A Domain-Specific Language to manage Requirements Traceability

Saida Haidrar1*, Adil Anwar1, Jean-Michel Bruel2, Ounsa Roudies1
1 E3S Research Center, Mohammed V. University in Rabat, Siweb, EMI, Morocco.
2 IRIT/CNRS, University of Toulouse, France.

* Corresponding author. Tel.: +212 669085263; email: saidahaidrar@gmail.com
Manuscript submitted January 10, 2018; accepted March 8, 2018.
doi: 10.17706/jsw.13.9.460-480

Abstract: Tracing requirements back to stakeholders and initial documents on one hand, and forward to their corresponding designed system artifacts, on the other hand, are crucial activities. Requirements are often expressed independently from those artifacts and can take several forms: texts or models. This makes hard and tedious the inference of trace links between requirements and artifacts. This paper introduces ReqDL, a domain-specific language for describing requirements and, at the same time, capturing bi-directional traceability data, which concerns more precisely system modeling elements. The paper also introduces a generation algorithm based on ReqDL specifications in order to automatically generate independent trace models. Indeed, we present ReqDL concrete and abstract syntax in terms of grammar and metamodel. Using ReqDL expressions, we aim at assisting the traceability in formalizing easily understandable requirements and establishing an initial relationship between requirements and other system artifacts. The main result is the generation of requirements traces models which incorporate explicit and implicit trace links between requirements and design elements. Moreover, we provide a working example through which we demonstrate ReqDL practicality and usefulness.

Key words: Requirements engineering, model-based engineering, systems engineering, traceability, complex systems, metamodel, domain-specific language, ReqDL.

1. Introduction

Requirements traceability reflects the knowledge about where and how the requirements have been taken into account in the system. This knowledge is very important for stakeholders within a system development project. It helps to handle the system complexity and its skills diversity by keeping track of requirements over system development artifacts, and up to their stakeholders and sources. Model-Based System Engineering (MBSE) proposes models to represent all artifacts handled by a system development process. The principle is to raise the abstraction level in order to better manage the system’s complexity. System Modeling Language (SysML) is dedicated to modeling systems [1]. Its requirements diagram is devoted to tackling requirements formalization and linkage. This diagram can describe links between requirements (composition, refinement, ...). It can also describe links between requirements and other modeling elements (blocks, activities, ...). However, this diagram does not enable users to manage traceability issues throughout the whole development process. For example, requirements cannot be traced back to their origins neither traced towards all their subsequent development artifacts and properties. Furthermore, a dedicated trace model is not supported. Such model could fit all cases where traces have to be used for many practical usages, such as consistency checking and change impact analysis. In the context
of Model-Based System Engineering, dedicated languages are of particular interest. A domain-specific language (DSL) provides a notation tailored towards an application domain and is based on the relevant concepts and features of that domain [2]. Our specific issue is requirements traceability management. Therefore, our idea is to deal with this by constructing a new dedicated language and applying metamodeling tools. Our proposed traceability solution stands for the main traceability phases which are trace creation, trace representation and trace generation. While expressing requirements, the existence of eventual dependencies between requirements and different artifacts might be detected. Trace capture is an important concern which consists in eliciting various data that would be involved in the traceability task. So that such data will trigger the creation of trace links.

Requirements specification and formalization has been extensively researched in the past years, much progress has been made in this issue. However, expressing requirements is still performed independently from the traceability task, which means that those expressions do not elicit the needed traceability data that would assist trace links generation. In addition, even if system engineering tends to formalize the practice of systems development through the use of models, the application of model-based system engineering tools (MBSE) for establishing a common interface between requirements and artifacts remains limited. This paper describes a requirements language, called ReqDL, that supports the explicit referencing to system artifacts in requirements expressions. We have designed the vocabulary of ReqDL so as to enable analysts to integrate previous data in requirements expressions. We propose a structural description of the requirement's functionality, which semantically captures the system element that performs or undergoes the action. Based on the Xtext framework [3], an abstract grammar and a static semantic are given to enhance the DSL understanding and usage. Hence, ReqDL addresses both requirements description and bi-directional traceability (i.e., backward and forward traceability). Another important concern on requirements traceability is the representation of traceability data. Indeed, the trace links must be semantically enriched in order to better understand these relationships and exploit them for many development aspects (analyzing the change impact, checking the consistency of requirements and system properties . . .). For this purpose, we propose to incorporate trace link concept into the ReqDL model. We afford also key features necessary to trace requirements, such as trace types, trace categories, supplier and client elements. The final contribution responds to the challenge of automating the traces generation task. We exploit captured trace data within ReqDL expressions so that we can produce a separate trace model, compliant with the defined meta-model. Indeed, we built trace links using a generation algorithm that manipulates ReqDL and SysML constructs. The remainder of this paper is structured as follows: in section 2, we introduce a background about requirements description approaches and list a set of needed properties for requirements languages. Section 3 gives an overview of ReqDL language. The ReqDL abstract syntax and static semantics are described in section 4. Section 5 describes ReqDL editor and gives a process for generating trace models. A Water distiller example is presented in section 6 as a proof of concept, followed by a review of related approaches and a discussion of our approach according to the given list of requirements (section 7 and 8). Finally, we give some conclusions and perspectives of this work.

2. Background

2.1. Requirements Description Approaches

There are several approaches for describing requirements. Basically, these approaches can be classified as purely textual, model-based, and formal approaches.

2.1.1. Text-based approaches

The most common approach is to write requirements using natural language (NL). This can be used for different purposes to describe any kind of requirements. However, the major issues of NL are imprecision,
misunderstandings, ambiguity, and inconsistency [4]. As an improvement of these concerns, structured natural language (SNL) [5] is used in order to give more structure to requirements documents. They are specifically oriented to algorithms and specific programming languages. For those structured languages, it is almost difficult to describe complex logical conditions. Nowadays, there exist a number of industrial solutions that are dedicated to managing textual requirements. IBM Rational DOORS is one of the leading requirements management system tools. It is based on natural language and provides features that are needed to capture, track and manage requirements. Similarly, Dassault Systems’ Reqtify permits to manage textual requirements in complex systems. It allows users to make relations between requirements expressed in several ways. These tools provide a way to define relationships between requirements as well as relationships between requirements and other artifacts. But, it does not provide a strong semantic either for requirements expressions or their trace links.

### 2.1.2. Model-based approaches

The SysML Requirements diagram helps in better organizing requirements, and also shows explicitly the various kinds of relationships between different requirements [6], and between requirements and other system artifacts. Another advantage of using this diagram is to standardize the way of specifying requirements. Thus, SysML allows the representation of requirements as model elements, which means that requirements are part of the system specification [7]. In addition to these diagrams, SysML tables can be used to represent requirements in a tabular format. However, traceability means provided by SysML are very restricted given that neither bi-directional traces are neither supported nor purpose-driven traceability, which might be an obstacle to trace links usage. Moreover, SysML meta-model does not provide the rich semantic we need to bind requirements to previous and subsequent development artifacts.

These tools make the link between requirements coming from different sources unlike SysML requirements diagram which limited the links to SysML constructs.

### 2.1.3. Formal approaches

Meyer has proposed in [8] an approach that combines specification and design components and design by contracts. Indeed, the author describes requirements using a unique language Eiffel that incorporates design elements into requirements expressions. This approach uses the object-oriented paradigm. Some works tend to propose a syntax to express requirements for system engineering. In [9], the authors present a structured natural language named RELAX, in a CAS (complex Adaptive System) context. Recently, the so-called FORM-L has been proposed in [10] in order to permit formal modeling of requirements. However, these two syntaxes are developed for a very specific range of systems.

We therefore think that each of these approaches taken independently does not make it possible to describe the requirements and their different dependencies. We then retain the interest of determining a set of properties as a framework for comparing the different requirements languages.

### 2.2. Desirable Properties for Requirements Specification

We start by giving a list of desirable properties for requirements description languages. According to our literature review on requirements specification approaches and tools [7], [11]-[14], we have identified the following non-exhaustive list of language requirements from three perspectives: the language ease of access and use, requirements coverage, and requirements specification characteristics.

**Languages ease of use:** If the requirement expressions are based on graphical modeling, it may facilitate the communication of models to stakeholders. All the involved stakeholders have to understand the model, that is why it is important to use human readable specification. The utilized languages for requirements

---

description should be as independent towards methodology as possible.

Requirements coverage: The requirement specification approach is evaluated according to the degree of managing the requirement. Thus, it is important to check whether the approach incorporates all Requirements types and Requirements levels.

Moreover, Requirements dependencies have to be highlighted while describing the requirement. Requirements priority should also be covered by the specification. In addition, we intend to explore which of the approaches afford a well-defined requirements model and manipulate requirements with the high level of abstraction using meta-models. Requirements characteristics: A number of necessary qualities for requirements expression (consistent, unambiguous, traceable, verifiable...) are proposed in [14]-[16], which are basically related to a good software requirements specification (SRS) document. We choose to discuss the following qualities according to requirements expressions in order to provide an easier set for analyzing the above approaches².

- Consistent: checks if the requirement specification agrees with other documents of the project such as system goals and models.
- Modifiable: checks if the structure and style, applied for expressing requirements, are such that any changes to the requirements can be made easily.
- Unambiguous: The requirement language should provide non-ambiguous constructions in order to produce non-ambiguous requirements.
- Correct: Every requirement stated is one that the software shall meet, the user can determine if the requirements specification reflects correctly his actual needs.
- Complete: checks if all significant requirements of every type are included. Thus, the approach should be able to specify all types of requirements.
- Traceable: the origin of each requirement is clear and it facilitates the referencing of each requirement in future development.

Table 1. Comparative Table of Requirements Specification Approaches

<table>
<thead>
<tr>
<th>List of desirable properties</th>
<th>NL</th>
<th>SysML</th>
<th>DOORS</th>
<th>ReqTify</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relax Form-L Graphical modeling</td>
<td></td>
<td></td>
<td></td>
<td>++ -</td>
</tr>
<tr>
<td>- -</td>
<td>Human readable</td>
<td>++</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>+ -</td>
<td>Independent</td>
<td>++</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>+ -</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Requirements dependencies
- - Requirements types ++ + + |
+ - - Requirements levels - - |
- - - Requirements model - - |
+ - - Abstraction level - - |
- + - - - - - - |

Consistent
++ + Modifiable |
+ + Unambiguous |
- - ++ Correct |
+ - - ++ Complete |
- - - ++ - Traceable |
- + + + - - |

³ https://eclipse.org/proposals/polarsys.reqcycle/
Table 1 highlights a mapping between the list of the properties discussed in this section, and the requirements languages discussed in section 2.1. The entries are classified as high support (++), medium support (+), and low support (-). Requirements coverage is partially addressed in SysML, or not addressed at all by some of the studied approaches [17], [10].

In particular, they do not describe requirements according to their definition level, and requirements dependencies and types are not very enriched. Except for SysML, none of the approaches did define models and meta-models to represent requirements related information. Moreover, we notice that even if formal solutions stand for some of requirements characteristics (i.e. Consistent, Unambiguous, and Correct), they do not provide a graphical tool and easily understandable concrete syntax. They are not so much human-readable and they are dependent on very topical domains and methodologies, with no support for traceability.

In this context, we propose a language dedicated to describing requirements and managing their traceability. This language is easy to handle and understand, and as close as possible to languages and tools used by engineers. The approach takes advantages of the MBSE techniques while focusing on trace extraction from requirements specifications. A discussion of ReqDL in light of the above properties is presented in Section 9.

3. ReqDL Overview

This section presents the ReqDL language, expanding and extending preliminary work [18] which introduced the basic concepts and elements of the language. ReqDL takes the form of model-based and structured language, including operators designed specifically to describe requirements and capture their trace links. These operators and their informal semantic are introduced in this section and the abstract syntax of the language is given in Section 4.

Fig. 1 shows that the ReqDL package is related to the SysML package via the import relationship, which means that our DSL affords the possibility to import SysML, constructs from both structural and behavioral models, with the purpose of tracing them to requirements. ReqDL package includes two views Requirement and Trace Link. There may exist a dependency relationship between them, which takes a cross-referencing form.
3.1. ReqDL Vocabulary

The vocabulary of ReqDL is designed in order to enable analysts to clearly describe requirements. A textual concrete syntax is used here to enable faster input/creation of ReqDL descriptions and advanced editing features such as text completion. Table 2 gives the set of ReqDL concepts, organized into attributes, modal and temporal and logical operators, action operators, and dependencies. This vocabulary aims at expressing requirements in a simple and easily understandable way in order to reduce language effects when documenting requirements. First of all, ReqDL addresses all kinds of requirements and documents information about the requirements as attributes. This includes, for example, the unique identifier of a requirement, its name and traceability status, the requirement definition context, the type, and the level of requirement definition. Indeed, further information can be added to a subset of requirements using the keyword include, such as the author and sources of the requirement. The description attribute of the requirement embodies the function or the action that requires being performed (see attributes section in Table 2). In addition, there are a number of operators intended to be used for describing the action included in the requirement. These are organized into auxiliary operators, temporal and logical, and operators of actions. Auxiliary operators determine the level of obligation of the requirement. The execution conditions of the action are included in the description, using the temporal and logical operators. Action operators are used for differentiating autonomous, interactive and passive actions. ReqDL is designed not only to specify the requirements but also to establish a binding between the requirements and different elements that come from the system development process. For this purpose, it is possible to display the model element in the description attribute of the requirement, which is related to the action using the keyword using (see operators sections in Table 2). Finally, the concrete syntax of the ReqDL language integrates a set of dependencies in order to capture explicit and direct links that exist between different artifacts. Different keywords are used in order to denote these requirements such as Satisfy, Verify, and Derive... In the following, we give the informal semantic of ReqDL, which highlights the meaning of the various operators and attributes related to ReqDL requirements and trace links.

3.2. Informal Semantic

The most important component in requirement expressions is the description attribute; it takes a syntactic structure of the action or functionality included in the requirement. In order to specify semantically requirement, we use the relevant concepts from table 2, and the instructions below (adapted from [12]):

Determine requirement obligation: the modal verbs shall, should, and will are used to determine respectively either the requirement is obligatory, urgent or recommended in the future.

Describe the action of the requirement regarding the system behavior: requirement action refers to the functionality specified by the requirement. It is described using a verb sentence that must be as clear as possible. We might semantically distinguish three kinds of requirement actions:

- Autonomous action: the system performs the action autonomously with no external interaction or event, this is directly expressed.
- Interactive action: the system provides the action as a service for the user, this is expressed using the structure: "Provide the user with the ability to..."
- Passive action: the system performs the action depending on an external event from a third part, this is expressed using the structure "Be able to ..."

Import the SysML model element who executes the action and insert it into description sentence after the keyword 'using': it aims at capturing the element that is associated with the action.

Add conditions: The operators "where" and "when" are used to determine under which conditions or
constraints the required functionality is performed.

Create a trace link between the requirements and its related system elements.

So, we presented several concepts used in ReqDL language in order to specify requirements and their dependencies. Thereafter, we give the abstract syntax of the language that defines the requirements textual vocabulary.

### Table 2. ReqDL Operators and Attributes

<table>
<thead>
<tr>
<th>ReqDL Operators and attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td></td>
</tr>
<tr>
<td>identifier</td>
<td>unique identifier of the requirement</td>
</tr>
<tr>
<td>name</td>
<td>name of the requirement</td>
</tr>
<tr>
<td>traceable</td>
<td>the Boolean status stating whether the requirement is traceable</td>
</tr>
<tr>
<td>level</td>
<td>determines the definition level of the requirement in a form of (Client requirement, System requirement, Component requirement)</td>
</tr>
<tr>
<td>type</td>
<td>determines the type of the requirement (General, Functional, NonFunctional, Physical, Design)</td>
</tr>
<tr>
<td>include</td>
<td>includes contexts and their attributes in order to associate extra information with the requirement, it may be the source or the author of the requirement</td>
</tr>
<tr>
<td>description</td>
<td>describes a syntactic structure of the requirement functionality using different operators</td>
</tr>
<tr>
<td>Modal Operators</td>
<td></td>
</tr>
<tr>
<td>shall, will or should</td>
<td>denote respectively legally obligatory, future, or urgently recommended requirements</td>
</tr>
<tr>
<td>Temporal and logical operators</td>
<td></td>
</tr>
<tr>
<td>when, where</td>
<td>requirement action is performed or provided under certain temporal or logical constraints</td>
</tr>
<tr>
<td>Action Operators</td>
<td></td>
</tr>
<tr>
<td>action ..</td>
<td>Autonomous action</td>
</tr>
<tr>
<td>provide ..with the ability to</td>
<td>Interactive action</td>
</tr>
<tr>
<td>be able to ..</td>
<td>Passive action</td>
</tr>
<tr>
<td>using</td>
<td>determines the element of the system allowing to perform, to refine, to satisfy, or to verify the action of the requirement</td>
</tr>
<tr>
<td>Trace Links</td>
<td></td>
</tr>
<tr>
<td>Derive</td>
<td>the requirement is derived from another one</td>
</tr>
<tr>
<td>Copy</td>
<td>the requirement is a copy of an existing one</td>
</tr>
<tr>
<td>Compose</td>
<td>the requirement is contained in another one</td>
</tr>
<tr>
<td>Refine</td>
<td>the requirement is a refinement of another one</td>
</tr>
<tr>
<td>OriginatesFrom</td>
<td>relates requirements to their sources</td>
</tr>
<tr>
<td>ResponsibleOf</td>
<td>relates requirement to the stakeholder who is in charge of it</td>
</tr>
<tr>
<td>Satisfy</td>
<td>relates requirement to the block that fulfills it</td>
</tr>
<tr>
<td>Verify</td>
<td>relates requirements to test cases</td>
</tr>
<tr>
<td>ImplementedBy</td>
<td>relates requirements to system fragments, implementation plans, code source, etc.</td>
</tr>
<tr>
<td>MappedTo</td>
<td>relates requirement to a particular attribute, operation, state or value of the artifact</td>
</tr>
</tbody>
</table>

### 4. ReqDL Abstract Syntax

#### 4.1. ReqDL Metamodel

Description: The devised language includes concepts often used to specify requirements. It has to be defined through formalism in order to express requirements models in a structured manner. Figure 2 gives an overview of the abstract ReqDL metamodel for describing requirements constructs, and model elements that are involved in traceability. Model is considered as a top-level element that may contain several model elements, which are imported from SysML, and requirements and trace links which are specific to ReqDL. A Requirement may have a Description field, may contain any number of related requirements, and may be connected to an arbitrary number of Contexts in order to assign further information to a subset of requirements. This extensibility mechanism is based on the definition of the relevant Context Attributes. As an example, through these attributes we can assign information about author and sources to client requirements. A Description may contain an ActionLog, and ModelElement. Model elements typically capture SysML components we want to trace to the requirement. TraceLink generalizes the relationship...
between requirements and the other elements handled by traceability capture. The trace link has two attributes (Category and Type) and relates two traceable elements i.e., the supplier and the client. We specify three categories of links according to three types of traceability [12]: Pre-specification requirements traceability (PreRS) means traceability links between the requirements and the previous artifacts that underlie them, post-specification requirements traceability (PostRS) refers to traceability links between requirements and artifacts belonging to subsequent development activities, and traceability between requirements (TraceBetReq) relates to dependencies between requirements. This metamodel is useful in terms of guiding practitioners to write requirements. For example, it is used to denote urgent, obligatory or future requirement by using the suitable modal verb. Next, we develop a set of constraints to be associated with ReqDL metamodel.

![Fig. 2. The ReqDL metamodel.](image)

Static semantic: The objective is to enhance static semantic of the ReqDL metamodel. We present thereafter some well-formed rules specifying the static semantics of the different trace links. We use the object constraint language (OCL) [19] to write these rules.

1) Requirements constraints:
A requirement cannot be an integral part of different requirements. A component requirement cannot be in a composition relationship with other requirements external to the composite requirement.

```
Context Requirement inv : self.outgoinglinks->select(t:TraceLink | toclIsTypeOf(Compose))->size() <= 1
```

Requirements can be reformulated many times by moving from a high-level of abstraction to a less abstract
The constraint lies in the fact that a requirement cannot be a copy of several abstract requirements.

\[\text{Context Requirement inv: self.outgoinglinks->select(t:TraceLink | t.oclIsTypeOf(Copy))->size() <= 1}\]

2) Trace links constraints:

A Derive link connects two requirements of different levels.

\[\text{Context TraceLink inv: self.oclIsTypeOf(Derive) implies self.client.oclIsTypeOf(Requirement) and self.supplier.level<> self.client.level}\]

A Satisfy link connects a requirement of level “ComponentRequirement” with “Block” or “Activity” model elements.

\[\text{Context TraceLink inv: self.oclIsTypeOf(Satisfy) implies (self.client.oclIsTypeOf(Activity) or self.client.oclIsTypeOf(Block)) and self.supplier.level = Level::ComponentRequirement}\]

A Verify link connects a “ComponentRequirement” or “SystemRequirement” level requirement to a test case.

\[\text{Context TraceLink inv: self.oclIsTypeOf(Verify) implies self.client.oclIsTypeOf(TestCase) and self.supplier.level = Level::SystemRequirement}\]

A Refine link can connect a requirement to another requirement, or a system requirement and the “UseCase” element.

\[\text{Context TraceLink inv: self.oclIsTypeOf(Refine) implies (self.client.oclIsTypeOf(Requirement) or self.client.oclIsTypeOf(UseCase)) and self.supplier.level = Level::SystemRequirement}\]

We have therefore proposed a meta-model to define the constructs of the new language ReqDL, as well as the properties describing the static semantics of this meta-model. In the next section, we will explain the Xtext-based grammar, designed to define the concrete syntax of the ReqDL language.

4.2. ReqDL Grammar

The ReqDL metamodel, in Fig. 2, is specified using the Ecore Implementation of the EMOF standard, provided by the Eclipse Modeling Framework (EMF). This metamodel is useful in terms of guiding practitioners to write requirements. For example, it is used to denote urgent, obligatory or future requirement by using the suitable modal verb. Listing 1 specifies the corresponding grammar describing requirements and model elements, and captures some traceability information. It is built using Xtext framework that is dedicated to describing an EBNF grammar for textual DSLs. Below, a detailed definition of ReqDL grammar is given, which uses Xtext to provide a textual concrete syntax, and includes a set of constructs. Each construct actually covers one concept from the ReqDL metamodel. We choose to discuss this grammar according to the main rules.

Each Xtext language description starts with a language declaration section [xte13]. The name of the language is given, and another existing grammar is declared to be reused (lines 1-2), this supports the grammar mixing feature. Then, three existing packages are imported using their namespace URIs, which are Ecore, UML, and SysML (lines 3-5). In fact, Xtext provides the ability to call external languages from the DSL grammar. We choose to refer to some UML and SysML constructs for capturing traceability data. The use of these references is shown through the description rule. The first rule in the Xtext grammar is always used as the entry or the start rule. The rule Model describes that the model will contain an arbitrary number (*) of Requirements and model elements, and includes contexts and model imports. The rule Requirement starts with an optional keyword Traceable (line 18), followed by an optional level description (line 19), and by a mandatory name and an arbitrary number (*) of contexts (line 20). A requirement has a set of attributes: identifier, type, description (lines 21-22). Possible values of Type are enumerated in line 23. We intent to structure the description of the requirement. It may contain temporal and logical conditions starting with the keyword where (line 26). Description includes subject, ObligationVerb, and Action (lines 27). The action can be of three types: autonomous, interactive, and passive (lines 30-33). It may contain a ModelElement related to the action description with the keyword using (line 28). These model elements are described using references to UML and SysML elements through EPackages imports, they might be activities, blocks, testcases and usecases (lines 34-
One of the main goals of ReqDL specifications is to define a direct trace link that may associate a requirement with one of the model elements. So, a TraceLink includes the following mandatory elements: Supplier, Client, TraceType, and TraceCategory (lines 42-51).

Listing 1: Xtext-based grammar for ReqDL language.

5. ReqDL Editor and Trace generation

The goal of the generation process is to depict implicit links that cannot be covered using ReqDL specification. Firstly, the core idea of creating ReqDL links between various artifacts is letting the analysts create these explicit links themselves. First, the requirement engineer describes a set of requirements using ReqDL operators and creates all possible links between these requirements. He can also associate each requirement with its definition context. Then, he captures the SysML model element that is in charge of fulfilling the specified action included in each requirement, especially in the case of component requirements. Finally, he creates manually all captured direct links of different categories respecting ReqDL syntax. Secondly, the ReqDL links are used to infer implicit links between requirements and system design, specifically between client requirements, blocks, activities, and properties. The generator is devised to automatically deduct these links using the algorithm 1, which is executed when the status of ReqDL requirements is assigned to Traceable. For example, a link can be generated between a client requirement and a specific activity or property, passing through all related sub-requirements and blocks. As a result, downstream traceability is addressed with a tiny level of granularity by generating independent trace models. In figure 3, we give an overview of the adopted generation scenario in order to keep track of client requirements. It takes ReqDL and SysML models as inputs and generates trace models, including ReqDL trace links and new inferred implicit links.

5.1. ReqDL Editor and Platform

Platform We use Xtext\(^5\) to implement the new DSL, ReqDL. Xtext is an Eclipse-based framework for programming language development and for DSLs \(^3\) Xtext provides a set of API libraries to describe the different aspects of a specific language. It is used to describe an EBNF grammar, in conjunction with Ecore implementations. This technology is used to automatically generate an ANTLR parser as well as an Eclipse

\(^5\) https://eclipse.org/Xtext/
\(^6\) http://www.eclipse.org/modeling/emf/
plugin that forms a complete code editor. Xtext not only facilitates building a DSL, it can support the construction of an Eclipse-based editor for the language. Xtext covers most aspects of language features including language infrastructure, parser, and compiler or interpreter. Xtext also allows customizing these features according to individual needs. Xtext uses EMF models to create in-memory object graphs while processing text. The in-memory object graph is called Abstract Syntax Tree (AST) [3]. It provides an implementation of Ecore resources that encapsulates the parser that converts text to an EMF model [3]. EMF is a platform that provides essential infrastructure for implementing task-specific model management languages. The ReqDL metamodel (Fig. 2) is specified using the Ecore Implementation of the EMoF standard, provided by the Eclipse Modeling Framework (EMF) [6].

**ReqDL Editor** We produced an Xtext-based editor for ReqDL in the Eclipse environment (Figure 4). The ReqDL models would have the extension .reqdl. The editor comes with syntax highlight (with Eclipse preferences for font, color, and style for comments, keywords, numbers, and Strings), predefined and selectable templates, a tree-like Outline view (on the right), and content assistance (for code/bracket completion). Default error handling and quick fixes are also available (underlining, error symbol on the far left side, and error details in the Problems view).

As an illustration, we have expressed two requirements from the vehicle specification [6]. They are from different definition levels as depicted in figure 4. We specify the client requirement "Performance" which includes the system requirement "Acceleration", and the system requirement "EnginePower" which derives from "Acceleration". There are described using model verbs, and autonomous actions to specify the related functionality. Note that the two requirements are specified to be tracked through their posterior and previous artifacts using the keyword "Traceable". Also, we notice that SysML elements are imported through the description of each system requirement and using the structure "SysMLmodel.elementName". All these features are illustrated in Fig. 4.

Thus, ReqDL is useful for determining which requirements are susceptible to tracking and those that do not need to be tracked by using "Traceable" attribute. Furthermore, this DSL is effective in eliciting traceability information through requirements expression. In fact, we opted for a syntactic analysis of the description attribute of the requirement in order to find out a primary binding between requirements and model elements. Also, ReqDL shows explicitly the source and stakeholder that are the basis of the requirement. We can notice that ReqDL is applicable for a wide range of requirements, which covers many types (i.e. General, functional, design, performance, security) and many definition levels of requirements (i.e., Client level, System level, and Component level).

![Fig. 3. Trace links generation process.](image)

ReqDL validation it is possible to add custom error handling that statically validates domain-specific constraints. This is usually done by doing static analysis, and Xtext provides dedicated hooks for custom validation rules. We use an Xtend1 class generated by Xtext in the ReqDL package. This class is the default template for custom domain-specific error handling through the use of checks annotations [3]. For the purpose of ReqDL validation, a set of validation methods was added to validate several aspects. For instance, we integrate a method that verifies whether there exists unique dependency between requirements. If there
exists more than one trace link between two requirements, an appropriate validation error is displayed as shown in Fig. 5.

Fig. 4. Xtext-based ReqDL editor with features.

Fig. 5. Custom editor feedback on error.

5.2. Traces Generation Algorithm

We devise an algorithm for automatically generate trace links using ReqDL specification. It is based on collecting system elements to be linked with these requirements. In listing 1, the input is a set of client requirements R that are specified using ReqDL and a set of model elements imported from SysML. The algorithm iterates over client requirements r included in R. Based on ReqDL trace data, it extracts a set reqs which contains all requirements that should be connected to the client requirement r (lines 3-6), and traces between requirements are thus detected. It iterates over these requirements to find all their related blocks, activities, and testcases. When it comes to a component requirement, related attributes and operations of its related blocks are detected, which are in charge of fulfilling r. As a result, the algorithm returns three set (At, Pt, Rt) and the client requirement r is traced.

Algorithm 1: Post-traces capture

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Input: ( R = \sum_{r \in R} i_r ) and ( B = \sum_{r \in R} o_r )</td>
</tr>
<tr>
<td>2</td>
<td>Output: ( T = \sum_{r \in R} a_r )</td>
</tr>
<tr>
<td>3</td>
<td>( a_r ) is a set of traceable elements to be linked with ( r ).</td>
</tr>
<tr>
<td>4</td>
<td>( o_r ) is a set of properties to be linked with ( r ).</td>
</tr>
<tr>
<td>5</td>
<td>( T ) is a set of traceable elements connected to ( R ) via ReqDL and SysML diagrams.</td>
</tr>
<tr>
<td>6</td>
<td>( o_r ) is the set of operations imported from SysML.</td>
</tr>
<tr>
<td>7</td>
<td>If ( \text{level} ) is a Component Requirement then</td>
</tr>
<tr>
<td>8</td>
<td>( T ) = \text{getAssociated}(r);</td>
</tr>
<tr>
<td>9</td>
<td>( o_r ) = \text{getAssociated}(r);</td>
</tr>
<tr>
<td>10</td>
<td>( \text{for each} a_i \in A_i ), ( \text{for each} b_i \in B_i ), ( \text{for each} c_i \in C_i ), ( \text{for each} d_i \in D_i ), ( \text{for each} e_i \in E_i ), ( \text{for each} f_i \in F_i )</td>
</tr>
<tr>
<td>11</td>
<td>( \text{end for} )</td>
</tr>
<tr>
<td>12</td>
<td>( \text{for each} p \in P_i )</td>
</tr>
<tr>
<td>13</td>
<td>( \text{add} \text{to}\text{set}\text{membership}(r, p) )</td>
</tr>
<tr>
<td>14</td>
<td>( \text{end for} )</td>
</tr>
<tr>
<td>15</td>
<td>( \text{end for} )</td>
</tr>
<tr>
<td>16</td>
<td>( \text{end for} )</td>
</tr>
<tr>
<td>17</td>
<td>( \text{end for} )</td>
</tr>
<tr>
<td>18</td>
<td>( \text{return} (A_t, P_t, R_t) )</td>
</tr>
</tbody>
</table>
6. Application Example

The application example is based on a detailed specification of Water Distiller System (WDS), developed by No Magic team, which includes a great number of high-level and specified requirements. We choose to use this example as an application because it is close to a real case. They have already planned a requirements diagram and other design diagrams. However, we found that the mechanism of traceability was not implemented for the specified requirements. That is why we decide to apply our traceability approach upon the distiller system. We presume that the client, to whom the distiller is destined, is a humanitarian organization dedicated to the purpose of providing safe drinking water to the broadest possible spectrum of people. We use the acronym WDS to designate Water Distiller System.

6.1. Requirements Specification using ReqDL

In this section, we present a description of requirements using ReqDL. In case of the distiller, we start with specifying a general client requirement ClR0, which is decomposed into two client requirements ClR01 and ClR02. We use the acronym ClR to designate a client requirement. They are textually described as follows:

ClR0: A humanitarian organization will provide safe drinking water to people.

ClR01: The sources of unclear water may be still or flowing. In some cases, there will not be sufficient elevation to gravity feed water to the distiller.

ClR02: Purified water needs to be distributed.

Listing 2: Extract of the WDS client’s requirements description using ReqDL

Listing 2 specifies those client requirements using ReqDL expressions. We use autonomous actions to describe the functionality of each requirement, preceded by a modal verb. The general requirement ClR0 is classified as a future requirement since it includes the modal verb “will”. ClR0 is also related to its context that includes the source and the author attributes. Moreover, the client requirement ClR0 integrates a context that depicts the requirement origin defining its source and author as attributes. As ReqDL allows describing requirements according to different definition levels, we are going to re-express the above client requirements in interaction with the entire system. We use the acronym SR to designate a system requirement. We come up with a set of system requirements which are:

SR1: The distiller shall purify water so that it is safe to drink.

SR2: The distiller should be able to accommodate an external source of unclean water.

SR3: The distiller shall output purified water to the distribution system.

SR4: The distiller shall incorporate a mechanism to remove residue that results from the distillation.
We use the keyword "System" to denote system-level requirements. Listing 3 shows that the system requirement SR1 is a copy of the client requirement CIR0 since it describes the same needed functionality that is water purification. The system requirements SR2 and SR3 are respectively derived from CIR01 and CIR02. SR2 includes a passive action using the keyword "be able to"

We aim at addressing downstream traceability with a tiny level of granularity by means of ReqDL expressions. That is why we describe WDS component requirements, that can be directly traced to a single or a small set of blocks contributing to the satisfaction of that requirement. We use the acronym CpR to designate a component requirement. 

CpR1: The boiler shall 'boil the water to sterilize it;
CpR2: The heat exchanger shall cool the distillate so that it may be safely distributed;
CpR3: The valve shall remove the residue.

<table>
<thead>
<tr>
<th>Traceable System Requirement Distillation</th>
</tr>
</thead>
<tbody>
<tr>
<td>{identifier: 01}</td>
</tr>
<tr>
<td>type: Functional</td>
</tr>
<tr>
<td>description : The Distiller shall 'purify_water'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Traceable System Requirement InputWater</th>
</tr>
</thead>
<tbody>
<tr>
<td>{identifier: 02}</td>
</tr>
<tr>
<td>type: Functional</td>
</tr>
<tr>
<td>description : The Distiller should be able to 'accommodate_unclean_water'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Traceable System Requirement OutputWater</th>
</tr>
</thead>
<tbody>
<tr>
<td>{identifier: 03}</td>
</tr>
<tr>
<td>type: Functional</td>
</tr>
<tr>
<td>description : The Distiller shall 'output_purified_water_to_the_distribution_system'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Traceable System Requirement ResidueRemoval</th>
</tr>
</thead>
<tbody>
<tr>
<td>{identifier: 04}</td>
</tr>
<tr>
<td>type: Functional</td>
</tr>
<tr>
<td>description : The Distiller shall 'incorporate_a_mechanism_to_remove_residue'</td>
</tr>
</tbody>
</table>

Listing 3: Extract of the WDS system requirements description usingReqDL

<table>
<thead>
<tr>
<th>Traceable Component Requirement Boiling</th>
</tr>
</thead>
<tbody>
<tr>
<td>{identifier: 001}</td>
</tr>
<tr>
<td>type: Functional</td>
</tr>
<tr>
<td>description : The Distiller shall 'boil_the_water_to_sterlize_it' using Boiler</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Traceable Component Requirement Cooling</th>
</tr>
</thead>
<tbody>
<tr>
<td>{identifier: 002}</td>
</tr>
<tr>
<td>type: Functional</td>
</tr>
<tr>
<td>description : The Distiller should 'cool_the_distillate_to_be_safely_distributed' using HeatExchanger</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Traceable Component Requirement ResidueRemoval include</th>
</tr>
</thead>
<tbody>
<tr>
<td>{identifier: 003}</td>
</tr>
<tr>
<td>type: Functional</td>
</tr>
<tr>
<td>description : The Distiller should 'remove_resulted_residue' using Valve</td>
</tr>
</tbody>
</table>

Listing 4: Extract of the WDS components requirements description usingReqDL

6.2. System Definition and Modeling

We describe the structure of the distiller system using SysML Block Definition Diagram (BDD), and Internal Block Diagram (IBD). The distiller hierarchical decomposition is depicted through the BDD while the communication interfaces between the system blocks are represented by the IBD. In figure 6.a), the distiller is decomposed into its constituent blocks, which are: Heat exchanger, boiler, valve, and user interface and controller. HeatExchanger has two functionalities, which are heating dirty water and condensing the steam. The Boiler is in charge of boiling water and the valve is supposed to eliminate the residue.

More specifically, we construct the IBD (figure 6.b)) of the distiller in order to represent a set of architectural connectors between the identified blocks in the BDD with specific communication interfaces and ports. Dirty water is poured into the HeatExchanger to heat it, and then it is transmitted to the evaporator in order to boil it. Then the produced steam is cooled using the condenser and thus the purified
water is obtained. Activity diagrams consist of several activity partitions that capture the parallel behaviors of their corresponding blocks (figure 7). The main distiller activities are Heat Water and Boil Water and Condense Steam.

We use ReqDL in order to incorporate the set of needed system elements into the requirements specification. So, the SysML model is imported and its constructs are represented using the keyword "ModelElement" and a reference to SysML model (lines 1-6).

SysML provides allocation mechanism that consists in specifying a set of related items in a structured environment [6]. Creating allocations in the model helps to maintain consistency between system elements, especially between behavioral models and structural models. We choose to reuse the allocation relationships through our DSL in order to associate imported SysML blocks with their related model elements such as activities.

Listing 5: Extract of the WDS model elements description using ReqDL

```plaintext
1 import <WDS.uml> ;
2 ModelElement Block Distiller referTo SysMLmodel.Distiller
3 ModelElement Block Boiler referTo SysMLmodel.Boiler
4 ModelElement Block Valve referTo SysMLmodel.Valve
5 ModelElement Block HeatExchanger referTo SysMLmodel.HeatExchanger
6 ModelElement Activity HeatWater referTo SysMLmodel.HeatWater
7 ModelElement Activity BoilerWater referTo SysMLmodel.BoilerWater
8 ModelElement Activity CondenseSteam referTo SysMLmodel.CondenseSteam
9 ModelElement Activity HeatExchanger referTo SysMLmodel.HeatExchanger
10 TraceLink HeatWater Allocate HeatExchanger
11 TraceLink BoilerWater Allocate Boiler
12 TraceLink CondenseSteam Allocate HeatExchanger
13 TraceLink DrainResidue Allocate Valve
```
6.3. Trace models generation

In this section, explicit trace links will be created based on the ReqDL grammar and static semantic. In listing 5 and 6, many existing dependencies are captured across requirements expressions using ReqDL. These dependencies specify direct relationships between requirements (i.e. composition and refinement and derivation). For example, a composition dependency exists between the client requirement WaterSource and the client requirement WaterPurification. Moreover, some of the requirement descriptions incorporate the name of the model element which is in charge of fulfilling the required action. For instance, the component requirement Boiling is described using an action that is fulfilled by the block Boiler. As a result, a direct trace link is captured between the block Boiler and the requirement Boiling. Thus, ReqDL gives the possibility to create direct links between requirements and model elements. These model elements are extracted from SysML diagrams since ReqDL is able to import SysML constructs (i.e. UseCase, Block, Activity, and Testcase). In the listing below, we give ReqDL expressions that depict various traceability links according to the distiller requirements

```
1 TraceLink WaterDistribution Compose WaterPurification
2 TraceLink WaterSource Compose WaterPurification
3 TraceLink Boiling Derive Distillation
4 TraceLink Cooling Derive Distillation
5 TraceLink Cooling Derive OutputWater
6 TraceLink ResidueRemoval Derive Distillation
7 TraceLink ResidueRemoval Derive OutputWater
8 TraceLink Cooling Derive OutputWater
9 TraceLink Distillation Copy WaterPurification
10 TraceLink Boiler Satisfy Boiling
11 TraceLink Distiller Satisfy Distillation
12 TraceLink HeatExchanger Satisfy Cooling
13 TraceLink Valve Satisfy ResidueRemoval
14 TraceLink WaterQualityTest Verify WaterPurification
```

Equation 6: Extract of the WDS trace links description using ReqDL

At this stage, we are invited to apply the generation algorithm specified in algorithm 1 which allows generating the WDS trace model, including both existing ReqDL links and new inferred links between high-level requirements and system design artifacts. For example, implicit satisfaction links are created between the client requirement ClR0 and the activities Boil water, Condense steam and Drain residue. These implicit links are depicted in figure 8 in a form of red ellipses. After generating all the trace links between client requirements and system design, we deal with enhancing human-friendly representation of these trace links. To this end, we propose to construct graphic-based models to depict the created trace links. We use DOT language, which draws directed graphs as hierarchies. It runs as a command line program, web visualization service, or with a compatible graphical interface [20]. We use Graphviz [21] as an open-source tool for drawing the trace model which is specified in DOT language script. The visualization mechanism is based on a model-to-model transformation, which turns ReqDL and trace model constructs into a DOT-based graph.

As a result, the client requirement ClR0 is effectively traced back to their source and to stakeholders that propose them Client and RSD. These PreRS links are useful for checking the consistency of the requirement and if it responds to a client expectation and a system goal, specified in the specification document (RSD). Furthermore, dependencies between requirements are depicted through the trace model. As an example, the derivation link between ClR0 and SR2 and the copy link between ClR0 and SR1. Concerning components requirements CpR1 CpR2 and CpR, the trace model shows up several post-trace links, such as satisfaction link that relates CpR1 to the block boiler. These links allow checking the usefulness of system components and properties. All black links are manually captured using our DSL ReqDL. Red links are automatically
generated using the generation algorithm and associate high-level requirements such as CIR0 with design components such as blocks (Boiler and HeatExchanger) and activities (Boilwater and Drainresidue), these links are useful for detecting the impact of changing requirements on all their related requirements and model elements.

Fig. 8. The WDS traces model

7. Related Works

The need of keeping track of requirements within system development process led us to devise a domain-specific language for capturing trace links between requirements and SysML constructs. So, traceability data is no longer misty since it is recorded in a separate model with a human-friendly representation. Recently, much research in traceability has focused on suggesting model-based approaches for mapping artifacts of system development process. Mader and Gotel in [22] propose a semi-automatic approach for maintaining traceability links between requirements and UML artifacts, which is based on capturing flows of events in a prototype modeling tool. In fact, the authors deal with only event-based artifacts of UML, whereas our approach tackles a variety of structural and behavioral artifacts and provides a way for binding those artifacts to client requirements. Paige et al. in [23] have also studied the traceability issue from a model-driven perspective. They propose a means for identifying trace links in an MDE process through a traceability elicitation process, and they provide a way to define and implement such trace links. Similarly, Mirakhorli et al. in [24] are working on automating the traceability of quality concerns through a tactic-based approach. This is a very topical issue Comparing with our approach that integrates the ability to configure the generated trace models regarding many development purposes, e.g. system quality and change management. There already exist DSLs targeting traceability, Rahman and Amyot in [25], for instance, devise a DSL for capturing model language artifacts that might be linked to requirements within requirement management system (RMS). Their aim is similar to ours except that they enhance traceability only for software development artifacts and they utilize existing RMS for requirement and trace management. Thus, traceability tasks (i.e. trace capture, trace representation, and trace generation) are not
totally handled, and the approach does not produce trace links and models by itself. This work could be
taken as a complement to our work since they automatically capture model artifacts to be connected to
requirements. Graf et al. in [26] present an Eclipse-based DSL of ReqIF, the Requirement Interchange
Format. ReqIF is an emerging standard for requirements interchange, which is driven by the German
automotive industry. Their solution is a mapping table with three element types: source element, target
element, and additional information such as descriptions. The elements are identified by a data structure
called tracepoint, whose inner structure depends on the metamodel being used. Graf et al. describe that
their technical solution is often based on the import of requirements into the target models of other tools.
They see this as an important drawback as not all models support extensions. Many works have been done
in the requirements traceability field in connection with SysML specifications. Soares and Al. in [7] propose
to enhance traceability concept in requirements expression. They provide extensions to the SysML
requirements diagram and SysML requirements tables. A classification of user requirements and traces is
proposed and thus enhances checking requirements and trace consistency. However, in this work, the
proposed solution still requires the human expertise for generating trace links. Since the SysML extension
affects only requirements and their tables and does not extend traces, the granularity and bidirectional
traceability are not supported. Unlike our approach, this work focuses only on the representation of
requirements and trace tables by means of SysML extensions. The aforementioned approaches are related
to the issue of requirements model-based traceability. However, most of them do not take into account all
the above desirable properties. Some of the proposed solutions are specific to predefined purposes and
artifacts. In this context, our approach delivers a dedicated language for requirements traceability
management, which focuses on structuring requirements using rich semantic in order to capture eventual
trace links.

8. Discussion

The trace model construction is based on the captured information and a graphical representation of
ReqDL and SysML design models in a form of reusable trace models depicting all their binding links. We
propose to use ReqDL for specifying requirements and building a primary crossing between requirements
and artifacts. ReqDL is also effective in detecting traceable requirements and dependencies between them,
which permit to checking requirement consistency and dropping unneeded ones. This specific language
gives the possibility to infer the model element in charge of fulfilling the requirement based on a syntax
analysis of its action. Below, the list of desirable criteria of requirements modeling is reused to evaluate the
domain-specific language ReqDL.

Language ease of use: ReqDL affords structured and textual requirements specification which is easily
understandable and both human and machine readable. The requirements specification is done
independently from any applied methodology. Up to now, we specify text-based requirements models and
we generate graphical trace models. As a perspective, we consider devising a graphical concrete syntax of
ReqDL specifications.

Requirements Coverage: The approach fulfills a high level of requirements coverage. ReqDL handles all
types of requirements and specifies trace links between any types of requirements to any entity of the
system. We are not dealing with only topical or specific requirements.

Consistent: Consistency of requirements can be discovered by explicitly describing their relationships,
and the type of each relationship using our DSL. In addition, the added property "Priority" to
"TraceableRequirement" allows distinguishing between critical requirements and optional ones, which
permits to avoid requirements conflicts.

Traceable: The main result that satisfies this issue is the three given categories of traces (Pre-RS, Post-RS,
and TracebetReq) and their various types. The origin of the requirement is clear and it is possible to refer to the requirement for future development. The proposed traceability approach doesn't totally automate the generation task because the trace capture still requires human efforts. However, the algorithm allows inferring the trace type and category and creates effectively the trace. Therefore, we can say that the solution is semi-automatic.

Modifiable: ReqDL model is modifiable since its structure and style are such any changes to requirements can be made completely, and consistently. Separate requirements expressions are highly afforded within ReqDL models.

Unambiguous: Ambiguity can be solved with the use of formal methods [27]. Even though we choose to specify requirements with a textual concrete syntax which is unavoidable in the early phases of requirements engineering, the use of a domain-specific language and the meta-modeling tool allows overcoming partially natural language unambiguity. Xtext-based DSL allows the user to recognize common sources of ambiguity and fix these situations when they arise.

Correct: According to [27], no technique can ensure full requirements correctness. However, the ReqDL specification has the ability to relate to other SysML design models, verifying requirements are correctly

Complete: using ReqDL, requirements can be modeled regardless of their types and abstraction degree. The proposed types of requirements are well described with the assignment of their definition level.

9. Conclusions and Perspectives

This paper presented a new requirements specification language called ReqDL, designed to explicitly address requirements traceability issue. ReqDL has three types of operators to describe requirements and capture trace data: attributes, description, model elements, and trace links. We introduced a process for generating requirements trace models that builds up a view of the implementation of the requirements. Several perspectives on our work are under consideration. We are enhancing the visualization of the generated trace model in a human-friendly representation using graphs. Besides, we intend to work on a tool support to generate the trace model according to the user's requirements. In the future, we would focus on some empirical and formal validation of our approach by applying our approach and algorithm to different case studies. Moreover, we intend to develop additional ReqDL functionalities in order to manage inconsistencies between requirements.

References


**Saida Haidrar** is a Ph.D. student and a member of the Siweb research team of Mohammadia School of engineers. In 2013, she received an engineering degree in Computer Science at INPT in Rabat. She is interested in traceability management in Model-Based Systems Engineering.
Adil Anwar is currently an associate professor in computer science at the university of Mohammed-V in Rabat and as a member of the Siweb research team of Mohammadia School of engineers. In 2009, he received a Ph.D. degree in Computer Science at the University of Toulouse. He is interested in software engineering, including model-driven software engineering, mainly by heterogeneous software language, traceability management in Model-Based Systems engineering, combining formal and semi-formals methods in software and system's development.

Jean-Michel Bruel is head of the SM@RT team of the IRIT CNRS laboratory. His research areas include the development of software intensive Cyber-Physical Systems, methods/model/language integration, with the main focus on requirements and Model-Based Systems Engineering. He has defended his "Habilitation à Diriger des Recherches" in December 2006 and obtained in 2008 a full professor position at the University of Toulouse. He has been head of the Computer Science department of the Technical Institute of Blagnac from 2009 to 2012.

Ounsa Roudies is a full Professor since 1990 at the Computer science Department of Mohammadia School of engineers (EMI), Mohammed V University in Rabat, Morocco. She is head of Siweb research team (Information System and Web). In particular, Prof. Roudies is interested in Reuse and requirements engineering issues.