Management, Software Engineering Process, and Software Quality Knowledge Areas. Requirements Engineering is generally considered in the literature as the most critical phase within the development of software (Juristo et al., 2002; Komi-Sirviö et al., 2003; Damian et al., 2004; Minor and Armarego, 2005). Dealing with ever-changing requirements is considered the real problem of Software Engineering (Berry, 2004). Already in 1973, Boehm suggested that errors in requirements could be up to 100 times more expensive to fix than errors introduced during implementation (Boehm, 1973). According to Brooks (1987), knowing what to build, which includes requirements elicitation and technical specification, is the most difficult phase in the design of software. Lutz (1993) showed that 60% of errors in critical systems were the results of requirements errors. Studies conducted by the Standish Group (TSG, 2003) and other researchers (van Genuchten, 1991; Hofmann et al., 2001) found that the main factors for problems with software projects (cost overruns, delays, user dissatisfaction) are related to requirements issues, such as lack of user input, incomplete requirements specifications, uncontrolled requirements changing, and unclear objectives. In an empirical study with 12 companies (Hall et al., 2002), it was discovered that, out of a total of 268 development problems cited, 48% (128) were requirements problems.

Requirements Engineering can be divided into two main groups of activities (Parviainen et al., 2004): (i) requirements development, including activities such as eliciting, documenting, analyzing, and validating requirements, and (ii) requirements management, including activities related to maintenance, such as tracing and change management of requirements. This article is about user requirements development, mainly the activities of documenting and analyzing user requirements for software-intensive systems. These are modeling activities that are useful for further Requirements Engineering activities. The assumption in this article is that improving requirements modeling may have a strong impact on the quality of later requirements activities, such as requirements tracing, and in the design phase.

1.1. Research question

The main research question to be answered in this article is given as follows:

How to improve user requirements modeling and analysis for software-intensive systems?

This question is mainly answered through the early introduction of graphical models, which are used to document and analyze requirements. The identification and graphical representation of requirements relationships facilitate that traces are made. This helps in uncovering the impact that changes in requirements have in the system design. Requirements are important to determine the architecture. When designing the architecture, at least part of the functional requirements should be known. In addition, the non-functional requirements that the architecture has to conform with should be made explicit.

1.2. Article outline

Initially, a subset of a list of user requirements for a Road Traffic Management System (RTMS) is presented, using natural language, to be further modeled and analyzed (Section 2). Current techniques for requirements modeling are presented in Section 3. A number of problems and limitations related to these techniques are discussed in the same section. These shortcomings led to a list of requirements for requirements modeling languages in Section 4 and the proposed approach in Section 5 to fulfill the missing characteristics of the list. From the conclusion of Section 4, the starting point for requirements modeling languages is to use SysML diagrams and tables, which are presented in detail in Section 6. Then, SysML’s constructions are extended in Section 7 and proposed to model the initial list of user requirements (Section 8). The article ends with discussion (Section 9) and conclusions (Section 10).

2. List of requirements for RTMS

The list of requirements given below is a subset from a document which contains 79 atomic requirements for RTMS (AVV, 2006). The document is a technical auditing work based on an extensive literature study and interviews, in which the stakeholders were identified. The requirements were gathered through interviews with multiple stakeholders.

The stakeholders (and the related number of requirements) were classified as: the Road Users (1), the Ministry of Transport, Public Works and Water Management (2), the Traffic Managers (10), the Traffic Management Center (8), the Task, Scenario and Operator Manager (22), the Operators (4), the Designers of the Operator’s Supporting Functions (15), and the Technical Quality Managers (17). In this article the requirements of the Traffic Manager were selected as example to be modeled using SysML diagrams and constructions in Section 8. The requirements are given as follows.

*Traffic Manager:*

- **TM1—**It is expected that software systems will be increasingly more intelligent for managing the traffic-flow in a more effective and efficient manner.
- **TM5—**To optimize traffic flow, it is expected that gradually, region-wide traffic management methods will be introduced.
- **TM6—**The traffic management systems must have a convenient access to region-wide, nation-wide, or even European-wide parameters so that the traffic-flow can be managed optimally.
- **TM7—**It must be possible for the Traffic Managers/experts to express (strategic) "task and scenario management frames", conveniently.
- **TM8—**The system should effectively gather and interpret all kinds of information for the purpose of conveniently assessing the performance of the responsible companies/organizations that have carried out the construction of the related traffic systems and/or infrastructure.
- **TM9—**The system must support the Traffic Managers/experts so that they can express various experimental simulations and analytical models.
- **TM10—**The system must enable the Traffic Managers/experts to access various kinds of statistical data.
- **TM11—**The system must enable the Traffic Managers/experts to access different kinds of data for transient cases such as incidents.
- **TM12—**The system must provide means for expressing a wide range of tasks and scenarios.
- **TM13—**The traffic management will gradually evolve from object management towards task and scenario management.

3. Requirements modeling approaches

There are several approaches to modeling requirements. Basically, these approaches can be classified as graphics-based, purely
textual, or a combination of both. Some are generic while others are part of a specific methodology.

The most common approach is to write user requirements using natural language. The advantage is that natural language is the main mean of communication between stakeholders. However, problems such as imprecision, misunderstandings, ambiguity and inconsistency are common when natural language is used (Kamsties, 2005).

With the purpose of giving more structure to requirements documents, structured natural language is used (Cooper and Ito, 2002). Nevertheless, structured natural language is neither formal nor graphical, and can be too much oriented to algorithms and specific programming languages. Other collateral effects are that structured specifications may limit too early the programmers' freedom, and are mostly tailored towards procedural languages, being less suitable for some modern languages and paradigms.

User Stories have been used as part of the eXtreme Programming (XP) (Beck, 1999) agile methodology. They can be written by the customer using non-technical terminology, in the format of sentences using natural language. Although XP offers some advantages in the Requirements Engineering process in general, such as user involvement and defined formats for user requirements and tasks, requirements are still loosely related, not graphically specified, and oriented to a specific methodology.

A well-known diagram used for requirements modeling are the Use Cases. Even before UML emerged as the main Software Engineering modeling language, Use Cases were already a common practice for graphically representing functional requirements in other methodologies, such as Object-Oriented Software Engineering (OOSE) (Jacobson, 1992). Use Cases have some disadvantages and problems (Simons, 1999). They are applied mainly to model functional requirements and are not very helpful for other types of requirements, such as non-functional ones (Soares and Vrancken, 2007). Use Case diagrams lack well-defined semantics, which may lead to differences in interpretation by stakeholders. For instance, the include and extend relationships are considered similar, or even the inverse of each other (Jacobson, 2004). In addition, Use Cases may be misused, when too much detail is added, which may incorrectly transform the diagrams into flowcharts or making them difficult to comprehend.

Two SysML diagrams are distinguished as useful mainly for Requirements Engineering activities: the SysML Requirements diagram and the SysML Use Case diagram (OMG, 2008). One interesting feature of the SysML Requirements diagram is the possibility of modeling other type of requirements besides the functional ones, such as non-functional requirements. The SysML Use Case diagram is derived from the UML Use Case diagram without important modifications. In addition to these diagrams, SysML Tables can be used to represent requirements in a tabular format. Tabular representations are often used in SysML but are not considered part of the diagram taxonomy (OMG, 2008). Detailed explanation about SysML diagrams and tables for Requirements Engineering are given in Section 6.

A comparison of the aforementioned requirements modeling approaches is given in the next section. The objective is to identify shortcomings of these approaches, which is used as the starting point of the proposed approach for a solution, in Section 5.

### 4. Desirable requirements specification properties for software-intensive systems

A list of desirable requirements for requirements modeling languages, together with a mapping of common languages and techniques, is given in Table 1. This non-exhaustive list of requirements for requirements modeling languages is based on literature review presented in the introduction, on the modeling languages briefly presented in Section 3, and on specific texts about requirements (IEEE, 1998; Beck, 1999; Luisa et al., 2004; Robertson and Robertson, 2006). This list uses “(M) Must have” and “(S) Should have” for each entry of the table, according to the MoSCoW labels (Page et al., 2003) (Must, Should, Could, Want/Won’t have).

The characteristics proposed in IEEE (1998, Section 4.3) (correct, unambiguous, complete, consistent, ranked for importance, ranked for stability, verifiable, modifiable, and traceable) are related to a good Software Requirements Specification (SRS) document. In this article, these characteristics were used in the context of requirements modeling languages and techniques.

The reason for each entry of the list is given as follows.

#### 4.1. Must have requirements

The modeling languages must provide graphical means to express requirements. Common graphical models may facilitate the communication of models to stakeholders. Models must be human readable, as the multiple stakeholders involved have to understand the models. In this case a balance is necessary, as the more machine readable requirements are, the less human readable they become. In addition, as multiple stakeholders and designers with different backgrounds are involved, the modeling languages should be as methodology independent as possible.

It is well-known by Software Engineering researchers and practitioners that requirements are related to each other (Robertson and Robertson, 2006). These interactions affect various software development activities, such as release planning, change management and reuse. A study has shown that the majority of requirements are related to or influence other requirements (Carlshamre et al., 2001). Due to this fact, it is almost impossible to plan systems releases only based on the highest priority requirements, without considering which requirements are related to each other.

From a project management point of view, one important characteristic of a requirement is its priority. Prioritizing requirements is an important activity in Requirements Engineering (Davis, 2003). The purpose is to give an indication of the order in which requirements should be addressed. Another important property of a requirement from the project management point of view is to identify its risk. For instance, a manager may be interested in identifying the impact for a project if a specific requirement is not fulfilled. Risk management is basically the activity concerned with trying to

<table>
<thead>
<tr>
<th>List of requirements properties and representation techniques.</th>
<th>NL</th>
<th>SNL</th>
<th>XP</th>
<th>UC</th>
<th>RD</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M) Graphical modeling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M) Human readable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M) Independent towards methodology</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M) Relationship between requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M) Relationship requirements/design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M) Requirements risks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M) Identify types of requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M) Priority between requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M) Non-functional requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M) Grouping related requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M) Consistency</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M) Modifiable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M) Ranking requirements by stability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S) Solve ambiguity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S) Well-defined semantics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S) Machine readable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S) Correctness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S) Completeness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S) Verifiable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S) Traceable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S) Type of relationship requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please cite this article in press as: Soares, M.S., et al., User requirements modeling and analysis of software-intensive systems. J. Syst. Software (2010), doi:10.1016/j.jss.2010.10.020
detect previously risks in a project and preventing problems with specific plans. Despite their importance, non-functional requirements are usually not properly addressed in requirements modeling languages. For instance, UML Use Case diagrams are strong in modeling functional requirements. The various types of requirements must be identified in order to provide better knowledge of requirements for the stakeholders.

From the software design point of view, grouping requirements in the early phases of software development helps in identifying subsystems, components, and relationships between them. As a matter of fact, grouping requirements has a positive effect when designing the software architecture.

According to IEEE (1998), a SRS should be consistent and modifiable. In this article, these two properties of a SRS are considered of great significance, and are grouped with “Must have” requirements. The reason is that inconsistency between documents (Boehm, 1973; Pressman, 2009) and difficulty of changing requirements (Berry, 2004) are major causes for future problems during software development. A SRS is consistent if it agrees with other documents of the project, such as project management plans and system design models. Thus, the modeling language must be able to highlight conflicting requirements and non-conformances between requirements and design. A SRS is modifiable if its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style. For instance, requirements must be expressed individually, rather than intermixed with other requirements. Thus, the modeling language must be able to describe requirements in a well-structured way.

Finally, as changing requirements is a source of problems, knowing how stable a requirement is, i.e., how ready it is for further design phases, is essential.

4.2. Should have requirements

Ambiguity should be solved, as ambiguity in requirements is a major cause of misunderstandings between stakeholders and designers. Thus, modeling languages should provide well-defined semantics, which increases machine readability.

According to IEEE (1998), an SRS is correct if every requirement stated is one that the software shall meet. The user can determine if the SRS correctly reflects his/her actual needs. Thus, the modeling languages should facilitate the user in this activity.

According to IEEE (1998), an SRS is complete if all significant requirements of every type are included. Thus, the modeling languages should be able to specify all types of requirements.

According to IEEE (1998), an SRS is verifiable if there is a cost-effective process with which a person or machine can check that the software meets the requirement. In general, any ambiguous requirement is not verifiable. Thus, the modeling language should provide non-ambiguous constructions in order to facilitate that the designer can create non-ambiguous requirements models.

According to IEEE (1998), an SRS is traceable if the origin of each of its requirements is clear, and if it facilitates the referencing of each requirement in future development. Thus, the modeling language should provide means to trace the requirement through design phases. In addition, the type of these relationships should be explicit.

4.3. Resulting table

Table 1 maps the list of requirements for modeling languages discussed in this section with the modeling languages discussed in Section 3. In the table, NL stands for natural language, SNL stands for structured natural language, XP stands for the XP User Stories, UC stands for both SysML and UML Use Cases, RD stands for SysML Requirements diagram, and T stands for SysML Tables. We classified the entries as fully supported (●), half supported (●), or not supported (○) (or not easily supported, or poorly supported).

From the table, it is clear that “Must have” requirements, such as “Priority between requirements”, “Requirements risks”, “Identify types of requirements”, and “Ranking requirements by stability” are partially addressed or not addressed at all by most of the studied requirements modeling languages. The next section presents the approach followed in order to try to fulfill all the given requirements.

Another conclusion from the table is that some “Must have” requirements and the majority of “Should have” requirements are fulfilled or at least partially fulfilled by a combination of the SysML Requirements diagram and SysML Tables. Thus, a possible starting point in addressing all requirements is to extend these SysML constructions.

5. Proposed approach

With the explicit choice to use SysML, the proposal starts with detailing SysML capacities for Requirements Engineering (Section 6). In Section 7.1, a classification for each atomic requirement is proposed, avoiding the confusion of which type of requirement is written in the user requirements document. The basic SysML Requirements diagram is extended with new requirements properties such as priority. Individual requirements modeled by the SysML Requirements diagram may be combined depending on their semantics. This can be useful for the early discovery of subsystems, in project management activities such as release planning, and to propose the system architecture (Section 7). User requirements are also represented in a tabular format, which may facilitate requirements tracing during the system life cycle. This is important to know what happens when related requirements change or are deleted, which improves traceability. Finally, Use Case diagrams are used to represent the actors involved and the scenarios to be implemented (Section 8). Then, Use Cases are related to SysML Requirements using one of the proposed relationships.

Although the idea in this article is to use graphical models already in the early phases of system development, natural language is still considered important. Despite its problems, there are also advantages, as natural languages are the primary communication medium between people.

After being structured and graphically represented (Fig. 1) using SysML Tables, SysML Requirements and SysML Use Case diagrams, user requirements are detailed into system requirements, being specified using other models, such as other UML/SysML diagrams or using formal methods.

The SysML constructions (diagrams and tables) for modeling user requirements are explained in detail in the following section.

6. Modeling user requirements using SysML

The SysML Requirements diagram helps in better organizing requirements, and also shows explicitly the various kinds of relationships between different requirements. Another advantage of using this diagram is to standardize the way of specifying requirements through a defined semantics. The SysML Requirements constructs are intended to provide a bridge between traditional requirements management specifications and the other SysML models. When combined with UML for software design, the requirements constructs provided by SysML can also fill the gap between user requirements specification, normally written in natural language, and Use Case diagrams, used as initial specification of system requirements (Soares and Vrancken, 2008a).
A SysML Requirement can also appear on other diagrams to show its relationship to design. With the SysML Requirements diagram, visualization techniques are applied from the early phases of system development. The SysML Requirements diagram is a stereotype of the UML class diagram, as shown in Fig. 2.

## 6.1. Relationships between requirements with SysML

Implementing all requirements in a single system release may be unattractive because of the high cost involved, lack of sufficient staff and time, and even client and market pressures. These difficulties make prioritization a fundamental activity during the Requirements Engineering process. Prioritizing requirements is giving an indication of the order in which requirements should be considered for implementation. However, it is not always possible to plan a system release based only on the set of more important requirements due to requirements relationships. A better knowledge of requirements relationships may be useful to make more feasible release plans, to reuse requirements and to drive system design and implementation.

The SysML Requirements diagram allows several ways to represent requirements relationships. These include relationships for defining requirements hierarchy, deriving requirements, satisfying requirements, verifying requirements and refining requirements. The relationships can improve the specification of systems, as they can be used to model requirements. The relationships: hierarchy, derive, master/slave, satisfy, verify, refine and trace are briefly explained as follows.

In large, complex systems, it is common to have a hierarchy of requirements, and their organization into various levels helps in dealing with system complexity. For instance, high-level business requirements may be gradually decomposed into more detailed software requirements, forming a hierarchy. Discovering the hierarchy of requirements is an important design step in Requirements Engineering. SysML allows splitting complex requirements into more simple ones, as a hierarchy of requirements related to each other (represented by the symbol ⊕). The advantage is that the complexity of systems is treated from the early beginning of development, by decomposing complex requirements.

The concept of hierarchy also permits the reuse of requirements. In this case, a common requirement can be shared by other requirements. The hierarchy is built based on master and slave requirements. The slave is a requirement whose text property is a read-only copy of the text property of a master requirement. The master/slave relationship is indicated by the use of the copy keyword.

The derive relationship relates a derived requirement to its source requirement. During Requirements Engineering activities, new requirements are created from previous ones. Normally, the derived requirement is under a source requirement in the hierarchy. In a requirements diagram, the derive relationship is represented by the keyword deriveReqt.

The satisfy requirement describes how a model satisfies one or more requirements. It represents a dependency relationship between a requirement and a model element, such as other SysML diagrams, that represents that requirement. This relationship is represented by the keyword satisfy. One example is to associate a requirement to a SysML Block diagram.

The verify relationship defines how a test case can verify a requirement. This includes standard verification methods for inspection, analysis, demonstration or test. For example, given a requirement, the steps necessary for its verification can be summarized by a state-machine diagram. The keyword verify represents this relationship.

The refine relationship provides a capability to reduce ambiguity in a requirement by relating a SysML Requirement to another requirement.

---

**Fig. 1.** Approach for modeling user requirements with SysML.

**Fig. 2.** Basic SysML Requirements diagram.
model element. This relationship is typically used to refine a text-based requirement with a model. For example, how a Use Case can represent a requirement in a SysML Requirements diagram. The relationship is represented in the diagram by the keyword refine. The refinement is distinguished from a derive relationship in that a refine relationship can exist between a requirement and any other model element, whereas a derive relationship is only between requirements.

The trace relationship provides a general purpose relationship between a requirement and any other model element. Its semantics has no real constraints and is not as well-defined as the other relationships. For instance, a generic trace dependency can be used to emphasize that a pair of requirements are related in a different way not defined by other SysML relationships.

6.2. SysML Requirements table

Requirements traceability is an important quality factor in a system’s design. Basically, requirements traceability helps in identifying the origin, destination, and links between requirements and models created during system development.

Identifying and maintaining traces between requirements are considered important activities during Requirements Engineering (Gotel and Finkelstein, 1994; Sahraoui, 2005). The activity of requirements tracing is very useful, for example, to identify how requirements are affected by changes. For instance, in later development phases a requirement may be removed, and the related requirements may also be deleted or reallocated. Another case is when a requirement has changed and the stakeholders need to know how this change will affect other requirements. Traceability also helps to ensure that all requirements are fulfilled by the system and subsystem components. When requirements are not completely traced to the specific design elements, there is a tendency to lose focus as to the specific responsibility of each design model. This can lead to costly changes late in the life cycle and can also lead to incorrect or missing functionality in the delivered system. As a matter of fact, important decisions on requirements and the correspondent models are better justified when traceability is given proper attention (Ramesh and Jarke, 2001). One way to manage the requirements traceability in SysML is by using requirements tables.

SysML allows the representation of requirements, their properties and relationships in a tabular format. One proposed table shows the hierarchical tree of requirements from a master one. The fields proposed for Table 2 are the requirement’s ID, name and type. There is a table for each requirement that has child requirements related by the relationship hierarchy.

6.3. SysML Use Case diagram

The Use Case diagram shows system functionalities that are performed through the interaction of the system with its actors. The idea is to represent what the system will perform, not how. The diagrams are composed of actors, Use Cases and their relationships. Actors may correspond to users, other systems or any external entity to the system.

The SysML Use Case diagram is derived without important extensions from the UML Use Case diagram. The main difference is the wider focus, as the idea is to model complex systems that involve not only software, but also other systems, personnel, and hardware.

The detailed sequence of events in a use case can be represented in different manners. It is common to describe the sequence of events in structured language based on a pre-defined pattern, or by using Activity diagrams (Almendros-Jimenez and Iribarne, 2005), Sequence diagrams (Almendros-Jimenez and Iribarne, 2007), or Petri nets (Soares and Vrancken, 2008b). Within SysML, a Use Case may also be related to a SysML Requirements diagram. Which of these techniques to use depends on the intended reader and the development phase.

One important limitation of Use Cases diagrams is that their focus is on specifying only functional requirements. Non-functional requirements, such as performance, and external requirements, such as interfaces, which are fundamental in software-intensive systems, are not well-represented by Use Case diagrams.

7. Extensions to SysML Requirements diagram and tables

SysML is a highly customizable and extensible modeling language (OMG, 2008). Organizations that develop systems for several different domains may create a profile for each domain. Profiles may specialize language semantics, provide new graphical icons and domain-specific model libraries. When creating profiles, it is not allowed to change language semantics; normally profiles may only specialize and extend semantics and notations.

The basic SysML Requirements diagram is extended in this section. The purpose is to try to address the identified shortcomings presented in Table 1. The first extension is performed by creating stereotypes of stereotypes, in which case they are named sub-stereotypes (Section 7.2). Sub-stereotypes are similar to class inheritance in UML: they inherit any properties of their super-stereotypes, and add their own. These stereotypes are used to express the different types of user requirements proposed in Section 7.1. The second extension is to add properties besides the two default ones (Id and Text) (Section 7.3). The third extension is about grouping related requirements (Section 7.4). The last extension is to extend the SysML Table to provide requirements in a tabular format (Section 7.5).

7.1. User requirements classification

A common classification proposed for requirements in the literature is based on the level of abstraction, in which requirements are classified as functional or non-functional (Robertson and Robertson, 2006). Functional requirements describe the services that the system should provide, including the behavior of the system in particular situations. Non-functional requirements are related to emergent system properties such as safety, reliability and response time. These properties cannot be attributed to a single system component. Rather, they emerge as a result of integrating system components. Non-functional requirements are also considered quality requirements, and are fundamental to determine the success of a system.

A table of contents of a requirements specification with the following requirements items: external interfaces, functions, performance, logical database, design constraints, and software system attributes, is suggested in IEEE (1998). For sake of simplicity, and as some of the items can be considered non-functional requirements (performance, design constraints and software system attributes), or functional requirements (logical database), the second classification used in this article (after user vs. system requirements) is as follows (Soares and Vrancken, 2008a):
A risk is an uncertain event or condition that, if it occurs, has a positive or negative effect on a project’s objectives (PMI, 2008). The number of tagged values to be attached to each property, and also an additional properties are not mandatory and may appear in any stereotype. The extensions proposed attaches a tuple $R = \{P, I\}$ to each requirement, in which $P$ indicates the probability and $I$ is the impact of the effects of occurring that risk. The suggested values for $P$ are: very low, low, moderate, high and very high. The suggested values for $I$ are: insignificant, tolerable, serious, very serious or catastrophic. Numeric values can also be assigned, but may lead to confusion. The combination of both values can be used as input to strategies to manage project risks.

If the requirement is derived from another requirement, it is useful to know its source. The source property describes where the derived requirement originated. This information is important to trace requirements during system life cycle development.

One approach to prevent future problems, such as delays, is to create tables considering risk, probabilities of occurrence, priority and impact. Then, labels can be given to each requirement, as for instance, colors, which visually help managers to know more about the requirements. Better contingency plans can be created and also special attention given to critical requirements (for instance, improved testing, inspections or the use of formal methods and tools.).

According to the PMBOK (PMI, 2008), knowing which requirements have high priority is useful for risk analysis and during system development. Prioritizing requirements is giving an indication of the order in which requirements should be addressed. A review of requirements prioritization techniques can be found in (Greer, 2005). Some recommendations on how to prioritize requirements (or triage) can be found in (Davis, 2003). A well-performed prioritization provides better system release planning, based on balancing importance vs. effort. Ranking assignment is the simplest prioritization technique (Greer, 2005). Basically, it consists of dividing requirements into groups, giving to each requirement a label, such as (critical, standard, optional) or the MoSCoW labels. The number of groups may vary, but within a group, all requirements have the same priority.

At least the main stakeholder directly responsible for the requirement should be known. In case there is more than one responsible stakeholder, the choices are to write all of them, or just write the most important. This information is represented in the responsible property.

The requirements version is useful to show if the requirement was changed. This property is fundamental, as uncontrolled changes are a source of problems in Requirements Engineering. In addition to the version, the date of creation/change is added.

In order to improve the activity of tracing requirements to design models, a property that relates the specific requirement to models of the design is added. Identifying and maintaining traces between requirements and design are considered important activities in Requirements Engineering (Sahraoui, 2005).

The resulting SysML Requirement with the proposed extensions is depicted in Fig. 4.

7.4. Grouping requirements

By modeling requirements with SysML, system complexity is addressed from the early system design activities. Managing decomposition is a crucial task in order to deal with complexity. Requirements may be decomposed into atomic requirements, and may later even be related in the sense that together they are capable of delivering a whole feature, i.e., they are responsible for a well-defined subsystem.

SysML Requirements may be part of other SysML Requirements, as a hierarchy (Soares and Vrancken, 2008a). Related SysML Requirements can be grouped into a single SysML Requirements...
7.5. Extension to the SysML Table

Table 3 shows an example of requirements data expressed in a tabular format. The proposed table shows the requirement Id, the name of the requirement, to which requirement it is related (if any), the type of relationship and the requirement type. This allows an agile way to identify, prioritize and trace requirements. As a matter of fact, whenever a requirement is changed or deleted, the SysML Requirements Relationship Tables (SRRT) are useful to show that this can affect other requirements.

8. Case study: RTMS user requirements modeling with SysML

In this section, a modeling approach is applied to model the list of user requirements presented in Section 2.

From the Software Engineering point of view, the development and maintenance of RTMS is a challenge due to many factors. The most important ones are mentioned as follows. First, requirements are frequently changed. The major reason is that the area of road network control is still largely uncharted territory (Vrancken and Soares, 2010). Thus, algorithms, techniques and methods are frequently being developed by traffic engineers. In addition, policies for transportation are often being changed as well (Eurostat, 2006).

8.1. SysML Requirements diagrams

The associated SysML Requirements diagram for the list of user requirements is depicted in Fig. 6. For the sake of simplicity, not all properties are included.

<table>
<thead>
<tr>
<th>Table 4</th>
<th>Hierarchy requirements table—TM4.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Name</td>
</tr>
<tr>
<td>TM5</td>
<td>Region-wide traffic management</td>
</tr>
<tr>
<td>TM6</td>
<td>Traffic flow managed optimally</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5</th>
<th>Hierarchy requirements table—TM7.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Name</td>
</tr>
<tr>
<td>TM9</td>
<td>Simulation analytical models</td>
</tr>
<tr>
<td>TM12</td>
<td>Wide range tasks scenarios</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 6</th>
<th>Hierarchy requirements table—TM9.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Name</td>
</tr>
<tr>
<td>TM10</td>
<td>Access statistical data</td>
</tr>
<tr>
<td>TM11</td>
<td>Access transient data</td>
</tr>
</tbody>
</table>

8.2. SysML Requirements tables

Tables 4–6 show SysML Requirements tables expressing hierarchy for requirements TM4, TM7, and TM9. The other proposed type of table (SRRT), relating requirements and their relationships for each SysML Requirements diagram is presented in Table 7.

<table>
<thead>
<tr>
<th>Table 7</th>
<th>SysML Requirements relationship table for TM.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Name</td>
</tr>
<tr>
<td>TM7</td>
<td>Task/scenario frames</td>
</tr>
<tr>
<td>TM8</td>
<td>Gather/interpret info.</td>
</tr>
<tr>
<td>TM13</td>
<td>Object task/scenario</td>
</tr>
</tbody>
</table>
9. Discussion

The list of desirable properties of requirements specification, shown in Table 1, is used again, this time to evaluate the approach.
proposed in this article (see Table 8). In the list, SRDE stands for the extended version of the SysML Requirement diagram, and SRRT for an extended version of SysML Tables used for Requirements Engineering activities.

From the list, it is clear that the proposed user classification in Section 7.1 and the extensions in Section 7 fulfill almost all the properties identified in Table 1. The partially fulfilled properties, “Well-defined semantics” and “Solve ambiguity”, are not fulfilled even when the extended SysML Requirements diagram and SysML Tables are used. These properties are solvable by increasing formality for the modeling language, i.e., by using formal methods. However, when using formal methods, other properties, such as “human readable” may be lost.

In IEEE (1998), a list of characteristics that are expected for a software requirements document is given. To finalize the evaluation, how each of these characteristics are addressed by the approach presented in this article is briefly presented as follows:

**Correctness:** According to IEEE (1998), no technique can ensure correctness. However, the SysML Requirements diagram provides the possibility of relating requirements to other design models, facilitating that the user can determine if the SRS correctly reflects the actual needs.

**Unambiguity:** Ambiguity can be solved with the use of formal methods (Hinchey et al., 2008). The issue is that natural language is ambiguous, but unavoidable in the early phases of Requirements Engineering.

**Completeness:** The proposed types for requirements are well-described with the extensions proposed for the SysML Requirements diagram and tables. Thus, all types of requirements can be modeled.

**Consistency:** Conflicts between requirements can be discovered by explicitly describing their relationships, and the type of each relationship. In addition, by grouping related user requirements, conflicts within a group of requirements and between groups can be discovered.

**Ranked by importance:** Typically, not all requirements are equally important. The approach presented in this article fulfill this characteristic by adding two properties to the basic SysML Requirements diagram: Risk and Priority.

**Ranked by stability:** Stability can be expressed in terms of the number of expected/performed changes to any requirement. This is addressed in this article by controlling version and date of a requirement, through the additional property Version/Date.

**Verifiable:** As ambiguity is not solved with the application of SysML, this characteristic is not fully present. However, the advantage of using SysML is the possibility of relating SysML Requirements to formal design models that can be formally verified.

**Modifiable:** The requirements document is modifiable if its structure and style are such that any changes to requirements can be made completely, and consistently, while retaining the structure and style. Expressing each requirement separately is highly desirable. This characteristic is addressed in this article by modeling requirements using a well-defined SysML Requirements diagram, and by organizing the relationship between requirements.

**Traceable:** A requirement is traceable if its origin is clear and if it is possible to refer to it in future development. The solution proposed in this article is to create SysML Tables expressing the relationships between requirements and other design models.

The proposed approach presented in this article uses two SysML diagrams and SysML Tables. This is necessary because multiple aspects of user requirements modeling are covered, which is useful as multiple stakeholders are involved. Thus, the SysML Use Case provides systems’ view of functional requirements and actors, delimiting the system scope. Requirements relationships and properties are graphically represented using the SysML Requirements diagram, and SysML Tables gives a tabular format for requirements.

As UML in general and Use Cases in particular do not support goal-oriented modeling (Moody et al., 2010), future research will focus on comparing the proposed approach presented in this article with other techniques based on Goal-Oriented Modeling, such as i* (Yu, 1997) and KAOS (Dardenne, 1993). Both techniques support graphical modeling of requirements. The KAOS graphical notation is less complex and easier to use than i* and focus more on the late requirements phase (Quartel et al., 2009).

Goal-oriented modeling has been enthusiastically embraced by the Requirements Engineering research community but has so far had negligible impact on practice (Moody et al., 2010). As SysML is a UML profile, and UML is currently the de facto modeling language for software-intensive systems, SysML has already an advantage at least in terms of potential use.

UML and SysML present additional advantages over the graphical notation i*. Unlike i*, which lacks explicit design rationale for its graphical conventions, SysML is well-defined. SysML is conformant to an official metametamodel, the MOF (OMG, 2006), while i* semantic constructs and grammatical rules are defined using nat-

<table>
<thead>
<tr>
<th>List of requirements</th>
<th>SRDE</th>
<th>SRRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M) Graphical modeling</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(M) Human readable</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(M) Independent towards methodology</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(M) Relationship between requirements</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(M) Relationship requirements/design</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(M) Requirements risks</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(M) Identify types of requirements</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(M) Priority between requirements</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(M) Non-functional requirements</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(M) Grouping related requirements</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(M) Consistency</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(M) Modifiable</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(M) Ranking requirements by stability</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(S) Solve ambiguity</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(S) Well-defined semantics</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(S) Machine readable</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(S) Correctness</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(S) Completeness</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(S) Verifiable</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(S) Traceable</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>(S) Type of relationship requirements</td>
<td>●</td>
<td>○</td>
</tr>
</tbody>
</table>
ural language (Moody et al., 2010), which leads to problems of inconsistency, ambiguity, and incompleteness.

10. Conclusions

It is essential to have properly structured and controlled requirements specifications that are consistent and understandable by stakeholders. This is addressed in this article by presenting an approach to model and analyze a list of user requirements using the SysML Requirements diagram, the SysML Table, and the SysML Use Case diagram.

As usual in system development, changes in requirements are likely to happen, and using the SysML Requirements diagram is useful for developers to manage these changes. For instance, when a stakeholder asks for a change in one specific requirement, using the many relationship types that describe traceability between models helps to uncover possible impacts in other models. The relationships are also useful to aid in requirements prioritization in order to decide which requirements should be included in a certain system release. Another advantage of using the SysML Requirements diagram is to standardize the way of specifying requirements through a defined semantics. As a direct consequence, SysML allows the representation of requirements as model elements.

In this article, a classification of user requirements is proposed. Then, the SysML Requirements diagram is introduced and the requirements relationships are detailed. SysML Tables are useful to represent decomposition in a tabular form and to improve traceability, which is an important quality factor when designing software-intensive systems. The SysML Requirements diagram is extended with new stereotypes including the proposed classification, which distinguish requirements as functional, non-functional or external. Some properties not presented in the original SysML Requirements diagram are added in order to represent important requirements characteristics. These properties were chosen based on an extensive literature review.

Finally, requirements are important to determine the architecture. For instance, external requirements help in delimiting the system context in relation with its environment. When designing the architecture, at least part of the functional requirements should be known. In addition, the non-functional requirements that the architecture has to conform with, such as portability, performance, and other quality attributes (security, modifiability), should be made explicit. Domain architecture and software architecture are topics for future research.

References

Abran, A., Bourque, P., Dupuis, R., Moore, J.W., Tipp, LL (Eds.), 2004. Guide to the
Eurasip. 2006. Keep Europe Moving—Sustainable Mobility for our Continent Mid-
Eurasip. 2006. Keep Europe Moving—Sustainable Mobility for our Continent Mid-
Eurasip. 2006. Keep Europe Moving—Sustainable Mobility for our Continent Mid-
Eurasip. 2006. Keep Europe Moving—Sustainable Mobility for our Continent Mid-


Michel dos Santos Soares obtained a BSc degree in Computer Science from the Federal University of São Carlos, Brazil, in 2000, a MSc degree in Computer Science from the Federal University of Uberlândia, Brazil, in 2004, and a PhD in Software Engineering from the Delft University of Technology, in Delft, The Netherlands. Since 2010 he is an Assistant Professor at the Federal University of Uberlândia, Brazil. His research interests include Modeling and Analysis of Software-Intensive Systems, Software Architecture, Requirements Engineering, and Software Quality.

Jos Vrancken obtained a masters degree in Mathematics from the University of Utrecht, in 1982, and a PhD degree in Computer Science from the University of Amsterdam in 1991. Since 1991, he was employed by the Dutch government as a systems architect in the fields of road traffic and water management. Since 2003, he is an Assistant Professor at Delft University of Technology in the use of IT in the design, maintenance and exploitation of infrastructures, with emphasis on the road and the data communications infrastructures.

Alexander Verbraeck is a Full Professor of Systems and Simulation at the Faculty of Technology, Policy and Management of Delft University of Technology, The Netherlands. In addition, he is part-time Research Professor in Supply Chain Management at the R.H. Smith School of Business of the University of Maryland in the USA. He has a BS in Mathematics, and an MSc and PhD in Computer Science, all from Delft University of Technology. His research focuses on discrete-event simulation, serious gaming and training, logistics and transportation, and project management. He is a member of the Center for Project Management at TU Delft, and he is heavily involved in project management research and training for industry. As a co-director of TU Delft’s serious gaming institute, he researches novel applications of interactive simulation and virtual worlds. Many of his multi-year research projects have been funded by national programs and by the European Union. His applied research in simulation and gaming has been funded by many organizations. He presented over 100 refereed papers at conferences, wrote close to 20 book chapters, and published his work in a number of international journals.