Grant Agreement nº	732463
Project Acronym:	OpenReq
Project Title:	Intelligent Recommendation Decision Technologies for Community-Driven Requirements Engineering
Call identifier:	H2020-ICT-2016-1
Instrument:	RIA (Research and Innovation Action)
Topic	ICT-10-16 Software Technologies
Start date of project	January 1st, 2017
Duration	36 months

D5.1 OpenReq Approach for Requirements Knowledge and Dependency Management

Lead contractor:	UH
Author(s):	UH, UPC
Submission date:	December 2017
Dissemination level:	PU

Project co-funded by the European Commission under the H2020 Programme.

Abstract: A brief summary of the purpose and content of the deliverable.

OpenReq is a project that aims to enhance requirements engineering activities. This document describes the state of the art and practice about, and the technical approach for the knowledge representation and dependency management work package. We also outline the envisioned software architecture for the software services realizing the requirements knowledge and dependency management. Some of the initial results are summarized that will be refined later over the course of the project. The results include the surveys of requirements reuse, requirements management systems, and software product lines and variability as well as the initial ontology focusing on requirements as artifacts.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

OpenReq is a project that aims to enhance requirements engineering activities. The focus areas cover activities in the entire requirements engineering life-cycle, starting from requirements elicitation and analysis to supporting strategic decision making in requirements prioritization, release planning and requirements reuse. The improvements for requirements engineering we are looking for can be achieved through improved requirements management processes, engineering methods and assisting tools.

The “Requirements knowledge and dependency management” work package of OpenReq focuses on the phases after the requirements have been elicited and, in some cases, preliminarily analyzed in terms of validity and quality. The requirements are treated holistically which mean that the different kinds of relationships between requirements are covered rather than treating requirements as detached entities. We collectively refer to any kinds of such relationship between requirements as an *interdependency.* We use the term interdependency to emphasize that we focus on requirement-level artifacts, rather than on general dependencies or traceability between requirements and other artifacts, general plans and specific implementations.

In the core, there are two concerns. First, interdependencies that have not been specified need to be detected. The knowledge about interdependencies need to be then represented holistically in a model. Second, interdependencies need to be taken into account in requirements and product management so that requirements are not considered only as singular entities but also as an interdependent whole.

The requirements knowledge management aims at representing the requirements rigorously. We introduce the approach that will be taken for representing requirements rigorously by an ontology and describe the preliminary ontology proposal. The actual objective of OpenReq ontology for requirements engineering is defined later over the course of the OpenReq project.

The interdependency management of OpenReq takes advantages of technologies and concepts in model-based analyses and diagnosis. We bring in results from software variability and software product lines as well as knowledge-based product configuration. Additional related topics covered in requirements knowledge and dependency management are requirements reuse and requirements patterns use in system’s requirements.

In this document, we first provide an overview of the state of the art and practice for the fields of research that are relevant and related to the work package five. Second, we describe the approach that covers the concepts and technologies that we plan to apply during the project to address the research challenges. Third, we describe the software architecture of the part of the OpenReq infrastructure that will demonstrate, validate and facilitate the practical application of our results.

2 State of the art and practice

This section summarizes the state of the art and practice related to OpenReq knowledge and dependency management. The research methods applied in this section are as follows. We updated the existing tertiary study (Bano2014) on requirements engineering with more recent results following a simplified systematic review protocol adopted from our tertiary study (Raatikainen2017). This resulted in 108 potential additional systematic reviews on requirements engineering that were taken into account for the topics covered in this section. For the specific topics, we have carried out a systematic mapping study for requirements reuse and a survey of tools as described in more detail below in respective subsections. We have also carried out a tertiary study about software product lines and variability (Raatikainen2017) as well rely on our earlier review of knowledge based configuration and variability (Tiihonen2016) to complement our existing knowledge (Tiihonen2014).

2.1 Impacted requirements engineering activities

On the basis of our literature study, the activities in requirements and product management that are primarily involved in, or depended on, requirements knowledge and dependency management are requirements prioritization and release management. The objective of this OpenReq work package is not to develop methods for these activities but, rather, managing requirements knowledge and especially interdependencies for them. Therefore, we only briefly summarize the state of the art in the requirements prioritization and release management below. Requirement specification, documentation and analysis are also concerned activities but artifacts involved in these activities are described more detail later in this document.

Requirements prioritization is an activity during which the most important requirements for the system are discovered (Sommerville2010). Requirements prioritization has been a topic of extensive research, which is evidenced by the several systematic reviews that we summarize as follows. Requirement prioritization is typically concerned about requirement priority demonstration, requirement selection, requirement categorization and requirement value assessment (Thakurta2016). There exist numerous different prioritization technique proposals, e.g., as summarized in the recent systematic reviews covering the prioritization techniques (Sher2014, Pitangueira2015, Thakurta2016). Prioritization is affected by structural complexity, including interdependencies of the system under development (Thakurta2016). Interdependencies, although considered important, are largely neglected or simplified and, e.g., direct simple additivity of property values is presumed (Daneva2008, Herrmann2008, Achimugu2014).

Specifically, in agile development, requirements are managed in a backlog in which interdependencies are one of the most common factor to be taken into account in ordering, i.e. prioritizing, the backlog (Silva2017). Prioritization is often seen an optimization problem for which solutions, such as algorithms, are proposed but the focus of such prioritization methods is, e.g., on novelty, expressiveness or accuracy of the solution rather than use or practical utility (Pergher2013). In fact, little empirical context details are provided and there are few supporting tools for prioritization (Pergher2013). Scalability of prioritization methods is another identified challenge (Babar2011, Achimugu2014).

Release planning is “concerned with selection and assignment of requirements in sequences of releases such that important technical and resource constraints are fulfilled” (Svahnberg2010). This is typically done by a person or team that negotiates priorities of requirements and decides what each release should comprise. In a systematic review of strategic release planning models (Svahnberg2010), 24 different models are distinguished. Requirements interdependencies in general are clearly the most common requirement selection constraint that is taken into account in release planning, yet further details about nature of about interdependencies or interdependencies management are not given.

A more recent survey indicates similar results (Ameller2016). The empirical validation for the majority of models is also noted to be immature. In Open Source development communities, setting development priorities and defining releases can be a collaborative, communicative and consensus-based process (e.g. Raymod1999 and VonHippel2003) for which the RMS needs to provide functionalities, such as commenting and voting for the requirements’ priorities. There are several strategies for defining releases, which can vary from feature-driven, time-based, frequent and further (GomesdaSilva2017).

2.2 Requirements modeling and representation

We summarize some proposed methods for requirements modeling and representation that we have collected from current research literature. Then, we summarize the known reported empirical evidence to outline what is being applied in practice. However, we are not aware of any systematic studies that provide a survey or meta-analysis of requirements modeling and representation approaches.

2.2.1 Requirements modeling and representation approaches

Natural language (NL) is a common means for representing requirements knowledge. Typically, a requirement is expressed by free form sentences. An archetypal form is a “shall”-sentence. An example of a more specific form are structured NL — for example EARS (Mavin2016) proposes five structured NL requirements templates. Another example of structured NL are contextual templates, such as “use cases”, which prescribe pre- and post-conditions for requirements and steps that are carried out to fulfill the requirement.

A development project can start with only a high-level vision of the software’s purpose, which is then used to build a minimally viable prototype for gathering customer feedback. Here, requirements can be expressed as improvement ideas, bug reports or “user stories”, which can later be bundled together as “epic requirements”, giving the requirements body additional structure. In the agile software development environment, requirements can be stored and then organized in spreadsheet-like backlogs in which prioritized requirements can be contained in a hierarchical structure.

In more established environments, textual requirements can be stored in a dedicated requirements management system (RMS), examples which we cover in detail later in this section. In a RMS, the high-level requirements may or may not be refined into work tasks, depending on which style is in use in the software development project. Workflow management tools such as task- or issue trackers can then be used for managing details of what needs to be done for achieving the goals of each requirement. These systems also provide functionalities that help discussing, prioritizing and refining the requirements, which is essential for enabling distributed collaboration.

Beyond NL, different types of requirements modeling notations have been proposed. These often include a graphical notation. Goal modeling (Mavin2017) as exemplified by i* (Dalpiaz2016), is one group of modeling methods that have been proposed for requirements modeling. Feature modeling (Kang1990) is another family of methods that can be used for representing requirements. Finally, we mention UML and SysML as examples of more general modeling languages that are capable of expressing requirements.

Requirements can typically contain a set of *properties* such as priority and effort. Properties can also be called attributes. We use the term *meta-data* for covering additional characteristics, such as change history. Each property, or more precisely a property type, has a value in a requirement. Requirements in different projects generally have different sets of property types. For instance, an analysis found about 280 different property types, concluding that there is no single property type that can be generally relied on to be applicable to any situation (Riegel2015).

One specific means to represent requirements is the Requirements Interchange Format (*ReqIf*), which is a standard of the Object Management Group (OMG) (OMG2016). ReqIf can be used for exchanging requirements between different requirement management systems. ReqIF provides concepts for representing requirement documents that consist of individual requirements organized in a compositional (part-of) hierarchy. Different requirement types can be defined, each with many type-specific properties. These are instantiated to requirement document(s). Binary relationship types with optional properties can be defined and instantiated between individual requirements. Actual content of a requirement is expressed as text or rich text (XHTML) attribute value, complemented with additional attributes. In other words, no built-in semantics exists in ReqIf.

2.2.2 Empirical studies on requirements modeling and representation

In order to find out what kinds of requirements models and representation are being used, we summarize in the following the results of our review. Since 2010, several empirical studies have been conducted to explore aspects related to specification and modelling of requirements. These are summarized in Table 1. We manually review requirements engineering related conferences RE, REFSQ, ESEM, FSE/ESEC and ICSE, and journals REJ, IST and EMSE from 2010 onwards. We classify them into two major groups, taking into account the scope of each study. The first group contains empirical studies that investigate general practices on RE and include some results related to the specification and modelling of requirements. In this group, two studies are focused on investigating the challenges and needs of RE, either in general or for a specific type of requirements or software project (Hiisilä2015, Sikora2011), and three studies exemplify state-of-the-practices in requirements engineering (Méndez2015, Palomares2017, Raatikainen2011). In the second group, we find empirical studies that are specifically focused on the specification and modelling of requirements (Hotomski2016). We summarize below some highlights of these studies and results that are related to the scope of this report.

In (Hiisilä2015), one of the goals is to investigate what the challenges of a customer organization RE process are in an outsourced development environment. With that purpose, they conduct a case study in a Finnish insurance company, performing 17 interviews and analyzing 15 large software development projects. In addition, the authors conducted five workshops in a company to validate their results. This extensive case study highlights one challenge in particular: how to model requirements that cover comprehensively different aspects from the enterprise context?

A qualitative study (Hotomski2016) explores the current practices for managing two related types of software documentation: requirements and acceptance tests. They interviewed 20 practitioners from 17 business units in 15 companies to investigate the company practices for writing, maintaining and linking requirements and acceptance test documents. The results are related to how requirements and text documents are modelled in practice, how requirements and acceptances tests are updated, and what difficulties are faced in the project. The results show that NL is widely used by most of the participants — in the case of waterfall companies, they use mostly plain requirements (i.e., free-format requirements), while in the case of agile companies, they use mostly user stories.

The design of a globally distributed family of surveys to study the state-of-the-practice in RE as well as the results of the first run of the survey in Germany with 58 participants are presented in (Méndez2015), which was later extended to ten countries and 228 companies (Mendez2017). In the results, participants agree on the fact that the definition of standardized RE artefacts with document templates across different projects environments or tool support is important.

The goal in (Palomares2017) is to investigate the state of the practice in the reuse of requirements. The authors conducted an exploratory survey based on an online questionnaire. In addition to questions related to requirements reuse, they also include questions related to the participants' background or to general RE practices used in the participant's daily work, such as specification languages. They received 71 responses from requirements engineers with industrial experience in the field. In the results, the largest share of responses (57 participants) uses requirements in NL (being just plain requirements, use cases or other scenario-based approaches), followed by UML (38 participants) and goal-oriented languages (5 participants). 12 participants state to use other languages, from which four used BPMN to write requirements.

The state-of-the-practice of RE in the nuclear energy domain in Finland is presented based on a descriptive case study that focuses on the safety-related automation systems of the nuclear power plants (Raatikainen2011). Data was collected by interviewing two nuclear energy domain experts representing public authority and five experts working at the three power companies (utilities). Here, practically all requirements specifications were written using NL, causing the challenge of how representing interdependencies among requirements and beyond requirements.

To identify the key industry needs, (Sikora2011) conducted an in-depth study with representatives from large, internationally operating companies in the domain of embedded systems in Germany. The authors interviewed ten practitioners with a clear view of the RE needs on their companies, and they collected extra data by means of pre- and post-interview questionnaires sent to the same participants. Their results are related to the use of NL and requirements models, the support during RE for high system complexity, the quality assurance for requirements and the interrelation of RE and the architectural design of embedded systems software. In detail, their results show that most of the participants use NL to specify requirements, and that requirements models do not have a widespread use. Additionally, a challenge related to the specification and modelling of requirements is identified: the use of NL to specify requirements is not satisfactory.

Table 1. Empirical studies on specification and modelling

Source	**Relevant Results on Specification and Modelling**
(Hiisilä2015)	● *Challenges:* Modelling comprehensive requirements from the enterprise context.
(Hotomski2016)	● *Languages:* NL (waterfall companies use mostly plain requirements, agile companies use mostly user stories).
(Méndez2015)	● *Templates & Tools:* The definition of standardized RE artefacts with document templates across different projects environments and/or tool support is important.
(Palomares2017)	● *Languages:* NL (plain requirements, use cases or other scenario-based approaches) (57 participants); UML (38 participants); Goal-oriented languages (5 participants); other (12 participants).
(Raatikainen2011)	● *Languages:* NL. ● *Challenges:* Modelling interdependencies among requirements.
(Sikora2011)	● *Languages:* NL rated as often or always (9 participants); Models are not common (8 participants sometimes or rarely use them). It is much easier to understand complex requirements if they are specified by means of models. ● *Challenges:* Use of NL to specify requirements is not satisfactory (5 participants).

As a summary, natural language in its various forms seems to be the most commonly used method for representing requirements. Different kinds of diagrams are rarely applied or are applied only to a part of the requirements that are used to specify software systems. Therefore, we will focus on natural language requirements in OpenReq.

2.3 Requirements interdependencies

Next, we first elaborate the concept of interdependency in general and introduce taxonomies that have been proposed for defining requirements interdependencies. We also summarize results from empirical studies on experience of interdependencies in their realistic context in requirements management.

2.3.1 The concept of interdependency as traceability

Interdependencies of requirements are considered a special class of requirements traceability focusing only on information about requirements (Dahlstedt2005, Zhang2014). In general, requirements traceability is defined as: "ability to describe and follow the life of a requirement, in both forward and backward direction, ideally through the whole system life cycle" (Gotel1994).

Two different requirements traceability relations can be differentiated from which direction (from-to) they take can also be separated (Dahlstedt2005):

- *Pre-traceability* refers roughly to the domain that requirements concern.
- *Post-traceability* refers to the realization of requirements, e.g. by completed work tasks or developed software artifacts.

Traceability can also differentiate between (Dahlstedt2005):

- *Horizontal* traceability between the information of same type
- *Vertical* traceability between previous and subsequent phases in the development process i.e. between information objects of different types

Alternatively, the scope of interdependencies can be divided and simplified as (Carlshamre2000):

- *Interna*l from a set of requirements to another set of requirements referring to interdependencies.
- *External* from a set of requirement to other artifacts than requirements encompassing different traceability.

Consequently, an interdependency is horizontal traceability or dependency within same types of artefacts, such as requirements artefacts. There can be different types of requirements such as epics and user stories. In addition, user and technical requirements can be involved in interdependencies because the information is still the same type. Pre- and post-traceability are more general and thus not within the scope of our definition of interdependencies.

The nature of interdependency can be (Carlshamre2000):

- Explicit i.e. presentation exist explicitly as the content or property of a requirement.
- Implicit so that interdependency appears dynamically. For example, a set of requirements are not initially dependent but when they are selected to a release, they become dependent.

Even in explicit interdependencies, it is possible that their presentation is not captured in any model and that the interdependencies exist only logically.

In OpenReq, we focus primarily on interdependencies or horizontal traceability. Therefore, the following discussion is carried out in the light of interdependencies, although we do not disregard other traceability links. We treat requirement dependency roughly as a synonym for interdependency, yet emphasize the intention by preferring the term interdependency.

2.3.2 Interdependency types

Although interdependencies have been identified and even emphasized in research literature, there are only a few taxonomies (or classifications or typifications) that elaborate the nature of interdependencies. Not all interdependency taxonomies focus solely on interdependencies, but cover also traceability beyond interdependencies but we give a full account on each taxonomy. Most of the introduced taxonomies provide a grouping of interdependency types. The full taxonomies are tabulated in Appendix 1 including more detailed authors' definitions and groupings presented in italic font.

First, Pohl taxonomy is, to the best of our knowledge, the earliest taxonomy of interdependencies. It is based on a literature survey that covers roughly 30 research papers. This taxonomy has the largest number of interdependency types. The meaning of each interdependency type is sometimes described very shortly. The intention of Pohl taxonomy appears to be general for requirements engineering.

Second, Carlshamre taxonomy (Carlshamre2001) emerged from the phases of requirements prioritization and release management. As for the previous work used in Carlshamre taxonomy, there is a taxonomy of the following interdependency types: 1) cannot exist, 2) must exist, 3) positive cost, 4) negative cost, 5) positive value, and 6) negative value (Karlsson1997). These types are further specified for the release and product management purposes resulting in the taxonomy in question.

Third, Dahlstedt taxonomy (Dahlstedt2005) provides a more recent synthesis of interdependencies in the literature of requirements engineering in general. The focus of this taxonomy is on requirements interdependencies, which is explicitly differentiated from traceability. Dahlstedt taxonomy takes into account the aforementioned two taxonomies. The objective of Dahlstedt taxonomy is to synthesize and abstract interdependency types in order to result in a simpler taxonomy of smaller number of interdependency types. However, exact definitions are not provided for interdependency but the interdependency types are merely informally described. The types are not provided with a description of empirical assessment, which is reported to be in the author's agenda.

Fourth, Zhang taxonomy (Zhang2014) studied the abovementioned taxonomies. They took both Pohl and Dahlstedt taxonomies as the basis for their study, tested them empirically and proposed a Zhang taxonomy, which is provided in more formal definitions than its predecessors. The Zhang taxonomy is based on removing interdependency types that are not common or seldom found in practice (such as Test_case_for and Purpose). Authors also clarify confusing or ambiguous interdependency type definitions (e.g., Conflicts and Conflicts_with), combining and refining some overlapping and alternative interdependency types in the different models (e.g., Similar and Similar_ to). They also introduce new interdependency types (e.g., Be_exception_of). The full details is presented in the original study (Zhang2014).

In addition, a general model of grouping interdependency types was developed (Zhang2014). It consists of two classes at the top level and at the lower level into six classes, which, however, are not clearly described. The top-level class called ‘Intrinsic interdependencies’ consists of essential interrelated states of requirements, reflecting semantic and structural information of requirements. The ‘Additional interdependencies’ class covers e.g., release planning. Intrinsic interdependencies are considered more important for software engineers in discovering interdependencies because they may affect many software engineering activities, including requirements change impact analysis and project planning. They are also generally helpful in identifying additional interdependencies. Intrinsic interdependencies are divided into business, implementation, structure, and evolution classes while additional interdependencies consist of value and cost classes.

The existing studies on interdependencies in requirements engineering seem to utilize the above-mentioned taxonomies or subsets of types defined in these taxonomies. However, it is common that the exact types or semantics of the interdependency are not covered. That is, interdependency is treated as an important concern, but covered in an abstract or general manner. They are generally referred as “(inter)dependency” without any explicit semantics.

To summarize, there seem to be some taxonomies that share many similarities with each other, but vary especially in terms of number of interdependency types they present. Two taxonomies focus solely on binary relationships in interdependencies similarly as ReqIF while two taxonomies seem to allow more complex many-to-many relationships. However, they do not explicitly state this nor provide details or examples on any non-binary interdependency. In terms of using dependencies, e.g. in prioritization or release management, little or no details of the interdependency types or semantics are given although interdependencies are considered important. Therefore, rather than analytically synthesizing the taxonomies, we elaborate below the empirical experience of interdependencies.

2.3.3 Empirical experiences of interdependencies[1]

For the aforementioned Carlshamre taxonomy, studies have been conducted in empirical settings with Swedish companies in which the interdependency types were applied to 20 requirements in five realistic cases (Carlshamre2001). The researchers found that it is not easy to differentiate all interdependency types: as an example, differentiating the *AND/REQUIRES* interdependency from *CVALUE* is difficult in cases where two requirements are always linked, but not strictly require each other. The study also revealed that *CVALUE* and *ICOST* are common interdependencies in market-driven or evolutionary development when new features are added, whereas *AND* and *REQUIRES* are common in bespoken or early phase of development. The authors point out that *CVALUE* and *ICOST* are sometimes found in combination. However, especially these value-based interdependencies are often subjective, especially in measures. In general, the requirements managers who participated in the study were sure about correctness of interdependencies and they identified that there is no need for a fuzziness or confidence factor.

Only a few requirements are singular i.e. have no interdependencies while a few requirements cover most of the interdependencies. It was found that these highly interdependent requirements can be easily identified. Conflicting requirements were not found, probably because requirements could have been already negotiated. In further application of the taxonomy, it was found that the type of an interdependency does not matter as significantly as the strength of the interdependency (Carlshhamre2002). That is, there can be imperative (normative) value-related interdependencies (*CVALUE, ICOST*) as well as negotiable functional (normative) interdependencies (*AND, REQUIRES*).

Pohl and Dahlstedt taxonomies were evaluated by applying them to an existing system by three engineers in a case study of change propagation while rewriting the system. The study resulted in the following findings (Zhang2014). *Precondition* is identical with *Requires* and the most common interdependency within individual modules and in between modules. *Constraint* can be used to describe the relationships between non-functional and functional requirements, yet what is meant by a ‘constraint’ is not clearly explained and commonly agreed. Similar ambiguity is in

[1] For details of the interdependency types, see Appendix.

that s*atisfies* describes a specific kind of constraint relationship between requirements that seem to be different than intended in the original taxonomies. The *compares* and *conflicts* interdependency types are hard or ambiguous to understand and often misused, if at all. Some interdependency types could not be found at all in the requirements, such as *example_for*, *test_case_for*, *background*, and *purpose*. The value-based interdependency types *increase/decrease_value_of* and *increase/decrease_cost_of* are hard to quantify. Finally, the use of interdependencies is context-dependent, meaning that, e.g., the background, role in organization, and existing knowledge affect largely.

In addition, few industrial studies emphasize that interdependencies are important in requirements engineering (Lehtola2004, Vogelsang2010) in general. However, a more detailed typology of the interdependencies is not provided in these studies.

As a summary, interdependencies are a key concept for requirements knowledge and dependency management of OpenReq. However, few empirical studies assess the interdependency taxonomies, or otherwise provide details or semantics for interdependencies. These studies have also different focus and have been conducted only in relatively simple or limited contexts. Therefore, the results are not very conclusive. The existing taxonomies seem to be sufficient enough for the purposes of OpenReq and there even seems to be extraneous interdependency types that have little practical value. The existing studies indicate that relevant interdependency types are very context dependent, based on the nature of products such as bespoken or market driven, and the task at hand such as release planning or refactoring.

2.4 Interdependency detection in natural language requirements

Several works deal with the detection of interdependencies when requirements are specified using natural language. This body of research can be classified in different groups. Firstly, when interdependencies are explicitly stated in natural language texts in the body of requirements description, their identification is related to cross-references identification. Secondly, interdependencies can be identified by detecting similar requirements. Thirdly, several works can be found for identifying the specific type of interdependency namely inconsistency. Each one of the previous groups is presented in the following subsections.

2.4.1 Cross-reference detection

Cross-reference detection aims to identify the natural language (NL) expressions that denote cross references, i.e. a piece of text within a document which refers to related information elsewhere, in the same or different document, such as “This decision [referring to the previous requirement] has been taken due to the requirements stated in section 5.1”. The *interpretation* of these expressions and linking them to the targets is a part of cross-reference *resolution*. *Identifying* and resolving cross references in text is a part of the more general problem of requirements traceability, and therefore of requirements interdependencies. Cross-reference detection has been specially used in the field of legal requirements. Below, we describe research approaches related to detecting cross-references in legal texts.

Two studies (Breaux2008, Palmirani2003) identify natural language patterns for cross references. The former is based on a study of different US regulations and the latter one is based on guidelines for the Italian legal corpus. However, they tackle only cross-reference identification and not resolution, and therefore the automation for text structure markup is not provided. Breaux (Breaux2008) goes one step further and proposes the use of an explicit schema for modeling the structure of legal texts.

An approach for resolving external cross references is proposed in (Hamdaqa2009), where automated markup is generated using manually written regular expressions. These regular expressions (i.e., patterns) are defined by means of finite state machines. The identified patterns are limited, in the sense that they apply only to external cross references, and that they are exclusively based on best practices and thus insufficient for the richer citation styles used in actual texts.

Machine learning for cross reference identification and resolution in Japanese legislative texts is applied in (Tran2014). The use of machine learning can be advantageous since it does not require an a-priori specification of the patterns in cross references. However, it makes much more difficult to detect patterns with recursive structures or multiple layers, and therefore this kind of patterns are not handled in this work.

The approaches presented in (Sannier2017) and (deMaat2006) are similar, differences in language aside, and the patterns observed in their cross-reference detection are closely aligned. In (Sannier2017), it is proposed an approach for automated detection and resolution of cross-references, which leverages the structure of legal texts, formalized into a schema, and a set of NL patterns for legal cross-reference expressions. These patterns are based on an investigation of Luxembourg's legislation written in French.

The most important features of the approach are: 1) The use of a schema enables, using techniques from NLP, to automatically derive the necessary regular expressions for text markup generation; 2) It addresses, in an algorithmic way, subtleties that one needs to take into account with regard to the interpretation of complex cross reference expressions; and 3) It devises a set of cross-reference patterns in a very detailed and complete manner for cross-reference identification.

In the other study (deMaat2006), the patterns used in the cross references appear in the Dutch laws. This approach assumes that legal texts are already in a markup format with adequate structure to be transformed into the markup format required by the approach (contrasted to (Sannier2017), which did not require pre-existing markup. Another difference between the approaches is that (deMaat2006) does not elaborate the resolution process, while (Sannier2017) provides a detailed treatment of resolution.

2.4.2 Similarity detection

Similarity detection, also known as paraphrase detection, is an approach closely related to detection of interdependencies in between requirements. This relationship is evident in the case where two requirements have almost the exact same formulation, since in this case we would have an OR interdependency between the requirements. Imagine the requirements *The interface* *should use the letter type Arial* and *The interface should use the letter type Calibri*; it is clear that these two requirements are similar (except for the words *Aria*l and *Calibri*) and they cannot be used in the same system (since it is not possible two use two letter types for the whole system interface), so these requirements are related by an OR interdependency. However, even in other cases there are commonalities. As an example, if requirement R1 states that “It shall be possible to filter by name and address.” and requirements R2 states that “It shall be possible to filter by age.” it would probably be wise to treat the two requirements at the same time to save development resources. This example can be considered as an ICOST interdependency, by the terminology described above (Carlshamre2001). Other examples of interdependencies that can trigger a similarity analysis at a lexical level can be, e.g., the conflicting requirements “The button shall be blue.” and “The button shall be red.” In the following, we present some approaches that tackle the identification of this type of similarity in NL texts.

One approach to detect similar texts is to do a number of pre-processing steps based on NLP (e.g. breaking into words, removing stop words, stemming words, etc.) and a subsequent calculation of a similarity measure (e.g., Dice coefficient, Jaccard coefficient, Cosine coefficient, etc.). Several existing works follow this approach (NattOchDag2002; Prifti2011; Runeson2007; Sun2011; Wang2008), and most of them have been done in the field of textual documents or reports similarity — only the work of (NattOchDag2002) is specifically about requirements similarity. Table 2 presents a summary of these works in terms of what preprocessing steps are used and the similarity measures used.

Table 2. Similarity works based on NLP techniques and subsequent similarity calculation

Source	**Similarity Scope**	**NLP preprocessing**	**Similarity measures**
(NattOchDag2002)	Requirements	1) Lexical analysis, 2) Stop words removal, 3) Stemming	Cosine, Dice, Jaccard
(Prifti2011)	Reports	1) Tokenization, 2) Stemming, 3) Stop words removal	Cosine
(Runeson2007)	Reports	1) Tokenization, 2) Stemming, 3) Stop words removal, 4) Spellchecking, 5) Vector space model representation	Cosine
(Sun2011)	Reports	1) Tokenization, 2) Stemming, 3) Stop words removal	BM25F extension
(Wang2008)	Reports	1) Stemming, 2) Stop words removal, 3) Vector space model representation	Cosine

Well-known similarity techniques include *latent semantic analysis* (LSA), also known as *Latent Semantic Indexing* (LSI), which is a fully automatic mathematical/statistical technique that analyzes a large corpus of NL text and provides a similarity representation of words and text passages (Foltz1998). In LSA, a group of terms representing an article is extracted by judging from among many contexts, and a term-document matrix is built to describe the frequency of occurrence of terms in documents. Then, the matrix representing the article is divided by Singular Value Decomposition (SVD). When the LSA is applied to calculate the similarity between texts, the vector of each text is transformed into a reduced dimensional space, while the similarity between two texts is obtained from calculating the two vectors of the reduced dimension. One of the standard probabilistic models of LSA is the Probabilistic Latent Semantic Analysis (PLSA), which is also known as Probabilistic Latent Semantic Indexing (PLSI) (Hofmann1999). PLSA uses mixture decomposition to model the co-occurrence words and documents, where the probabilities are obtained by a convex combination of the aspects.

Random projection (RP) is another powerful technique for dimensionality reduction of the matrices representing a text: given a matrix X, the dimensionality of the data can be reduced by projecting it through the origin onto a lower- dimensional subspace, formed by a set of random vectors of the desired, reduced dimensionality (Lin2003).

Another important study is the Hyperspace Analog to Language (HAL) (Burgess1998). HAL and LSA share very similar attributes: they both use concurrent vocabularies to retrieve the meaning of a term. In contrast to LSA, HAL uses a paragraph or document as a unit to establish the information matrix of a term. HAL uses a window matrix scans through the entire corpus, using □ terms as the width of the term window (normally with □ = 10 terms) to record in a □ x □ matrix the weight of each shared term (number of occurrence/frequency). A 2□ dimensional vector of a term can be acquired by combining the lines and rows of the matrix corresponding to the term, and the similarity between two texts can be calculated by the approximate Euclidean distance. However, HAL has less satisfactory results than LSA when calculating short texts.

Other approaches are found in literature to deal with similarity detection. A study (Lee2014) obtains similarity from semantic and syntactic information contained in the compared NL sentences by using grammatical rules and the WordNet ontology. A set of grammar matrices is built for representing the relationships between pairs of sentences. Here, a NL sentence is considered as a sequence of links instead of separated words, each of which contains a specific meaning. The latent semantic of words is calculated via a WordNet similarity measure.

(Pedersen2005) presents SenseClusters, an unsupervised approach that is language independent, and uses no knowledge other than what is available in raw unannotated corpora to cluster together similar contexts. SenseClusters represents the contexts to be clustered using either a first order or second order representation. For the first order representation, a matrix where each row represents a context and the columns represent the identified lexical features is created, following a word sense discrimination approach that uses first order vectors based on local syntactic features to represent contexts. For the second order contexts, a co-occurrence matrix is constructed using a word sense discrimination method based on bigrams, co-occurrences, or target co–occurrence features identified The resulting context vectors represented using either first order representation or second order representation are clustered as the next step, using either vector spaces or different similarity measures (e.g. Cosine, Jaccard, Dice, etc.).

A *similarity metric* proposed in (Wang2012) computes the probabilistic edit distance of two NL sentences as predictions of semantic similarity. It learns weighted edit distance in a probabilistic Finite State Machine (pFSM) model, where state transitions correspond to edit operations that are even able capture long-distance word swapping or cross alignments. A FSM defines a language by accepting a string of input tokens in the language, and rejecting those that are not. A probabilistic FSM defines the probability that a string is in a language, extending on the concept of a FSM. Unlike other applications of FSMs where tokens in the language are words (e.g. Vidal2005), in the FSM language of (Wang2012) tokens are edit operations. A string of tokens that the FSM accepts is an edit sequence that transforms one side of the sentence pair into the other side.

Finally, in (Grabilovich2007) and (Chen2016) similarity is tackled from the machine learning perspective. The first one (Grabilovich2007) proposes a method that uses machine learning-based text classification techniques to build a semantic interpreter that maps fragments of natural language text into a weighted sequence of Wikipedia concepts ordered by their relevance to the input. This way, input texts are represented as weighted vectors of concepts, called interpretation vectors. The meaning of a text fragment is thus interpreted in terms of its affinity with a host of Wikipedia concepts. Then, the relatedness of texts in this space is assessed by comparing the corresponding vectors using conventional similarity metrics (e.g., Cosine). In (Chen2016), Latent Dirichlet Allocation (LDA) topic model and K-Nearest Neighbor algorithms are used in a short text classification approach that could be extrapolated to classify similar requirements together. LDA is a probabilistic, generative model, which in this case is used to generate the probabilistic topic of a text, since each topic is characterized by a probabilistic distribution over words.

To measure similarity, the model assumes that the discriminative words between two short texts usually have important information, which can reveal their implicit relationships. If there are several common latent topics related to the discriminative words in the two short texts, these latent topics can be used to compare each short text mutually. Therefore, two short texts can be assigned to the same class where their latent topics are similar. Then, KNN is used with a Euclidean distance to measure the similarity of the topics represented in a text and assigns the text to the most similar topic.

2.4.3 Inconsistency detection

The issues of detecting *inconsistencies* in requirements have received less attention when compared to detecting inconsistencies in requirements models (e.g., Escalona2013, Perrouin2009, Ali2013).

Inconsistency detection has been discussed implicitly in (Jain2009) and (Verma2008), whereas in (Zhu2005), a domain ontology is used as the basis for identifying inconsistencies. The domain ontology, which is based on an abstract requirements refinement process model, serves as an infrastructure for the refinement of software requirements, with the aim of acquiring comparable requirements descriptions. Thus, requirements inconsistency can be measured based on tangent plane of requirements refinement tree, after inconsistent relations of leaf nodes at semantic level have been detected.

Inconsistency detection in (Misra2016) is based on a content analysis technique that exploits the extraction, from requirement documents, of the interactions between the entities described in the document as Subject-Action-Object (SAO) triples, which are obtained using an NLP syntactic parser. This approach returns a measure of how much a part of a document deals with a certain topic. These measurements are obtained by assigning a score to each SAO, according to suitable weights for Subject, Action, Object and to a set of dictionaries related to the functionalities to be investigated. These scores can help to detect in the document, among others, sources of potential inconsistency.

2.5 Requirements reuse and patterns

Requirements reuse and *patterns* are specific complementary topics of requirements knowledge and dependency management. In a nutshell, requirements reuse refers to taking advantage of requirements knowledge obtained from previous IT projects and later on using this knowledge in a new one. Software patterns, thoroughly presented by (Schmidt 1996), are attempts to describe successful solutions to common software problems. In this section, we use the term requirement pattern to refer to those assets that provide a structured representation to reuse knowledge about actual requirements. Therefore, we exclude in the term requirement pattern assets related to the transformation of requirements (mainly, formalization rules and MDD), best practices in RE, way of writing guidelines, etc. Therefore, boilerplates, grammars and specification languages are not included in our description of requirement pattern.

The following subsection represents a literature-based background on requirements reuse and requirement patterns. The second subsection focuses on technology transfer by reporting the applications of academic proposals of requirements reuse and patterns in industry.

2.5.1 Background on requirements reuse and requirements patterns

This subsection is based on a systematic mapping (SMAP) of research publications, which we conducted following a rigorous protocol according to the guidelines described in (Petersen2015) and in (Kitchenham2007). The publications retrieved are from 1995 to mid-2017. We ran an automatic search of papers in several major digital libraries, including IEEE Xplore, ACM Digital Library, Springer Link and Science Direct. Our search looked for publications that included the words “reuse” or “pattern”, and “requirements” either in the title or in the abstract and it was tailored to the capabilities of each library. After filtering by title, abstract, and quick read protocol, 316 publications were selected. From them, 79 were found to use requirement patterns, showing the importance of patterns as a vehicle for requirement reuse, especially in the last 7 years (2010–2017) of research. This timespan covered 47 out of the 79 (60%) publications. We grouped the 79 publications into 69 proposals, since there were several papers that dealt with the same proposal.

In Table 3, we show a representative sample of the publications found. The sample has been chosen to be able to illustrate each value of each characteristic of a requirement reuse and patterns approach.

Table 3. Examples of requirements reuse and patterns

Source	Artifact	Size	Abstraction	Meta-model	Scope	Purpose	Process	Repo-sitory
(Bouraga2014)	i* models	Requirement clusters	Patterns	Yes	Social networks	Specification	-	Yes
(Caralt2007)	Ontology, use cases, interdependencies	Requirements	Patterns	Yes	General	Specification	Yes	-

(Carrillo-de-Gea2013)	Natural language requirements	Requirements	Patterns	-	General	Elicitation, specification	Specific process	Yes
(Chung2006)	Use cases, NFR diagrams	Requirement specifications	Patterns	Yes	General	Specification	-	-
(Daramola2012)	Natural language requirements, classification, ontology	Requirements	Templates	-	Security	Elicitation, specification	Specific process	Yes
(Dehlinger2005)	Natural language requirements, use cases	Requirement clusters	Patterns	-	Agent-based requirements	Elicitation, specification	Yes	Yes
(Franch2013, Renault2009)	Natural language requirements, classification, interdependen cies	Requirement clusters	Patterns	Yes	General	Elicitation, specification	Specific process	Yes
(Hauksdottir201 2)	Natural language requirements	Requirement clusters	Template variability models	-	General	Elicitation, specification	Specific process	Yes
(Jensen2009)	Natural language requirements	Requirements	No	-	Healthcare systems security	Elicitation, specification	Specific process	Yes
(Konrad2002, Konrad2005)	Diagrams, relationships	Requirement clusters	Patterns	-	Embedded systems	Specification	-	-
(Mannion1999)	Natural language requirements	Requirement clusters	Patterns	Yes	General	Elicitation, specification	Specific process	Yes
(Mazo2016)	Natural language requirements structure, domain models	Requirement clusters	Patterns	Yes	General	Specification	-	Yes
(Pacheco2017)	Natural language requirements, classification	Requirements	Templates	-	General	Elicitation, specification	Specific process	Yes
(Panis2015)	Natural language requirements, classification	Requirements	Templates	-	General	Specification	Specific process	Yes
(Toval2002)	Use cases, classification, interdependen cies	Requirements	Patterns	Yes	Security	Specification	-	Yes

(Wahono2002)	Natural language requirements, use cases, interdependencies	Requirement clusters	Patterns	-	Web applications	Elicitation, specification	Yes	Yes
(Withall2007)	Natural language requirements, interdependencies	Requirements	Patterns	-	General	Elicitation, specification	Specific process	Yes

As represented by the references included in Table 3, the common purposes of all of the proposals is the elicitation or specification of requirements by *artifacts*. In the reviewed studies, requirements were mainly specified in natural language (e.g., Withall2007), use cases (e.g., Chung2006; Dehlinger2005), and domain models (Toval2002). More formal representation of requirements are also used in some approaches, such as i* models (Bouraga2014), ontologies (Caralt2007) and syntactical structure of requirements (Mazo2016). Other artifacts that may be reused, which are not strictly requirements, include classifications of requirements in a requirements specification (e.g., (Panis2015; Franch2013)), relationships or interdependencies among requirements and other reuse artifacts (e.g., (Franch2013, Konrad2002)).

The *size* of reusable artifacts varies from individual requirements (e.g., (Daramola2012)) to requirement clusters (e.g., (Konrad2002)) and further to parts of requirement specifications and even to complete requirement specifications (e.g., (Chung2006)).

In some approaches, reuse specific requirements or complete requirements specifications without any *abstractions* are reused. Other proposals add abstraction to the reuse knowledge by using templates, patterns or feature models. The lower level of abstraction is applied with the following techniques: 1) templates that are natural language sentences with no required structure (e.g., (Hauksdottir2012)); 2) templates with a basic structure that may include parameters (e.g., (Daramola2012, Pacheco2017)); 3) requirements with a required structure that are compliant with language grammar (e.g. (Konrad2005) applied in (Post2011)). At the highest level of elaboration, we find patterns (e.g., (Withall2007 Franch2013)) and feature models proposed in domain engineering (e.g. (Mannion1999)). These approaches are based on the notion of variability of requirements.

All the proposals incorporating a *metamodel* do it to formally describe the artifact to be reused. In addition, some proposals also model with this metamodel the interdependencies and the arrangement of the interdependencies into a catalogue of reusable artifacts (Caralt2007, Franch2013; Mazo2016, Toval2002).

There are differences in terms of *scope*. Most of the published proposals are general even though the papers give examples of reuse in specific domains. Remaining proposals are specific for a particular domain, such as for web applications (Wahono2002), embedded systems (Konrad2002), or security requirements (e.g., (Daramola2012; Jensen2009)), without any aim at generalization.

The purpose of the proposals focus on the elicitation or specification of requirements because of the search criteria applied. Several proposals describe the process for applying requirements reuse, for instance those by (Pacheco2017, Carrillo-de-Gea2013, Renault2009).

Although most proposals give some ideas about how to structure the repositories of reusable artifacts, we highlight (Franch2013, Dehlinger2005, Pacheco2017), that give detailed descriptions of how to construct such repositories and how to classify and identify artifacts that are suitable for reuse.

2.5.2 Practice of requirements reuse and requirements patterns

The low percentage of research papers on software engineering that include industrial validation was already reported by (Lam1997a). More recently, the results of the systematic mapping of reusable knowledge on security requirements by (Souag2015) and the systematic literature review on requirements reuse (Irshad2017) corroborate the results: in these studies, no more than 22% of the identified papers performed an experimental validation in industry. In the case of the 316 publications reported in the previous subsection, only 81 conducted an empirical study to endorse the proposal and only a few of these studies were carried out in the industry (e.g., Eriksson2009; Goldin2013; Rine2000). Considering the 69 proposals that use patterns (see previous subsection), only 19 included some empirical study, and only 12 had conducted the study in industry settings. We classify these 12 approaches according to their empirical study approach as:

- *Case studies* that test the usefulness of the approach, making special emphasis on the percentage of requirements that has been reused (Issa2010; Issa2011; Mahmoud2010; Myklebust2014; Renault2009).
- *Interviews* of experts that explore the usefulness, advantages and disadvantages of the approach (Issa2010; Issa2011; Lam1997b).
- *Industrial experiences* that explain how requirements reuse is being applied in real industrial settings (Daramola2012; Hauksdottir2012; Hauksdottir2016; Heumesser2003; Panis2015; Zuccato2011).

2.6 Requirement management systems

The final OpenReq solution aims to provide requirements knowledge and dependency management functionalities that operate within existing, state of the practice requirement management systems (RMS). Therefore, we feel necessary to present a survey of the RMS that currently exist and their characteristics. We describe first the overview of the study and then three themes that are covered. The full data is available in Appendix.

2.6.1 Overview of RMSs study

First, we describe the OpenReq-relevant functionalities that RMS currently in the market exhibit. There is a large number of RMS as shown in the list of 91 non-discontinued RMS (Birk2017). Due to the impossibility to study all of these 91 systems, we used three reports published by consultant companies in 2016-2017 (Murphy2016, LeClair2016, Beatty2016) and one blog post on the subject (Birk2017) to identify the RMSs with significant market share and market presence. The RMSs selected to be elaborated in this state of the practice were the ones that appear as minimum in three of the indicated sources and for which we could download a demo version. The tools we selected are: Caliber, IBM® Rational® DOORS® Next Generation, Helix RM, inteGREAT (now named Modern Requirements4TFS), Jama, Jira, Polarion® REQUIREMENTS™, and TopTeam.

Our analysis procedure consisted of four intertwined steps that were iterated for each tool as deemed convenient. First, we read the public documentation on the tools. Second, we watched available video tutorials. Third, we installed the available demo versions. Fourth, we prepared a summary describing the relevant aspects of each tool in relation to OpenReq.

Below, we summarize the OpenReq relevant functionalities of the selected tools in terms of three themes: 1) Requirements Reuse, 2) Interdependency Extraction and Management, and 3) Other Support Techniques.

2.6.2 Requirements Reuse in RMS

The types of requirement reuse techniques identified in the analyzed tools are the following:

- **Copy and Paste (cloning) of Requirements:** The basic option of requirements reuse. It allows to copy and paste requirements between projects, or between different parts of the same project. The requirements become a part of the project where they are added to and lose the link to the original one.
- **Mapping of Requirements:** This option allows to map one (base) requirement to a new requirement, which is included in the same or different project. The difference with respect to the copy and paste technique is that the values of the mapped requirements remain equal with the base requirement. Thus, when a base requirement is changed in a project, the changes are propagated to all mapped requirements. There are different variants of this technique depending on which properties or relationships are mapped. It is possible to provide a functionality to notify the requirements engineer if some of the base requirements have changed, and, so, the mapped requirement has or will be changed. There are some tools where the synchronization of changes is not automatically applied and it may have to be approved by a user after seeing that the mapped requirement is out of synchronization. To facilitate the use of the feature, three RMS provide a visualization of mapping interdependencies between requirements.
- **Use of Requirement Templates:** This option allows to define templates (or types) of requirements. A requirement can be created from a template. After a new requirement is created, the requirement properties and the values of properties are independent of the template in which it is based on. Any changes to the template are not reflected in the new requirement.
- **Use of a Project Template:** This functionality allows to reuse the entire structure and all requirements of a project template by importing this structure and requirements into a new project. There are tools that include this technique that allow to the user to either preserve the consistency between the project template and the newly created project or discard the consistency or not.

- **Use of Requirement Libraries:** The RMS allows to define requirement libraries in which groups requirements are going to be reused together. This functionality is interesting especially in the case of requirements related with regulations, security rules, etc. Tools that include this functionality can preserve consistency between the changes in the libraries and the projects where they are used or discarded.

The summary of how the analyzed RMS cover the different requirements reuse techniques can be found in the Appendix 2 *Full details of RMS comparison*. Next, we highlight the main differences.

Six out of the eight studied RMS provide the “Copy and Paste of Requirements” technique. The difference between the RMSs is what properties of the requirement are being copied to the new requirement. Specifically, if the interdependencies (or links as called in most of the tools) are also copied. For instance, Caliber allows deciding whether or not a child requirement is cloned, but other interdependency types cannot be cloned. DOORS allows to include in the copy all interdependencies. For Modern Requirements4TFS and TopTeam, we can say that they have the functionality but on the basis of the demo version we downloaded and the documentation that we could access, we cannot say which parts of the requirement they allow to be copied.

Five out of the eight tools we examined have an implementation of the “Mapping of Requirements” technique. The differences between the tools are the following. On one hand, the properties and interdependencies with other elements that remain mapped are not the same in all five tools. Jira does not allow propagating changes in between mapped requirements, yet a specialized plug-in exists for implementing this feature. In Caliber, only the description of the requirement can be mapped. Jama allows also to map any property value of a requirement. Another difference among the tools is in how the changes in the base requirement affect the mapped requirements. In the case of Caliber, the changes are automatically applied. For instance in Jama and Polarion, the user has to accept to synchronize the mapped requirement. Jama allows to define reuse rules to state which parts of requirements must be mapped and when to apply the synchronization of mapped requirements when base requirements change.

The use of custom requirement templates is implemented only in DOORS and Modern Requirements 4TFS. In DOORS we were able to create such templates, but it was not possible to test if they can be shared with other projects and we did not find any document that clarified it. In the case of Modern Requirements 4TFS, it has the possibility to export FAQ. These FAQ are a library of questions, organized by subjects (e.g., security, portability, usability) that can be helpful during the elicitation of requirements. It is possible to associate to each question one or more reusable requirement. The problem we found in this functionality is that we could not find any initial FAQ template. Thus, a company using the tools should establish their own library of questions and reusable requirements.

Concerning the use of project templates, seven out of the eight tools have this possibility (all except Caliber). A project template defines the project properties and the types of requirements that the tool manages in order to specify templates suitable for different types of projects (Agile, CMMI, Traditional, etc.) In Modern Requirements 4TFS and TopTeam, it remained unclear if they allow to define new project templates. Finally, it is important to note that in Jama this functionality is provided by the duplication of a project, which can remain synchronized with the changes in the original project.

The concept of library of reusable requirements does not exist in any tool. However, in Modern Requirements4TFS, it can be simulated with the templates of questions and reusable requirements that can be exported and imported as templates allowing the reuse of requirements across different projects. Also in Jama, it can be simulated by the use of containers of reusable requirements, or by the use of project duplication, in the case of projects that group groups of related reusable requirements.

2.6.3 Interdependencies Extraction and Management

First, we define the techniques related with interdependencies extraction and management in the analyzed RMS:

- **Interdependencies Extraction**. RMSs in general allow extracting requirements from text documents in formats such as Word or Excel. However, interdependencies extraction means adding interdependencies between requirements from this kind of documents automatically. Although we have not found any tool that provides this functionality, we include this feature due to its relevance for the OpenReq project.
- **Interdependencies Definition.** When an interdependency is defined between two requirements, the type of interdependency can be stated. The types of interdependencies vary among RMS, and some RMS allow defining new interdependency types.
- **Interdependency Types Semantic Definition.** The terms used for dependency types differ. For the users of RMS, it can be confusing to specify interdependencies, if they do not understand the meaning of these terms and, thus, types. There are interdependencies types that are to be defined between requirements and others that are to be defined between requirements and other artifacts managed by the RMS. The semantic definition helps in applying correctly the interdependency types.
- **Interdependencies Traceability.** The requirement engineers using a RMS may be interested to know transitive interdependencies in which a requirement is involved in, or all interdependencies in a project. Typically, this is done by providing a traceability matrix or impact analysis grids.
- **Tagging of Suspect Interdependencies.** When a requirement involved in an interdependency changes, the type or meaning of the interdependency can be affected. An interdependency in these cases is tagged as suspect.

The summary of how the analyzed RMS cover the different techniques for the management and extractions of interdependencies can be found in the Appendix 2 *Full details of RMS comparison*. Next, we highlight the main differences.

Before addressing the techniques and how they are provided by the analyzed tools, it has to be noted that the tools do not use the wording “interdependencies”. This marks an important difference between research in requirements engineering and practice. Six of the eight tools call interdependencies “Links”. Caliber calls interdependencies “Traces” and Jama calls them “Relationships”.

Concerning interdependencies extraction from text requirements, none of the eight tools has this functionality. Instead, all RMS allow defining interdependencies. There are multiple types of interdependencies and the types vary between the different RMSs. In general, the RMSs allow creating new interdependency types. However, in Caliber, that just allows to create Parent/Child and Traces interdependencies it is not possible to create new interdependency types and in the case of Modern Requirements 4TFS and Polarion we could not determine from the demo version and documentation if it is possible.

Related to the specification of interdependencies, one problem is the semantics of the predefined interdependency types. Only Helix RM shows the definition of the interdependency types to the user when specifying an interdependency using the type. In the case of DOORS, the definition exists but it is presented only in the interface provided for the administration of interdependency types. Finally, in the case of TotTeam, it was possible to add a definition for a new interdependency type, but it seems that predefined types do not have such definition. In the other tools, we could not find any definition for the interdependency types. Although some interdependency types should be defined between specific requirements types, e.g., between non-functional and functional requirements, the tools do not enforce it. Only DOORS and TotTeam allow to define constraint rules that once defined the tools enforce.

Concerning traceability, all the eight analyzed tools provide one or more types of traceability matrices and an impact analysis grid. Jira does not provide this by default, yet these functionalities can be achieved by installing separate plugins. Functionalities of these tools vary in between implementations. When the RMS present interdependency matrices, they show qualifications in the interdependencies, such as the direction of the interdependency. Also all the tools allow to know the interdependencies where one requirement is involved. This visualization can be done when the requirement is edited or when the requirement is inspected jointly with the rest or requirements in a tree or grid view of requirements (in this case interdependencies are seen tagged as an icon at the side or the requirement identifier).

Interdependencies may become suspicious when one of the requirements involved in an interdependency changes. Jira allows sending notifications whenever a requirement changes, which allows for tracking also interdependency changes. There are two tools, DOORS and ToTeam, where it has to be configured if the user wants that this qualification is automatically tagged once the change occurs. In Caliber and Jama, the tagging is automatic. In the case of Helix RM and also in the other tools mentioned before, it is possible for the user to define an interdependency as suspicious manually. In the case of Modern Requirements 4TFS, we did not find in the demo version or documentation about the possibility of tagging suspicious links.

An additional issue with suspicious interdependencies is that as they can appear in a transitive closure of interdependencies - when one interdependency becomes suspicious, it could be necessary to also tag other indirect interdependencies. In tools as DOORS, Polarion and TopTeam we have found a mechanism to analyze and apply recursively the suspect tag in interdependencies.

2.6.4 Support Techniques - Additional functionalities

In this subsection, we group additional functionalities in the selected RMS that help in the definition of requirements. We define these functionalities as:

- **Glossary and natural language analysis support.** There are RMS that allow to manage a glossary of terms, which is used to help in the definition of requirements. Specifically, there are RMSs that highlight the terms in the glossary when requirements are defined and show their definitions by selecting the word. In our analysis, we also looked for functionalities to help in the analysis of natural language description of requirements.
- **Recommendation of requirements.** Recommendation of requirements consists of recommending requirements to the users of a RMS that could be interesting to be included in the requirements specification of a project. Although we have not found any tool that provide this functionality, we include this feature due to the relevance for OpenReq project.
- **Search.** A RMS usually allows to search requirements that contain certain words in their description or on the basis of requirement properties.
- **Links to other Requirements in Requirement properties.** There exist tools that allow to add links to requirements from requirement properties, usually in the description of the requirement. These links are not the typical interdependencies addressed in the subsection above, but links that allow to quickly navigate in the tool interface from one requirement definition to another one, which is related some way.
- **Dashboards.** Dashboards show project information about the current status and progress of a project with respect to the requirements defined for it. The information provided may be requirements satisfied, requirements pending, values of specific metrics on the project, requirements assigned to a user and recent changes in the project.
- **Customization.** Many RMS allow building extensions to accommodate versatile needs. Here, manufacturers can offer open Application Programmer Interfaces (API) or Software Development Kits (SDK) to encourage customization.
- **Integration.** Requirements management tools can be integrated with other tools that are used in enabling the software development environment workflow. These can include issue trackers, version control-, code review- and continuous integration tools.
- **ReqIF format.** ReqIF is an XML file format that can be used to exchange requirements, along with its associated metadata, between software tools from different vendors. The requirements exchange format also defines a workflow for transmitting the status of requirements between partners.

The summary of how the analyzed RMS cover the additional functionalities can be found in the Appendix 2 *Full details of RMS comparison*. Next, we highlight the main differences.

Only three of the eight tools provide glossaries for specific projects, but no-one does the natural language analysis of requirement description. Caliber allows to do a certain analysis of requirements description regarding the glossary, and highlighting in text ambiguous terms and words in the glossary. That is, when requirements are written, words are automatically colored to indicate use of a glossary term or alert of an ambiguous term that you should replace with a better-defined term. DOORS stand out because it is possible to create interdependencies between requirement parts and terms in the glossary. TopTeam suggests words from the glossary to be included in requirements definition.

It is important to remark that we did not find any requirements recommendation functionality in any of the eight tools we analyzed. The tool that is closest to offer this functionality is Modern Requirements 4TFS: FAQ questions and reusable requirements associated could be used as a repository for recommendations (see Requirement Templates above).

Searches of requirements are possible in all the eight tools. The differences are in the possibility of saving queries, that is clearly possible in Caliber, Jira and Modern Requirement 4TFS. Another difference is in the possibility to add relevant tags to requirements that facilitate searches. This is possible in DOORS, JIRA and Modern Requirements 4TFS. Finally it has to be noted that in the case of Jira, Modern Requirements 4TFS, it is possible to define global queries that are shared among all users assigned to a project or personal queries. The rest of the tools have searches that allow to filter requirements that contain certain terms in the requirements body text or some of their attributes.

The possibility of adding links to a requirement in the definition of another requirement is possible in DOORS, Jira, Polarion and TopTeam.

Dashboards with a summary of a project exists in all the RMSs except in Caliber. All RMSs allow configuring the dashboards, and to include or exclude widgets with the contents that is interesting for the particular user and project.

Helix RM, Jama, Jira, Polarion and TopTeam offer APIs for accessing the requirements data. Only Caliber, DOORS, Jira and Polarion offer a SDK that allows building own extensions to the software itself. In the case of Modert Requirements4TFS, we have not found evidence of any type of customization.

Plugins built for Jira offer integration with most common software development- and project management tools. Helix RM, Jama and Polarion can be integrated with some systems, including Jira. Modern Requirement 4TFS only can be integrated with other products of the same provider. DOORS support integration by means of Open Services for Lifecycle Collaboration that facilitates a wide variety of integrations. Finally, Caliber provides traceability from requirements to artifacts managed by other requirement management systems as HP Quality Center.

Finally, only Polarion and Doors offer support the ReqIF format for importing or exporting requirements by default. Plugins are available for enabling this feature are available for Jira.

2.7 Software product lines, variability and knowledge-based configuration

As the final topic of this state of the art, we describe the field of *Software Product Lines* (SPLs) from where we bring systematic reuse and modeling techniques. SPL engineering covers *variability* and *variability modeling* as its subtopics. As a related topic, we utilize *knowledge-based configuration* that is a more general-purpose approach applicable to SPLs and beyond.

2.7.1 Software product line, reuse, and variability

To enable the commonality and diversity in software to be addressed, SPLs, which are also synonymously called software product families, have become an established software engineering practice, as evidenced by several textbooks (Weiss1999, Bosch2000, Clements2001, Pohl2005) although the notion of software product line dates all the way back to 1960's. The products of an SPL can be any kinds of software systems, such as embedded systems, software products, or digital services. Although what an SPL is not unambiguous, a definition of an SPL is as follows: “a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way” (Clements2001). Another definition of an SPL is the following: “A software product line consists of a product line architecture, a set of reusable components and a set of products derived from the shared assets” (Bosch2000).

As a set of products define an SPL, the similarities within and differences between these products are essential to the characterization of an SPL and are typically associated with commonality and variability, respectively. Commonalities are often realized through reuse, so that the same artifact is used for different products in the same manner.

Variability is defined as "the ability of a software system or an artifact to be efficiently extended, changed, customized or configured for use in a particular context" (Svahnberg2005). The different products of an SPL are manifestations of variability, where variability is taken advantage of because of the differences it offers. The activity of realizing differences, as a part of application engineering, is in general referred as resolving variability, resulting in different variants. To address the phenomenon of variability, software variability management is a key activity within SPL engineering, thus forming a key characteristic distinguishing it from the development of a single system. Here, variability management means general activity to cover all activities or concerns of variability, from the identification of variability, through realization, to maintenance. Variability management is synonymous, e.g., with variability handling (Galster2014).

Within the broader context of variability management, a variability model is a means and an artifact to represent the variability of software to the stakeholders. The stakeholders include internal stakeholders, such as software developers who develop the software and variability therein, managers who make the decision about the software that is developed, sales representative who are at the interface with customers, as well as external stakeholders who make the decision to buy the software and users who use the software but can also be in the role of a customer. Variability models cover any kinds of representations about variability. Broadly, variability models range from informal sketches or informal natural language specifications to various kinds of models using textual or graphical notations based on rigorously defined syntax and semantics (Raatikainen2017). We use the term variability model to refer to any model that contains unresolved variability, thus being developed during the domain engineering phase and forming an input to the application engineering phase, when variability is resolved.

Most probably, the best known example of a variability model is a *feature model* (Kang1990, Kang1998), which emerged in the 1990s, although much of the research seems to focus on the emergence of SPL research in the 2000s (Benavides2010, Hubaux2010). An example of a simple feature model as by a feature diagram of a mobile phone is shown in Figure 1. A feature model conveys many of the key concepts of variability modeling. A feature in a feature model can be seen as a characteristic of a system that is visible to the end user (Kang1990) or, more generally, a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate among product variants (Czarnecki2005). A feature model is typically represented as a graphical diagram arranged as a set of features and relations between a parent (or compound) feature and its child features (or subfeatures) and cross-hierarchy constraints (Benavides2010). The relationships represent variabilities such as optionality, meaning that a feature can be either selected or left out, or alternatives, meaning that one of the alternatives needs to be selected.

However, several additional concepts, adaptations, and extension have been proposed (Tiihonen2016, Raatikainen2017). There are also several formalizations of feature model (Schobbens2007). Specifically in OpenReq, we take advantages of Kumbang feature model conceptualization (Asikainen2006) as the basis for a feature model. Kumbang specifies the subfeatures of a feature tree as "part-of" relationships and allows defining separate is-a hierarchies. The Kumbang constraint language can be used to express cross-branch relationships. Kumbang also supports feature attributes. Finally, a feature model is provided with numerous different analyses (Benavides2010).

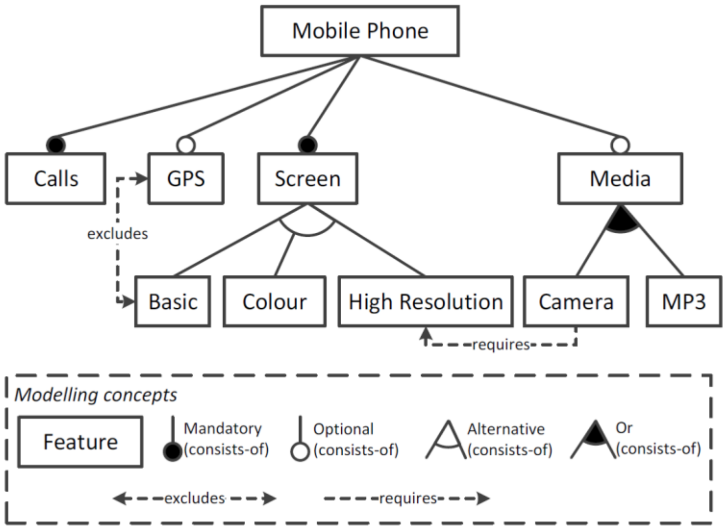


Figure 1. An example of a feature model based on the extensions of a feature model (Benavides2010) in (Myllärniemi2014)

2.7.2 Knowledge-based configuration (KBC)

Knowledge Based Configuration (KBC) emerged from various domains of physical products, such as computers and elevators. Model-based approaches to configuration knowledge representation provide a clear separation between domain knowledge and corresponding problem solving knowledge (Tiihonen2017). Hence, they avoid intermingling of both knowledge types. This intermingling has been shown to significantly increase related development and maintenance efforts (Soloway1987).

KBC is a relatively general and domain-independent approach. One hand, a large number of commercial general-purpose knowledge-based configurators, also known as configuration frameworks exist. For example, already in 2005, 30 vendors were identified based on their Web pages (Anderson2005). Gartner Group estimated in 2013 that Configure, Price and Quote application vendors generated $300M in revenue in 2012 (Sengar2013). On the other hand, an active KBC research community has been established: the Configuration Workshop series was arranged in 1996 and yearly since 1999 it has been the primary meeting venue. Other important forums are journal special issues (Darr1998, Faltings1998, Soininen2003, Sinz2007, Tiihonen2010a, Felfernig2011).

Technically, during the product development process of configurable products (cf., domain engineering), KBC captures variability, or configuration, knowledge systematically to a knowledge base referred to as a *configuration model* that a tool can utilize for product specification during the configuration task (cf. application engineering). Resolving variability in the configuration task utilizes a pre-designed architecture and components; no manual work such as the design of new assets (e.g., components) is required (e.g. Tiihonen1997a, Sabin1998).

Over the years, three trends in modeling variability in KBC can be identified. First, configuration knowledge can be directly represented as constraint satisfaction problems, production rules, logic programs, etc. representation mechanisms of the applied problem-solving method (Stumptner1997, Sabin1998). Second, individual central phenomena for modeling configurable products have been identified and conceptualized. Here, components are the building blocks of products in the sense that products (product individuals) consist of components (component individuals). Well-known approaches include connection-oriented (Mittal1989), resource-based (Heinrich1991), structure-based (Cunis1989), and function-based (Najmann1992) approaches.

Third, the unified approaches combine the earlier ideas into a covering ontology or conceptualization. Two widely cited and fundamentally similar conceptualizations are those of Soininen et al. and Felfernig et al. In the conceptualization of Soininen et al. (Soininen1998), configuration model knowledge specifies the entities that can appear in a configuration, their properties as attribute-value pairs, and the rules on how the entities and their properties can be combined. Individuals (instances) of configuration model concepts describe individual configurations and, thus, represent configuration solution knowledge. Finally, requirements knowledge specifies the systematized requirements on the configuration to be constructed. The conceptualization covers connection-, resource-, structure-, and function-based approaches and defines corresponding main concept types that are furthermore organized in a classification hierarchy. Constraints are an important part of the conceptualization. While the Soininen et al. conceptualization is general, the conceptualization of Felfernig et al. (Felfernig2000a, Felfernig2001) is provided with an operationalization as UML stereotypes and Object Constraint Language (OCL).

Many commercial and research configuration frameworks utilize Constraint Programming, i.e. they represent the problem to be solved declaratively as a *Constraint Satisfaction Problem (CSP)*. A constraint satisfaction problem is a tuple (V, D, C). Here, V is a set of finite domain variables, $V = \{v_0, v_1, \ldots, v_n\}$. Each variable has a (usually finite) domain that specifies the possible values of the variable, and the set of domains is D, $D = \{dom_0, dom_1, \ldots, dom_n\}$. C is a set of constraints specifying restrictions on the allowed combinations of variable value assignments. A solution to a constraint satisfaction problem is a set of assignments to each variable $\{v_0 = x_0, v_1 = x_1, \ldots, v_n = x_n\}$ such that each $x_i \in dom_i$ and the assignments are consistent with the set of constraints C (Mackworth1985).

Answer set programming (ASP) makes it possible to express the problem as a theory consisting of logic program rules with clear declarative semantics, and the stable models, i.e., the answer sets of the theory correspond to the solutions to the problem (Simons2002). Programs that follow the ASP paradigm are a generalization of normal logic programs. A generalized and unified syntax of ASP programs called ASP-Core-2 has been defined (Calimeri2012) and adopted by many ASP solvers. Optimality criteria, variables and built-in functions can be defined. The syntax of ASP programs is close to Prolog, but the computation method via model generation is different (Gebser2011).

More recently, KBC has evolved to include more advanced topics, such as the following ones relevant for OpenReq:

- Interactive scenarios, especially when human user is involved, require fast response times. Sometimes reasoning engines that are based on compiled knowledge representations, such as binary decision diagrams (BDDs) (Andersen et al. 2010), are applied to more efficiently determine if a solution exists or if specific selections are still possible.
- When no solution exists for a given set of customer requirements, conflict detection (Junker2004) and diagnosis approaches (Bakker1993, Felfernig2012) can be applied to assist users in resolving the conflicting requirements and to find a suitable solution. In this context, recommendation technologies can also be applied to determine personalized diagnoses in situations where no solution meets the preferences that the user specified (Felfernig2009).
- Existing configuration environments usually support a single user configuring a product. Support for group-based configuration where groups of users make the decisions has been identified as an emerging topic (Tiihonen2017). One example scenario is software release planning: a group of stakeholders has to decide on which requirements are implemented in which software release.

3 Concepts and technologies

In this section, we introduce the concepts and technologies that will be adopted in the OpenReq approach for requirements knowledge and dependency management work package and form the basis for software architecture and services that realize the approach.

3.1 The concept of a requirement

The concept of a requirement is in general somewhat ambiguous and we adopt a relatively general notion of a requirement rather than limiting to a specific definition of a requirement. We consider each individual requirement to be a single entity. A requirement is further characterized as *roadmappable*, meaning that a requirement is an entity that is decided to be implemented at some point of time, e.g., in a specific release, or disregarded. For clarity, we presume that each requirement has a separate unique identification (id) and content. The content describes the requirement in human-understandable form, often textually. Each individual requirement is presumed to be unique for a project or product under development.

A requirement type can define a set of named *properties* with *values*. Archetypical properties include priority, planned release, effort or assignee. In particular, a requirement has at least implicitly a status as a property that indicates whether the requirement has been implemented. However, the actual property types are context dependent and no single property is applicable to all contexts. Therefore, we allow any set of properties and do not require any specific property.

It is possible to structure individual requirements hierarchically. A more general requirement can consist of a set of more detailed requirements that we refer to as a *part-of hierarchy*. For example, an epic can consist of a set of user stories, which is a part-of relation. In addition, a requirement can have interdependencies beyond hierarchical part-of relation to other requirements such as *depend on* or *conflict with* (cf. the section about interdependencies above).

We aim to include relevant interdependency types from the existing literature. In addition to aforementioned hierarchy and requires interdependencies, similarity, the “increases values of” and “decreases value of” or the “increases cost of” and “decreases cost of” relationships seem to be relevant to be included. Overall, the approach of Dependency engine aims to be flexible to add new interdependency types but we also aim to keep the conceptualization simple and not to introduce concepts that have not been provided with practical utility. For example, interdependency types should be relevant in the OpenReq trials.

It is also possible to define interdependencies between properties of requirements especially for the purposes of release management. For example, two requirements should not be included in the same release and the interdependency is then defined between the release-properties between these requirements: the release property value of the two requirements should not have the same value. These interdependencies can be characterized more generally as constraints because they are not interdependencies between two requirements but depend also on the property values of requirements. The exact set of such constraints will be defined over the course of the project, but the constraints below are considered as a tentative set:

- A property value is fixed. For example, release must be "1".

- Larger or smaller than N. For example, requirement A must not be in an earlier release (smaller) than release "3" -- i.e. the release property must be 3 or larger.
- Requires (and can be in same release). For example, Requirement A requires Requirement B to be in the same or earlier release so that release property value of requirement A must be the same or smaller than the value of requirement B.
- Requires before. Same as above requires but cannot be in the same release but needs to be in the earlier release.
- Excludes local. For example, if requirement A is in release N (has value N for release-property), requirement B must not be in release N (must not have value N for release property).
- Excludes global ("not at all"). For example, if requirement A is included to any release, requirement B must not be in any release.
- Same release (multiple requirements). Requirements A,…,N must be in the same release.
- Sum of property values smaller/larger. For example, the sum of effort-property values of requirements that have release-property value “1” must not be larger than 100.

Dependency engine operates primarily with, and in the context defined by the state-of-the-practice large-scale requirements management systems (RMSs). An example of dedicated RMS is Doors but issues trackers, such as Jira, are also used especially in large-scale open source projects. Such RMS document and manage the requirements of a system under development. Essentially, a requirement in the RMS consists of similarly as described above of a unique ID; a phrase, figure or something to describe its content; properties or meta-data; and interdependencies to other requirements as a special class of properties. However, we are not restricted to any RMS but we are compliant with any similar, structured data source such as a dedicated database of OpenReq or structured messages.

Specifically, we aim to be compliant and adopt the basic idea of the ReqIF standard that has synthesized a widely accepted view of what forms a requirement, and seems to be compliant with, e.g., the Jira issue tracker. More specifically, individual requirements are instances of requirement types that correspond in ReqIF to SpecObject and SpecType, respectively. The requirement properties correspond to the *AttributeDefinition* and *AttributeValue* of ReqIF. Again, both part-of and interdependencies are compliant with ReqIF—binary interdependencies between individual requirements can be instantiated from relationship types, by *SpecRelation* and *SpecRelationType* of ReqIF and *SpecHierarchy* allows hierarchical structuring of *SpecObjects*.

3.2 Representing requirements by a feature model

The internal approach for Dependency engine is based on using requirements data stored in a RMS and using internally for Dependency engine the constructs of a feature model and its formalizations to represent this data[2]. A RMS focuses on managing each individual requirement, including their properties and relationships. However, the entire system is defined by a set of requirements as roadmappable entities with properties, and relationships to each other that constitute a model. Such a model is similar to a feature model. The rationale for constructing a

[2] For a concrete example, see D5.2: Requirements Dependency Engine Version 1

feature model is that a feature model is well-researched approach that is provided with various kinds of analysis as well as existing analysis and inference tooling.

In order to represent requirements by a feature model, we make each requirement correspond to exactly one feature of a feature model. The properties of a requirement correspond respectively to the attributes of a feature. For example, a requirement “A” with a property “release” and value “1” becomes a feature “A” with the attribute-value pair “release” and “1” in the feature model. The part-of (or consist of) relationships of requirements constitute the tree hierarchy, thus corresponding to subfeature-relationships. For example, when an epic requirement has user stories as its sub-requirements, the feature model representation will result in a model fragment where the epic is the parent feature and the user stories are its child features by part-of relationships. We apply part-of (or refines) rather than is-a relationships for hierarchy since the latter ones seem not to be common between requirements although could be supported. In order to construct the tree, a generic root is defined that, in practice, is the application or project itself, which has then all requirements underneath.

Different types of requirements can be represented by the concept of feature subtyping. For example, user requirement and technical requirement can be different subtypes. Each subtypes defines, e.g., its properties (or attributes).

Other relationships among requirements correspond to the cross-branch constraints. For example, *requires* interdependency can be expressed by “present” constraint that states that the presence of requirement “A” requires the presence of requirement “B”. Likewise, the constraints can be defined between the attributes of features in the feature model. In general, feature models allow expressing different constraints that depends on the selected feature model dialect and these constraints can be adapted then for the purposes of OpenReq.[3]

The resulting feature model is then a variability model. Variability denotes the status of requirements -- each new requirement becomes at first optional (cardinality 0-1) for the model. Then, various analyses can be carried out by selecting requirements to be included in a configuration. A configuration denotes requirements that are ready at a certain point of time and, thus, corresponds to a planned or an existing release. That is, a release management includes therefore a configuration problem: For a selected set of requirements, find other requirements that need to be taken into account because of the interdependencies (or other constraints). Details of interdependencies and constraints are elaborated above but, for example, if all requirements are set a target release and effort, for each release it can be analyzed if the effort is within the limits of available total effort for a release; are all interdependencies adhered to such as required requirement is not unintentionally in a later release or locally excluded requirements are in different releases. It is also possible to suggest repairs if something is not consistent such as changing target release of requirements.

[3] We use and possibly extend Kumbang constraint language http://www.soberit.hut.fi/KumbangTools/language/constraint-language.txt

3.3 Formalization of requirements by an ontology

Dependency engine will be based on providing requirements with formalization. For the formalization, we will develop an ontology that is compliant with ReqIF. The ontology will also be provided with a mapping to the Kumbang feature model conceptualization and language (Asikainen2007), which has already been provided with a formal semantics. The Kumbang concepts are also used as an internal representation for the requirements and not exposed to any external service. Kumbang seems to be able to represent all concepts that are represented in a RMS as detailed above. For example, properties can be defined freely and a relatively rich constraint language exists. Additional details about Kumbang can be found in the external link to KumbangTools webpage (http://www.soberit.hut.fi/KumbangTools/).

The ontology focuses on the properties, structure and interdependencies. Figure 2 shows a preliminary ontology that will be refined and finalized during the project (Task 5.4 and Deliverable D5.3 of OpenReq). We adhere roughly to the ReqIF standard although we simplify something but also give more specific and concrete definitions. The project part of the ontology that could include release, stakeholder and other concepts that, however, are not yet defined, because they are not at the core of this work.

- **Requirement** is an identifiable entity as described by its content that is an inherited characteristic from *RequirementType*. Content can be text, figure or anything else of type *Object*. Each requirement has a unique identification that is string. Content could identify a requirement by itself but we include identification for clarity.
- **RequirementType** defines the characteristic of a requirement. For example, Epics and user stories can be different *RequirementTypes*.
- **PartDefinition** is a placeholder that defines a place in the hierarchical structure so that sub-requirements (or child-requirements) can be attached to any (parent) requirement.
    - Sub-requirement is one kind of interdependency.
    - Roughly, a *PartDefinition* is an explicit connector for structure.
    - Cardinality is a pair of integers [n,m] that state the number of sub-requirements needed for the place in minimum and allowed in maximum.
    - Cardinality assumes that all requirements are different.
    - For example [1,1] means that one thing must be there in order for the (parent) requirement to be meaningful; [0,1] means that the part or sub-requirement is an optional part. If in a placeholder, there are two sub-requirements, [1,1] means that either one (but not both) needs to be as a part; and [1,2] means that either one or both must be there.
- **RelationshipType** is another interdependence type than those defined by *PartDefinition* such as requires or conflicts. The *RelationshipType* is a binary relation between two requirements. The relationships can have properties such as whether a “requires” is a soft or a hard constraint for the system. The ontology defines this in a general manner but a reference catalogue of *RelationshipTypes* will be provided.
- **Constraint** is an additional concept that can express more complicated interdependencies as described above.
- **AttributeDefinition** is any kind of property attached to a *requirementType*. For example, priority or release is an *AttributeDefinition*.

- **AttributeValueType** defines roughly the value type for a property such as integer for priority. We do not elaborate property typing further in this part of the ontology. However, types such as enumerations, a set of primitive types, etc. could be defined.
- **Project** denotes here generally some additional concepts that are needed but are not currently in the core of this work or generally structural considerations.

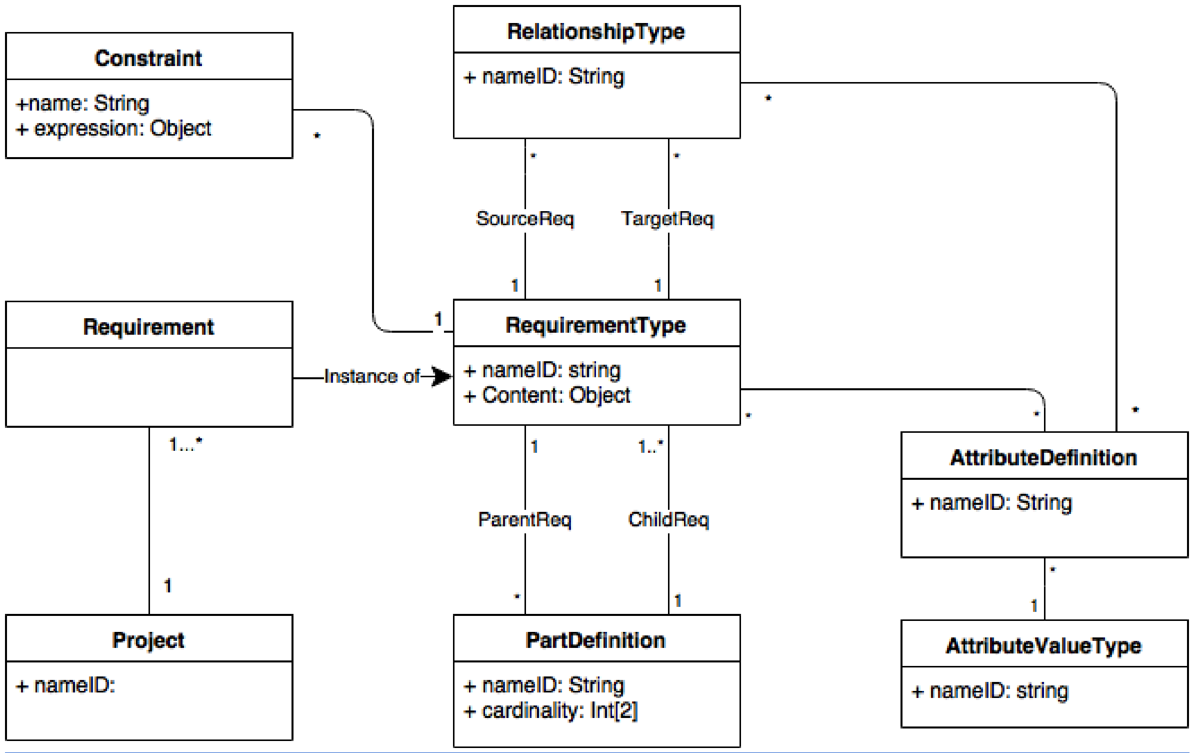


Figure 2. A preliminary ontology to represent requirements

3.4 Interdependency detection

Interdependency detection will combine the identification of explicit and non-explicit interdependencies in the requirements. By explicit interdependencies, we mean explicit references in the requirement to other requirements, while by non-explicit interdependencies we mean those ones that are not explicitly stated in the requirements but that can be identified by analyzing the requirements both from a syntactic and semantic point of view.

For the detection of explicit interdependencies, we aim at following the approaches used in the well-known area of cross-references detection and resolution. In the first approach, we will identify natural language patterns that are used in requirements text to refer to other requirements and use them to create new interdependencies. This approach will be based on works such as those of (Breaux2008) and (Palmirani2003). In the future, though, we will consider the possibility to extract these patterns automatically as done in (Sannier2017).

Regarding the detection of non-explicit interdependencies, first we will identify pairs of similar requirements. As explained in the *Similarity detection* subsection of the State of the art and practice, identifying similar requirements could be used as a basis to identify related requirements.

As there are several well-known components already developed to detect similar texts in English, our aim is to select one of these components to be integrated and expanded in OpenReq. The components we are currently evaluating are:

- **Semilar** (http://www.semanticsimilarity.org/). It provides an application and an API to identify similar requirements using different algorithms. The methods offered by Semilar, which are completely parametrizable, range from simple lexical overlap methods to methods that rely on word-to-word similarity metrics to more sophisticated fully unsupervised methods that derive the meaning of words and sentences such as LSA and LDA to kernel-based methods for assessing similarity. In addition, one can select the tokenizer, tagger, stemmer and parser to be used as pre-processing (having as options, for instance, the libraries OpenNLP, Stanford parser and WordNet).
- **Gensim** (https://radimrehurek.com/gensim/tutorial.html) (GNU LGPLv2.1 license). It provides an API, which includes implementations for popular algorithms such as LSA, LDA and RP. Gensim allows loading a corpus of texts to which a sentence can be compared. The calls to the algorithms are parametrizable.
- **Scikit-learn** (http://scikit-learn.org/stable/documentation.html) (BSD license). Its API allows the transformation of texts into vectors, using TF-IDF[4] among other algorithms, and measure the similarity over them using the Cosine measure.
- **Cortical** (http://cortical.io/). Among its functionalities, it has an API that provides the Cosine similarity between two given texts. In this case, the method is not parametrizable.

However, we do not discard to add new components to this list if the results of our evaluations are not good enough.

The second part of non-explicit interdependencies section is to improve and expand the requirements similarity detection with further features, such as:

- Creating a specific list of synonyms that are domain dependent, so that the similarity algorithms can know when two words that in principle are not synonyms are actually synonyms in a specific domain.
- Constructing models that can help to detect interdependencies by relating concepts on this mode. This is similar to the ontology used in (Zhu2005), but in our model, relationships will broaden the scope of the previous work, which is focusing on inconsistencies. For instance, if we know that technologies A and B are incompatible, A and B will be related in this model as conflicting. Therefore, when these two technologies are using at the same time in a single project, we can extrapolate that the requirements stating technologies A and B are actually conflicting and a new interdependency of this type will be created among them.

[4] TF-IDF (standing for Term Frequency - Inverse Document Frequency) computes the weight of the words in a vector as the multiplication of TF and IDF with the aim of achieving the benefits from both:

- TF assigns a weight proportional to the frequency of the term occurrences in the given text fragments.
- IDF assigns a weight depending on the number of given texts that include the term (rated to the total number of texts).

It is important to highlight here the fact that OpenReq project will deal not only with English language, but also with Italian and German, since the telecom trial deal with text written in the Italian language and, for the case of Siemens, with text (partially) written in the German language. This diversity of languages used to write the text analyzed supposes a challenge for the project. The majority of the existing NLP approaches target the English language, as they are trained and validated using English text corpora. Although NLP approaches and software libraries exist for both languages (Basili2015) (Rehbein2012), their performances (e.g., precision) might be inferior compared to the well-established, English-based ones.

3.5 Requirements reuse via requirement patterns

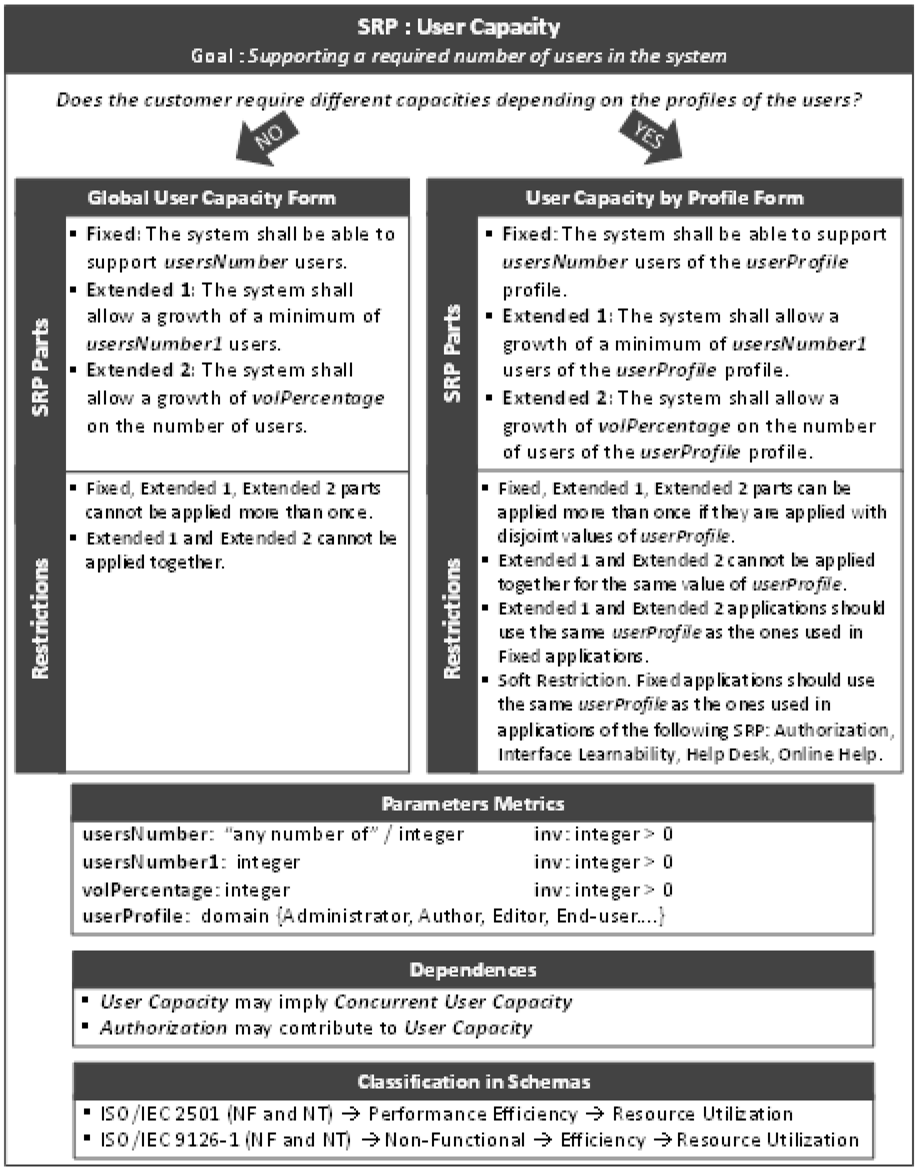
To achieve requirements reuse, we will adopt the PABRE framework (Franch2013, Renault2009), which stands for PAtterns-Based Requirement Elicitation, to OpenReq. The core asset in the PABRE framework are Software Requirement Patterns (SRP). An SRP is a set of requirements that pursue the same goal in a system to be developed, and where all specific elements of a certain project have been eliminated and converted into templates. Aside from these templates, the SRP has other attributes to guide the application of the pattern (e.g., name of the pattern, goal and keywords). Appendix 3 contains the metamodel of SRPs. Here, to facilitate the understanding of the structure and use of SRP, we present them through an example, the *User Capacity* pattern (see Figure 3), that illustrates the structure of patterns, their attributes and relationships allowed among them.

An SRP is a pattern that, when applied, produces software requirements related to the objective (goal) of that pattern. Applying the *User Capacity* SRP produces requirements related to the goal of S*upporting a required number of users in the system* under development.

A goal can be achieved in different ways. An SRP consists of several Forms, each one representing a different solution for achieving the goal. In our example SRP, its goal can be attained by defining the user capacity depending on the user profiles (*User Capacity by Profile* form), or by defining the global capacity of users, i.e. without taking into account the different types of users in the system (*Global User Capacity* form).

Forms are organized into Parts, each of them being a phrase template: a Fixed Part, which is always applied if the form is chosen, and some Extended Parts, which may be applied or not. Extended parts are only used if more precise requirements are required in the specification. For instance, in our example, the fixed part of the first Form is *The system shall be able to support %usersNumber% users* (*usersNumber* will be substituted in applying the SRP by “any number of” or by an integer greater than 0). The extended parts allow to specify the growth in number of users of the system. The first one states the growth by amount, whilst the second one states the growth by percentage.

Both fixed and extended parts are similar from a syntactic point of view. They are composed by a phrase template, i.e. the text to be used as a requirement and, if necessary, some optional Parameters to be instantiated when applying the pattern, e.g. *usersNumber* in the example above. Parameters have established their Metric, and eventually a correctness condition Inv (see these on the bottom of Figure 3) to define the values they may take.


SRP : User Capacity

Goal : *Supporting a required number of users in the system*

Does the customer require different capacities depending on the profiles of the users?

NO

Global User Capacity Form

SRP Parts

- Fixed: The system shall be able to support *usersNumber* users.
- Extended 1: The system shall allow a growth of a minimum of *usersNumber1* users.
- Extended 2: The system shall allow a growth of *volPercentage* on the number of users.

Restrictions

- Fixed, Extended 1, Extended 2 parts cannot be applied more than once.
- Extended 1 and Extended 2 cannot be applied together.

YES

User Capacity by Profile Form

SRP Parts

- Fixed: The system shall be able to support *usersNumber* users of the *userProfile* profile.
- Extended 1: The system shall allow a growth of a minimum of *usersNumber1* users of the *userProfile* profile.
- Extended 2: The system shall allow a growth of *volPercentage* on the number of users of the *userProfile* profile.

Restrictions

- Fixed, Extended 1, Extended 2 parts can be applied more than once if they are applied with disjoint values of *userProfile*.
- Extended 1 and Extended 2 cannot be applied together for the same value of *userProfile*.
- Extended 1 and Extended 2 applications should use the same *userProfile* as the ones used in Fixed applications.
- Soft Restriction. Fixed applications should use the same *userProfile* as the ones used in applications of the following SRP: Authorization, Interface Learnability, Help Desk, Online Help.

Parameters Metrics

usersNumber: "any number of" / integer	inv: integer > 0
usersNumber1: integer	inv: integer > 0
volPercentage: integer	inv: integer > 0
userProfile: domain {Administrator, Author, Editor, End-user....}	

Dependences

- *User Capacity* may imply *Concurrent User Capacity*
- *Authorization* may contribute to *User Capacity*

Classification in Schemas

- ISO/IEC 25010 (NF and NT) → Performance Efficiency → Resource Utilization
- ISO/IEC 9126-1 (NF and NT) → Non-Functional → Efficiency → Resource Utilization

Figure 3. Software requirement pattern example

Usually, fixed and extended parts must conform to some Restrictions for declaring multiplicities or dependencies among parts. In the *User Capacity* SRP, aside from restrictions on the possible number of appearances of each part in a specific SRS, there exist restrictions on the parameters' values in each application. For instance, in the second form of the example, the fixed part can be applied more than once in an SRS as long as the values assigned to the parameter *userProfile* are different. This allows to state restrictions on the user capacity of the system for different types of profiles, such as *Administrator* or *End-User*.

There is also another type of soft restriction that allows giving recommendations to maintain the consistency of the SRS. One example of such a restriction is using the same values for the *userProfile* in each application of SRP parts that uses this parameter (not only those ones of the example SRP, but also in its appearance in other SRPs such as *Authorization* and *Online Help*).

There exist dependencies among SRP in the same way as they exist among requirements. The example SRP is involved in two dependence relationships: the first one with the *Concurrent User Capacity* SRP, as there is a clear relationship among the number of users to support and the number of concurrent users to support; the second one with the *Authorization* SRP (provided the *User Capacity per Profile* form of the example SRP is used), since it allows defining the user roles of the system to develop.

Finally, SRPs are classified using Schemas, which are hierarchies of classifiers that facilitate the organization of SRP. It is possible to classify SRP following several schemas. The SRP in Figure 3 is classified according to two different classification schemas: ISO/IEC 25010 and its previous version ISO/IEC 9126-1. This classification makes the use of the catalogue easier according to both standard versions. These schemas also allow joining, in one classifier, SRP that may be applied as a group, and that address a same functionality or describe the same regulation required in the new system.

In OpenReq, we will adapt this structure of SRP to the needs of the project. For that, of course, we will need a catalogue of SRPs. The population of this catalogue will combine automatic extraction with expert assessment. We envisage three different ways in which this catalogue of patterns could be used in OpenReq:

1. *Browsing / Searching the SRP catalogue*. The browsing approach is based on the use of the SRP catalogue using one of the classification schemas of the catalogue and/or the relationships defined among SRPs in the catalogue. The browsing of the catalogue is optionally complemented by a search approach, which allows identifying the SRPs that have in their definition the terms used in the search.
2. *Propose SRP that are dependent.* Given a requirement, using a similarity algorithm it will be possible to recover similar SRP (by analyzing the templates in the SRPs with the given requirement). Then, once we know the similar SRPs to a given requirement, if they are dependent to other SRPs, we can propose these dependent SRP so they are reused. For instance, R_1 is similar to a template in the requirement pattern SRP_2, which is dependent on the requirement pattern SRP_3. In that case, SRP_3 could be proposed to be reused for R_1.
3. *Propose SRP the are related.* Again, given a requirement, using a similarity algorithm it will be possible to recover similar SRP (by analyzing the templates in the SRPs with the given requirement) (e.g., R_1 is similar to a template of SRP_1). Then, if SRP_1 is in classifier C_1 other SRPs in the same classifier could be proposed for reuse (e.g., SRP_2). So, for R_1 , SRP_2 will be proposed since they are both under the same classifier. A similar approach could be used for the keywords. If SRP_1 has keyword K_1, we can look in the catalogue for other SRPs that have keyword K_1 (e.g., SRP_3), and these SRPs could be proposed for reuse. So, in that case, for R_1 , SRP_3 will be proposed for reuse since they both are about keyword K_1.

3.6 Algorithms and technologies

The aforementioned mapping to feature model provides us with further existing mappings to representations by ASP and CSP. As for interdependency detection, the technology will be selected among the components presented in the previous section. Generally, the conceptualization and mappings to provide us with the formal semantics are in the core of our work and we apply, experiment and expand the algorithms and capabilities of existing technologies rather than develop entirely new ones. For example, Choco solver (see below) implements different algorithms that we can apply and experiment with.

The applied technologies will be based on commonly applied technologies in a micro-service architecture as well as existing research results. In more detail, the following technologies will be utilized

- Choco Solver (BSD license) www.choco-solver.org/
- KumbangTools for datamodels (BSD license version excluding Smodels) www.soberit.hut.fi/KumbangTools
- Configurator as a Service (CaaS) (BSD license) (Myllärniemi2012)
- One of the following (Semilar, Gensim, Scikit-learn, and Cortical) as the basis of the interdependency detection.

4 Dependency engine architecture

This section describes the planned software architecture for the software services that realize the requirements knowledge and dependency management approach. These services are collectively called *Dependency engine*.

4.1 Dependency engine context

Dependency engine operates in the context of logically two different stakeholders and systems. A view of the context is shown in Figure 4 below.

There are two kinds of stakeholder roles that indirectly interact with Dependency engine

- A ***requirements manager*** *(or engineer)* is responsible for eliciting, analyzing, and managing the requirements, including the properties and relationships of the requirements. In particular, the requirements engineer is responsible for the changes. This task is assisted by Personal recommender system (WP3) and Requirements intelligence system (WP2), and the requirements are stored in a RMS.
- A ***product manager*** makes decisions about what requirements in more general. The product manager is, in fact, a group of people or she interacts with a group of people. Therefore, the actions can include, e.g., voting or other negotiations about releases for which Group decision engine (WP4) is developed for. The product manager uses the existing RMS where the requirements are stored but can also have a dedicated own system.

Dependency engine has no direct (human) user interaction other than possibly for IT system administration related tasks. Therefore, there is no user interface but interfaces are REST-based calls between different systems and the calling systems shall contain the user interface.

Dependency engine operates in a context of other systems, namely a RMS and a release management system, or product management system in general as shown in Figure 4. The systems are logically different from the point of view of Dependency engine and, therefore, treated as two systems. In practice, for example Jira can used in the role of both of the systems. The release management in Jira is carried out by setting target release as the property for the requirements and dependency engine can then be used to check that no interdependencies are violated.

Figure 4 also depicts the two key task, one for both stakeholder roles, for which Dependency engine offers assistance.

- New interdependencies are extracted and existing interdependencies are checked during the requirements engineering, specifically during specification and analysis, phase. A model of requirements is constructed using feature model technologies.
- The validity check of release and possible repair proposals for invalid releases are carried out during the release definition and management. For example, Dependency engine can be used to check interdependencies such as if requirement A requires requirement B then requirement B is suggested to be included in the same or earlier release; or if the effort of selected requirements exceed the available resources, some requirements are proposed to be left out.

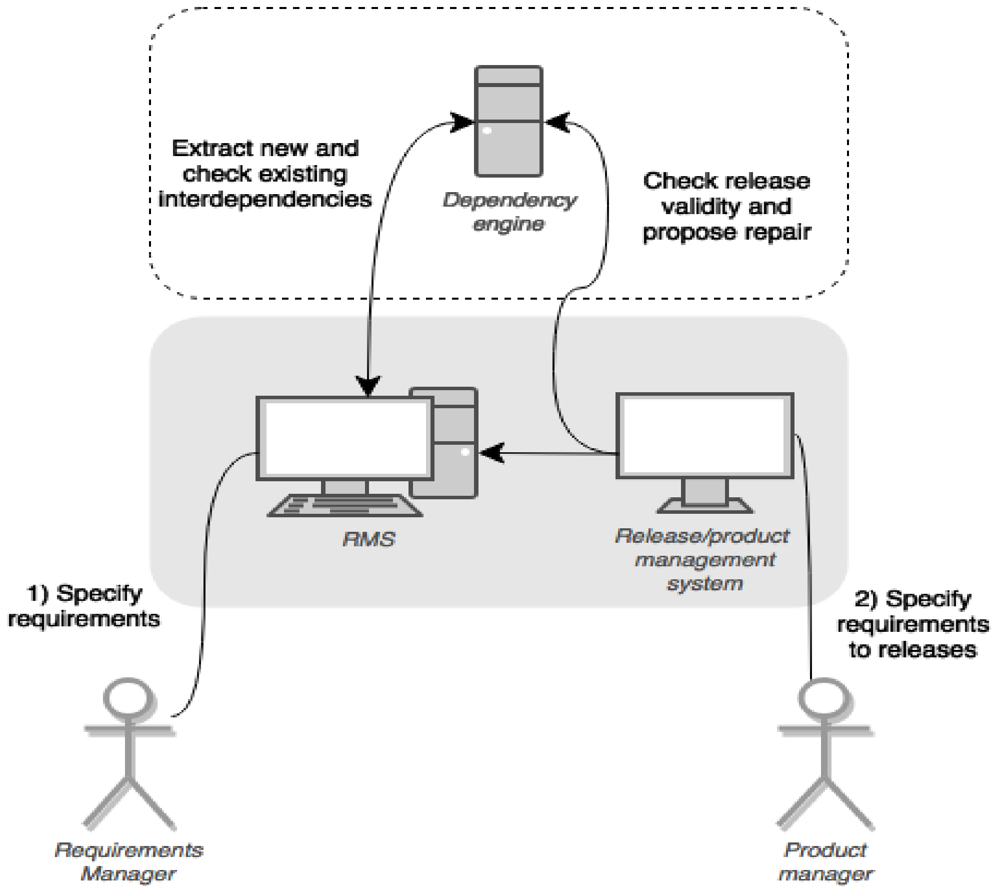

Extract new and check existing interdependencies
Dependency engine
Check release validity and propose repair
RMS
Release/product management system
1) Specify requirements
2) Specify requirements to releases
Requirements Manager
Product manager

Figure 4. The context of Dependency engine

4.2 Logical view of services in Dependency engine

Dependency engine consists of four independent service as shown in Figure 5. Each service has a REST-based interface and is relatively independent although operate in an orchestrated manner as a part of Dependency engine.

4.2.1 Services

Nikke is the service that contains the functionality and algorithm implementations for extracting new interdependencies from a set of existing requirements. In so doing, Nikke provides an external interface for uploading the requirements data that is then analyzed using NLP technologies and algorithms.

Milla is the service that initiates the functionality to construct the feature model representation of requirements, thus allowing the analysis of existing interdependencies and does the groundwork for release management purposes. In so doing, Milla provides an external interface for uploading requirements that are then converted to the proper format and forwarded to other services namely Mulperi and SpringCaaS. Milla has also ability to fetch the requirements from specific RMS.

Mulperi is a service that, one hand, takes care of the functionality of constructing a feature model representation from the requirements and, on the other hand, receives and converts queries about interdependencies in requirement to proper format. In general terms, Mulperi operates as a controller to SpringCaaS by converting incoming messages to proper feature model formats.

SpringCaaS (Spring boot Configurator as a Service) takes care of the required inference. It constructs an ASP or a CSP model, and uses existing solver (Smodels, Choco) as an inference engine to carry out required inference for the queries. For example, interdependencies are calculated as a transitive closure and repair is proposed if the selected requirements are in conflict.

4.2.2 Dependency engine external interfaces (APIs)

There are three external interfaces (application programming interfaces (APIs)) beyond system boundary.

- Nikke's external interface allows uploading the requirements for analysis. The requirements can be further forwarded to Milla
- Milla's external interface allows uploading requirements for constructing the feature model representation of requirements. Respective interface exists also for fetching requirements.
- Mulperi's external interface allows release management to make queries.

Milla's interface can be used from Nikke or directly from RMS. In the former case, requirements are forwarded to Milla after the analysis of new interdependencies. In the latter case, the requirements from RMS are uploaded directly bypassing Nikke, such as in case the requirements engineering has no time or authority to accept or reject Nikke's analysis results, or changes are so minor that Nikke's analysis is not considered necessary.

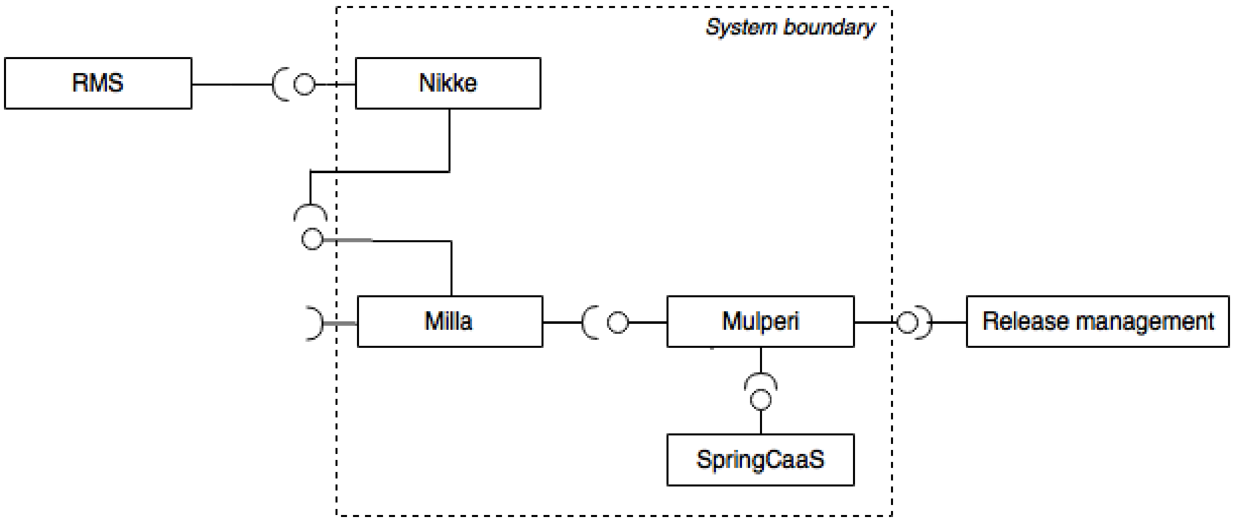


Figure 5. A logical view of services in Dependency engine and interfaces beyond system boundary

4.3 Behavior of Dependency engine

We describe the behavior of Dependency engine in Figure 6 by using the concrete scenario in which RMS and release management system are in Jira. Jira includes then a plugin that utilizes the services of Dependency engine by orchestrating and triggering the behavior. Because such a Jira plugin is developed in the WP6 or WP7 of OpenReq, the details of the Jira plugin are not covered here.

There are two different sequences of Dependency engine. First, Dependency engine helps the requirement manager in the management of requirements such as detection of similarities or duplicates, and detection that interdependencies are inconsistent. Internally for Dependency engine, a feature model representation of requirements is constructed. Second, Dependency engine helps the product manager in assigning requirements to different releases, such as ensures that the interdependencies are taken into account.

The upper sequence in Figure 6 describes the scenario of working with the requirements manager.

1. As a precondition for the sequence, the requirement manager stores requirements in Jira. She analyses the requirements in order to assure the quality of requirements including defining the interdependencies.
2. The requirements manager triggers the behavior of Dependency engine using the Jira plugin.
3. Jira as the RMS includes a Jira plugin that exports and sends the requirements including relevant properties to Nikke. For example, the relevant properties include priority and effort but exclude change history.
4. Nikke analyses the requirements using NLP technologies to detect new similarities or other interdependencies.
5. Nikke responds to the Jira plugin with the found interdependencies that are new and not explicated before in order to let requirements manager to review them.
6. The Jira plugin prompts requirements manager to accept or reject the new interdependencies.
7. The Jira plugin updates the accepted new interdependencies to Jira.
8. The Jira plugin notifies Nikke about the accepted interdependencies that are updated to Nikke's data model.
9. Nikke sends the requirement to Milla. The requirements include the properties as well as the accepted interdependencies.
10. Milla is responsible for parsing requirements information to a format understandable by Mulperi. Milla encapsulates the appropriately formatted requirements into a message that it sends to Mulperi. The currently primarily supported format is JSON format called MulSON.
11. Mulperi constructs a feature model in Kumbang language from the requirements
12. Mulperi sends the resulting Kumbang model in a XML message to SprinCaas.
13. SprinCaaS generates a CSP from the feature model. At the same time, SpringCaaS also carries out certain analyses, such as checks consistency.

14. SpringCaaS returns through Mulperi and Milla whether CSP construction succeeded or possible error.

The lower sequence in Figure 6 describes the scenario for product manager.

1. The product manager makes a decision that the requirements A and B should be in a release and makes the assignment to find direct consequences using the Jira plugin.
2. The Jira Plugin queries about the interdependencies of the requirements A and B by sending a message in JSON / XML to Mulperi's "Find direct Consequences" interfaces.
3. Mulperi converts the received query to the SpringCaaS Kumbang XML-format.
4. Mulperi sends the query message to SpringCaaS.
5. SpringCaaS reads the message, calculates interdependencies as a transitive closure on the basis of CSP constructed in the above sequence by using Choco solver for inference.
6. SpringCaaS returns interdependencies as an XML-based return message to Mulperi.
7. Mulperi returns the message to the Jira plugin that made the original query as a JSON message.
8. The Jira plugin proposes to the release manager the results of finding direct consequences by proposing to include the requirement C because it required by the requirement B.

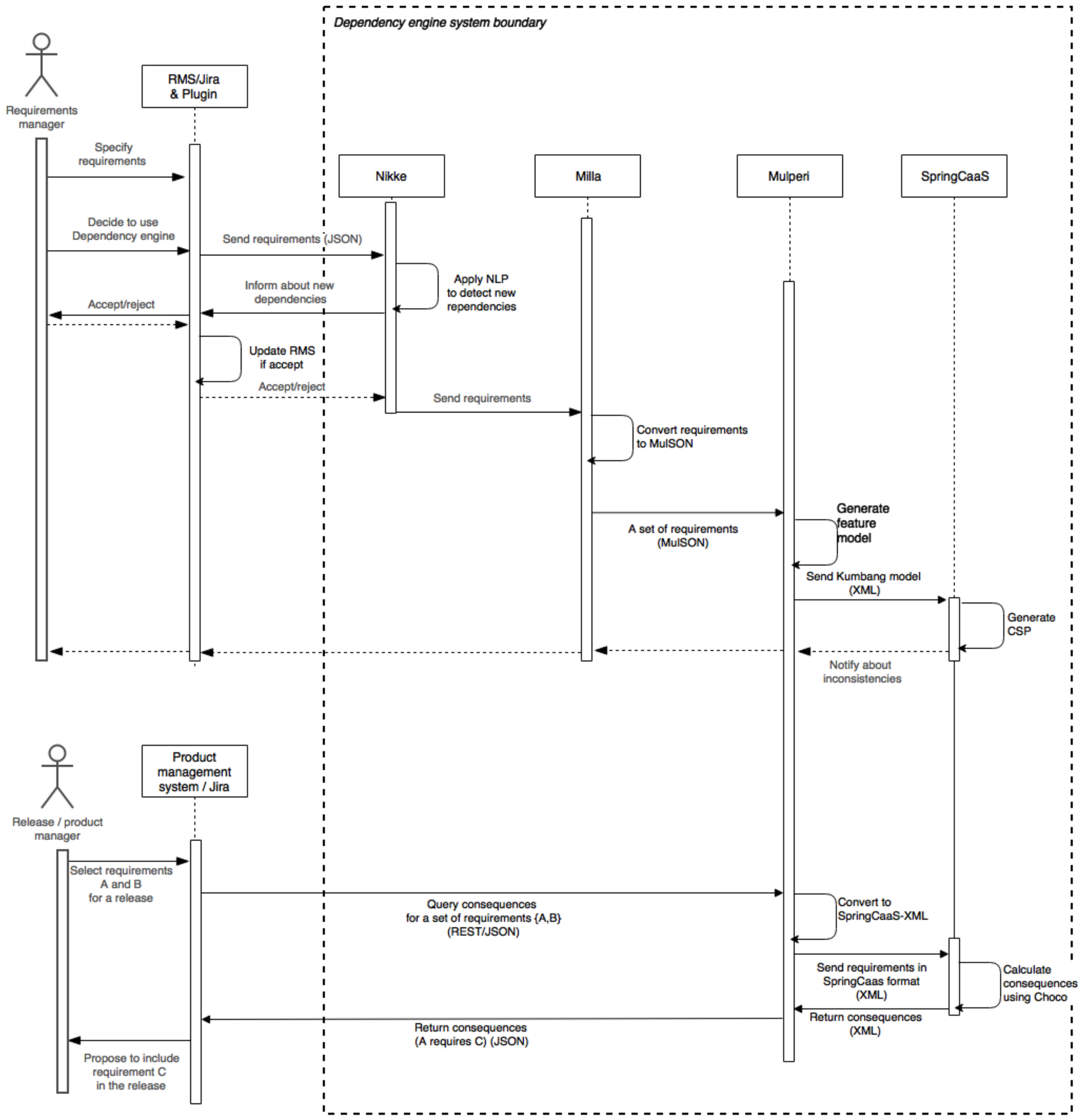

Dependency engine system boundary
Requirements manager
RMS/Jira & Plugin
Nikke
Milla
Mulperi
SpringCaaS
Specify requirements
Decide to use Dependency engine
Send requirements (JSON)
Apply NLP to detect new rependencies
Inform about new dependencies
Accept/reject
Update RMS if accept
Accept/reject
Send requirements
Convert requirements to MulSON
A set of requirements (MulSON)
Generate feature model
Send Kumbang model (XML)
Generate CSP
Notify about inconsistencies
Release / product manager
Product management system / Jira
Select requirements A and B for a release
Query consequences for a set of requirements {A,B} (REST/JSON)
Convert to SpringCaaS-XML
Send requirements in SpringCaas format (XML)
Calculate consequences using Choco
Return consequences (XML)
Return consequences (A requires C) (JSON)
Propose to include requirement C in the release

Figure 6. Example behavior of Dependency engine in the context of integration with a Jira plugin

References

Achimugu, P.; Selamat, A.; Ibrahim, R. & Mahrin, M. N. A systematic literature review of software requirements prioritization research, *Information and Software Technology,* 56(6),568-585, **2014**

Ali, R.; Dalpiaz, F. & Giorgini, P. Reasoning with contextual requirements: Detecting inconsistency and conflicts, *Information and Software Technology*, 55(1), 35-57, **2013**

Ameller, D.; Farré, C.; Franch, X. & Rufian, G. A Survey on Software Release Planning Models Product-Focused Software Process Improvement: 1*7th International Conference, PROFES 2016,* 48-65, **2016**

Anderson, A. Towards tool-supported configuration of services *Helsinki University of Technology, Department of Computer Science and Engineering (MSc thesis),* **2005**

Andersen, H.; Hadzic, T. & Pisinger, D. Interactive Cost Configuration Over Decision Diagrams *Journal of Artificial Intelligence Research, 37*, 99-139*,* **2010**

Asikainen, T., Männistö, T., & Soininen, T. Kumbang: A Domain Ontology for Modelling Variability in Software Product Families. Advanced Engineering Informatics, 21(1), 23-40, **2007**

Babar, M. I.; Ramzan, M. & Ghayyur, S. A. K. Challenges and future trends in software requirements prioritization *International Conference on Computer Networks and Information Technology,* 319-324 **2011**,

Bakker, R.; Dikker, F.; Tempelman, F. & Wogmim, P. Diagnosing and solving over-determined constraint satisfaction problems *13th International Joint Conference on Artificial Intelligence*, 276-281, **1993**

Bano, M.; Zowghi, D. & Ikram, N. Systematic reviews in requirements engineering: A tertiary study *2014 IEEE 4th International Workshop on Empirical Requirements Engineering (EmpiRE),*9-16 **2014**

Basili R., Bosco C., Delmonte R., Moschitti A. and Simi M. Harmonization and development of resources and tools for Italian natural language processing within the PARLI Project. Springer, **2015**.

Beatty, J., Stowe, M.J., Cardenas, A., Reinhart, D., & Bartlett, J. Requirements Management Tools Evaluation Report. Seilevel Report. Seilevel report. **2016**

Benavides, D.; Segura, S. & Ruiz-Cortes, A. Automated analysis of feature models 20 years later: A literature review *Information Systems, 35*, 615-636*,* **2010**

Birk, A., Heller, G. List of Requirements Management Tools. The Making of Software, http://makingofsoftware.com/resources/list-of-rm-tools. Accessed November 2017. **2017**

Bosch, J. Design and Use of Software Architectures: Adapting and Evolving a Product-Line Approach *Addison-Wesley, New York, USA,* **2000**

Bouraga, S.; Jureta, I. & Faulkner S. Requirements engineering patterns for the modeling of Online Social Networks features *IEEE 4th International Workshop on Requirements Patterns (RePa)*, 33-38, **2014**

Breaux, T. & Antón, A. Analyzing regulatory rules for privacy and security requirements *IEEE Transaction on Software Engineering*, 34(1), 5–20, **2008**

Burgess, C.; Livesay, K. & Lund, K. Explorations in context space: words *Sentences, Discourse, Discourse Processes*, 25(2-3), 211–257, **1998**

Calimeri, F.; Faber, W.; Gebser, M.; Ianni, G.; Kaminski, R.; Krennwallner, T.; Leone, N.; Ricca, F. & Schaub, T. ASP-Core-2: Input language format **2012**

Caralt, J.C. & Kim, J.W. Ontology Driven Requirements Query *40th Annual Hawaii International Conference on System Sciences (HICCS)*, **2007**

Carlshamre, P. Release Planning in Market-Driven Software Product Development: Provoking an Understanding *Requirements Engineering, 7*, 139-151*,* **2002**

Carlshamre, P.; Sandahl, K.; Lindvall, M.; Regnell, B. & och Dag, J. N. An industrial survey of requirements interdependencies in software product release planning *Proceedings Fifth IEEE International Symposium on Requirements Engineering,* 84-91 **2001**

Carrillo-de-Gea, J.M.; Nicolás, J.; Alemán, J.L.F.; Toval, A.; Vizcaíno, A. & Ebert, C. Reusing requirements in global software engineering *Managing requirements knowledge*, **2013**

Cesar Brandão Gomes da Silva, A.; de Figueiredo Carneiro, G.; Brito e Abreu, F. & Pessoa Monteiro, M. Frequent Releases in Open Source Software: A Systematic Review *Information, Multidisciplinary Digital Publishing Institute, 8*, 109*,* **2017**

Chen, Q.; Yao, L. & Yang, J.Short text classification based on LDA topic model *International Conference on Audio, Language and Image Processing (ICALIP)*, 749-753, **2016**

Chung, L. & Supakkul, S. Capturing and reusing functional and non-functional requirements knowledge: a goal-object pattern approach *IEEE international conference on information reuse and integration (IRI)*, 539–544, **2006**

Clements, P.; Northrop, L. M.; Gacek, C.; Knauber, P. & Schmidt, K. Software Product Lines: Practices and Patterns Successful Software Product Line Development in a Small Organization *Addison-Wesley,* **2001**

Cunis, R.; Günter, A.; Syska, I.; Peters, H. & Bode, H. PLAKON - an approach to domain-independent construction *2nd International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-89), 2*, 866-874*,* **1989**

CyLEDGE International Configurator Database *Cyledge configurator database,* **2015**

Czarnecki, K.; Helsen, S. & Eisenecker, U. W. Formalizing Cardinality-Based Feature Models and Their Specialization *Software process: Improvement and practice, 10*, 7-29*,* **2005**

Dahlstet, Å. G. & Persson, A. (Aurum, A. & Wohlin, C. *(Eds.))* Requirements Interdependencies: State of the Art and Future Challenges *Engineering and Managing Software Requirements, Springer Berlin Heidelberg,* 95-116, **2005**

Dalpiaz, F., Franch, X, & Horkoff, J. iStar 2.0 Language Guide. arXiv:1605.07767 https://arxiv.org/pdf/1605.07767v3.pdf, **2016**

Daneva, M. & Herrmann, A. Requirements prioritization based on benefit and cost prediction: A method classification framework *Software Engineering and Advanced Applications, 2008. SEAA'08. 34th Euromicro Conference*, 240-247*,* **2008**

Daramola, O.; Sindre, G. & Stalhane, T. Pattern-based Security requirements specification using ontologies and boilerplates *IEEE second international workshop on requirements patterns (RePa)*, 54–59, **2012**

Darr, T.; Klein, M. & McGuinness, D. L. Special issue: configuration design *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM), 12*, 293-294*,* **1998**

de Maat, E.; Winkels, R. & van Engers, T. Automated detection of reference structures in law *Conference on legal knowledge and information systems*, 41–50, **2006**

Dehlinger, J. & Lutz, T.R. A product-line requirements approach to safe reuse in multi-agent systems *4th international workshop on Software engineering for large-scale multi-agent systems (SELMAS '05)*, 1-7, **2005**

Eriksson, M.; Börstler, J.; Borg, K. Managing requirements specifications for product lines–An approach and industry case study *Journal of Systems and Software*, *82*(3), 435-447, **2009**

Escalona, M.J.; Urbieta, M.; Rossi, G.; Garcia-Garcia, J.A. & Robles Luna, E. Detecting Web requirements conflicts and inconsistencies under a model-based perspective *Journal of Systems and Software*, 86(12), 3024-3038, **2013**

Faltings, B. & Freuder, E. C. Special Issue on configuration *IEEE Intelligent Systems, 13*, 32-33*,* **1998**

Foltz, P.W.; Kintsch, W. & Landauer, T.K. The measurement of textual coherence with latent semantic analysis *Discourse Processes*, 25(2-3), 285–307, **1998**

Franch, X.; Quer, C.; Renault, S.; Guerlain, C.; Palomares, C. Constructing and using software requirement patterns *Managing requirements knowledge*, Springer, **2013**

Felfernig, A.; Stumptner, M. & Tiihonen, J. Special issue: configuration *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM), 25*, 113-114*,* **2011**

Felfernig, A.; Friedrich, G. E. & Jannach, D. UML as Domain Specific Language for the Construction of Knowledge-based Configuration Systems *International Journal of Software Engineering and Knowledge Engineering, 10*, 449-469*,* **2000**

Felfernig, A.; Friedrich, G. & Jannach, D. Conceptual modeling for configuration of mass-customizable products *Artificial Intelligence in Engineering, 15*, 165-176*,* **2001**

Felfernig, A.; Friedrich, G.; Schubert, M.; Mandl, M.; Mairitsch, M. & Teppan, E. Plausible Repairs for Inconsistent Requirements *Proceedings of the 21st International Jont Conference on Artificial Intelligence,* 791-796*,* **2009**

Felfernig, A.; Schubert, M. & Zehentner, C. An Efficient Diagnosis Algorithm for Inconsistent Constraint Sets *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM), 26*, 53-62*,* **2012**

Gabrilovich, E. & Markovitch, S. Computing Semantic Relatedness Using Wikipedia-based Explicit Semantic Analysis *20th International Joint Conference on Artificial Intelligence*, pp 1606–1611, **2007**

Galster, M.; Weyns, D.; Tofan, D.; Michalik, B. & Avgeriou, P. Variability in Software Systems --- A Systematic Literature Review *IEEE Transactions on Software Engineering, 40*, 282-306*,* **2014**

Gebser, M.; Kaufmann, B.; Kaminski, R.; Ostrowski, M.; Schaub, T. & Schneider, M. Potassco: The Potsdam Answer Set Solving Collection *AI Communications, 24*, 107-124*,* **2011**

Goldin, L.; Berry, D.M. Reuse Of requirements reduced time to market at one industrial shop: a case study *Requir Eng*, 20(1), **2013**

Gotel, O. C. Z. & Finkelstein, C. W. An analysis of the requirements traceability problem *Proceedings of IEEE International Conference on Requirements Engineering*, 94-101*,* **1994**

Hamdaqa, M. & Hamou-Lhadj, A. An approach based on citation analysis to support effective handling of regulatory compliance *Fut Gen Comput Syst*, 27(4), 395–410, **2009**

Hauksdottir, D.; Vermehren, A. & Savolainen, J. Requirements reuse at Danfoss 20th *IEEE international conference on requirements engineering (RE)*, 309–314, **2012**

Hauksdottir, D.; Ritsing, B.; Andersen, J.C: & Mortensen, N.H. Establishing Reusable Requirements Derived from Laws and Regulations for Medical Device Development *IEEE 24th International Requirements Engineering Conference Workshops (REW)*, 220-228, **2016**

Heinrich, M. & Jüngst, E. W. A resource-based paradigm for the configuring of technical systems from modular components *Seventh IEEE Conference on Artificial Intelligence Applications (CAIA-91),* 257-264, **1991**

Herrmann, A. & Daneva, M. Requirements Prioritization Based on Benefit and Cost Prediction: An Agenda for Future Research *16th IEEE International Requirements Engineering Conference*, 125-134*,* **2008**

Heumesser, N. & Houdek, F. Towards systematic recycling of systems requirements *25th International Conference on Software Engineering*, 512-519*,* **2003**

Hippel EV, Krogh GV. Open source software and the “private-collective” innovation model: Issues for organization science. Organization science, 14(2), 209-23 **2003**

Hiisilä, H.; Kauppinen, M. & Kujala S. Challenges of the Customer Organization’s Requirements Engineering Process in the Outsourced Environment – A Case Study *Requirements Engineering: Foundation for Software Quality*, Lecture Notes in Computer Science, vol 9013, **2015**

Hofmann, T. Probabilistic latent semantic indexing *International ACM SIGIR Conference (SIGIR ’99)*, 50–57, **1999**

Hotomski, S.; Charrada, E.B. & Glinz, M. An Exploratory Study on Handling Requirements and Acceptance Test Documentation in Industry *IEEE 24th International Requirements Engineering Conference (RE)*, 116-125, **2016**

Irshad, M.; Petersen, K. & Poulding, S. A systematic literature review of software requirements reuse approaches *Information and Software Technology*, 93, 223-245, **2017**

Issa, A.A. & Al-Ali, A. Use Case Patterns Driven Requirements Engineering *Second International Conference on Computer Research and Development*, 307-313, **2010**

Issa, A.A. &. Al-Ali, A.I. Automated requirements engineering: Use case patterns-driven approach *IET Software*, 5(3), 287-303, **2011**

Jain, P.; Verma, K.; Kass, A. & Vasquez, R.G. Automated review of natural language requirements documents: generating useful warnings with user-extensible glossaries driving a simple state machine *2nd India software engineering conference (ISEC '09)*, 37-46, **2009**

Jensen, J.; Tondel, I.A.; Jaatun, M.G.; Meland, P.H. & Andresen, H. Reusable security requirements for healthcare applications *International conference on availability, reliability and security (ARES)*, 380–385, **2009**

Junker, U. QuickXPlain: Preferred Explanations and Relaxations for Over-Constrained Problems *19th National Conference on Artificial Intelligence (AAAI'04),* 162-172*,* **2004**

Kang, K.; Cohen, S.; Hess, J.; Novak, W. & Peterson, A. Feature-Oriented Domain Analysis (FODA) Feasibility Study *Software Engineering Institute, Software Engineering Institute,* **1990**

Kang, K.; Kim, S.; Lee, J.; Kim, K.; Shin, E. & Huh, M. FORM: A feature-oriented reuse method with domain-specific reference architectures *Annals of Software Engineering, 5*, 143-168*,* **1998**

Karlsson, J.; Olsson, S. & Ryan, K. Improved practical support for large-scale requirements prioritising *Requirements Engineering, 2*, 51-60*,* **1997**

Kitchenham, B.; & Charters, S. Guidelines for Performing Systematic Literature Reviews in Software Engineering *EBSE Technical Report EBSE-2007-01*, **2007**

Konrad, S. & Cheng, B.H. Requirements patterns for embedded systems *IEEE Joint International Conference on Requirements Engineering (RE)*, 127–136, **2002**

Konrad, S. & Cheng, B.H. Real-time specification patterns *27th International Conference on Software engineering (ICSE)*, 372–38, **2005**

Lam, W.; McDermid, J.A. & Vickers, A.J. Ten steps towards systematic requirements reuse *Requir Eng*, 2(2), **1997a**

Lam, W. Achieving requirements reuse: A domain-specific approach from avionics *Journal of Systems and Software*, 38(3), 197-209, **1997b**

LeClair, A., Bittner, K., Mines, C., Turrisi, T. TechRadar™: Modern Software Requirements Management Tools, Q2. *Forrester report*. **2016**

Lee, M.C.; Chang, J.W. & Hsieh, T.C A Grammar-Based Semantic Similarity Algorithm for Natural Language Sentences *The Scientific World Journal*, **2014**

Lehtola, L.; Kauppinen, M. & Kujala, S. Requirements Prioritization Challenges in Practice *Product Focused Software Process Improvement: 5th International Conference*, 497-508, **2004**

Lin, J. & Gunopulos, D. Dimensionality reduction by random projection and latent semantic indexing *3rd SIAM International Conference on Data Mining*, **2003**

Mahmoud, A. & Niu, N. An experimental investigation of reusable requirements retrieval *IEEE International Conference on Information Reuse & Integration*, 330-335, **2010**

Mannion, M.; Keepence, B.; Kaindl, H. & Wheadon, J. Reusing single system requirements from application family requirements *International Conference on Software Engineering*, 453-462, **1999**

Mavin, A.; Wilksinson, P.; Gregory, S. & Uusitalo, E. Listens Learned (8 Lessons Learned Applying EARS) *2016 IEEE 24th International Requirements Engineering Conference*, 276-282*,* **2016**

Mavin, A.; Wilkinson, P.; Teufl, S.; Femmer, H.; Eckhardt, J. & Mund, J. Does Goal-Oriented Requirements Engineering Achieve Its Goal? *2017 IEEE 25th International Requirements Engineering Conference*, 174-183*,* **2017**

Mazo, R. & Feltus, C. Framework for Engineering Complex Security Requirements Patterns *26th International Conference on IT Convergence and Security (ICITCS)*, 1-5, **2016**

McDermott, J. R1: A Rule-based configurer of computer systems *Artificial Intelligence, 19*, 39-88*,* **1982**

Méndez, D. & Wagner, S. Naming the pain in requirements engineering: A design for a global family of surveys and first results from Germany *Information and Software Technology*, Volume 57, 616-643, **2015**

Méndez Fernández, D.M., Wagner, S., Kalinowski M., et al., Naming the pain in requirements engineering. Empirical Software Engineering, 22(5), 2298–2338, **2017**

Misra, J. Terminological inconsistency analysis of natural language requirements *Inf. Softw. Technol.*, 74, 183-193, **2016**

Mittal, S. & Frayman, F. Towards a generic model of configuration tasks *11th International Joint Conference on Artificial Intelligence (IJCAI-89), 2*, 1395-1401*,* **1989**

Murphy, T.E., Revang, M., Wurster, L.F. . Market Guide for Software Requirements Definition and Management Solutions. *Gartner report. ID: G00276075*. **2016**

Myklebust, T.; Lyngby, N.; Bains, R. & Hanssen, G.K. CoVeR maintenance and maintainability requirements by using tools *Reliability and Maintainability Symposium*, 1-6, **2014**

Myllärniemi, V.; Ylikangas, M.; Raatikainen, M.; Pääkkö, J.; Männistö, T. & Aaltonen, T. Configurator-as-a-service: tool support for deriving software architectures at runtime *Working IEEE / IFIP Conference on Software Architecture, Companion Volume*, 151-158*,* **2012**

Najmann, O. & Stein, B. A Theoretical Framework for Configuration *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: 5th International Conference (IEA/AIE-92), Springer, LNCS 604*, 441-450*,* **1992**

Natt och Dag, J.; Regnell, B.; Carlshamre, P.; Andersson, M. & Karlsson, J. A Feasibility Study of Automated Natural Language Requirements Analysis in Market-Driven Development *Requirements Eng*, 7(20), **2002**

OMG - Object Management Group Requirements Interchange Format (ReqIF) Version 1.2 (TechReport) *Object Management Group, Inc.,* **2016**,

Osterwalder, A. The Business Model Ontology - A Proposition In A Design Science Approach. PhD thesis, University of Lausanne, (2004).

Pacheco, C.; Garcia, I.; Calvo-Manzano, J.A. & Arcilla, M. Reusing functional software requirements in small-sized software enterprises: a model oriented to the catalog of requirements *Requirements Engineering,* 22(2), 275-287, **2017**

Palmirani, M.; Brighi, R. & Massin,i M. Automated extraction of normative references in legal texts *9th international conference on artificial intelligence and law (ICAIL'03)*, 105–106, **2003**

Palomares, C.; Quer, C. & Franch, X. Requirements reuse and requirement patterns: a state of the practice survey *Empir Software Eng*, 22(6), 2719-2762, **2017**

Panis, M.C. Reuse of Architecturally Derived Standards Requirements *IEEE International Requirements Engineering Conference (RE)*, 296–304, **2015**

Pedersen, T. & Kulkarni, A. Identifying similar words and contexts in natural language with SenseClusters *20th National Conference on Artificial intelligence (AAAI'05)*, 1694-1695, **2005**

Pergher, M. & Rossi, B. Requirements prioritization in software engineering: A systematic mapping study *2013 3rd International Workshop on Empirical Requirements Engineering (EmpiRE),* 40-44, **2013**

Perrouin, G.; Brottier, E.; Baudry, B. & Le Traon Y. Composing Models for Detecting Inconsistencies: A Requirements Engineering Perspective *Requirements Engineering: Foundation for Software Quality (REFSQ)*, **2009**

Petersen, K.; Vakkalanka, S. & Kuzniarz, L. Guidelines for conducting systematic mapping studies in software engineering: An update *Information and Software Technology*, 64, **2015**

Pitangueira, A.; Maciel, R. & Barros, M. Software requirements selection and prioritization using SBSE approaches: A systematic review and mapping of the literature *Journal of Systems and Software, 103*, 267-280*,* **2015**

Pohl, K. Process-centered requirements engineering *John Wiley & Sons, Inc.,* **1996**

Pohl, K.; Böckle, G. & van der Linden, F. Software Product Line Engineering: Foundations, Principles, and Techniques *Springer-Verlag Berlin Heidelberg, Germany,* **2005**

Post, A.; Menzel, I. & Podelski, A. Applying restricted English grammar on automotive requirements: does it work? A case study *17th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, 166–180, **2011**

Prifti, T.; Banerjee, P. & Cukic, B. Detecting bug duplicate reports through local references *7th International Conference on Predictive Models in Software Engineering*, 1-9, **2011**

Raatikainen, M.; Männistö, T.; Tommila, T. & Valkonen, J."Challenges of requirements engineering — A case study in nuclear energy domain *IEEE 19th International Requirements Engineering Conference*, 253-258, **2011**

Raatikainen, M.; Tiihonen, J.; Männistö, T. Software Product Lines and Variability Modeling: A Tertiary Study, *Journal of Systems and Software,* (submitted), **2017**

Raymond, Eric. "The cathedral and the bazaar." Philosophy & Technology 12(3), **1999**

Rehbein I., Ruppenhofer J., Sporleder C., and Pinkal M. Adding nominal spice to SALSA - frame-semantic annotation of German nouns and verbs. Conference on Natural Language Processing (KONVENS), **2012**

Renault, S.; Méndez, O.; Franch, X. & Quer, C. A Pattern-based Method for building Requirements Documents in Call-for-tender Processes *Int J Comput Sci Appl*, 6(5), 175–202, **2009**

Riegel, N. & Doerr, J. A systematic literature review of requirements prioritization criteria *International Working Conference on Requirements Engineering: Foundation for Software Quality*, 300-317 *,* **2015**

Rine, D.C. & Nada, N. An empirical study of a software reuse reference model. *Information and Software Technology*, 42(1), **2000**

Runeson, P.; Alexandersson, M. & Nyholm, O. Detection of Duplicate Defect Reports Using Natural Language Processing *29th International conference on Software Engineering (ICSE '07)*, pp.499-510, **2007**

Sabin, D. & Weigel, R. Product configuration frameworks — a survey *IEEE Intelligent Systems, 13*, 42-49*,* **1998**

Sannier, N.; Adedjouma, M.; Sabetzadeh, M. & Briand, L.C. An automated framework for detection and resolution of cross references in legal texts *Requirements Engineering*, 22(2), 215-237, **2017**

Schmidt, D; Fayad, M. & Johnson, R. Software Patterns *Communications of The ACM*, **1996**

Schobbens, P.-Y.; Heymans, P.; Trigaux, J.-C. & Bontemps, Y. Generic semantics of feature diagrams *Computer Networks, 51*, 456-479*,* **2007**

Sengar, P. MarketScope for Configure, Price and Quote Application Suites *Gartner, Inc.,* **2013**

Sikora E.; Tenbergen B. & Pohl K. Requirements Engineering for Embedded Systems: An Investigation of Industry Needs *Requirements Engineering: Foundation for Software Quality (REFSQ)*, Lecture Notes in Computer Science, vol 6606, **2011**

Silva, A.; Silva, A.; Araújo, T.; Willamy, R.; Ramos, F.; Costa, A.; Perkusich, M. & Dilorenzo, E. Ordering the product backlog in agile software development projects: A systematic literature review *Proceedings of the International Conference on Software Engineering and Knowledge Engineering*, 74-80*,* **2017**

Simons, P.; Niemelä, I. & Soininen, T. Extending and implementing the stable model semantics *Artificial Intelligence, 138*, 181-234*,* **2002**

Sinz, C.; Haag, A.; Narodytska, N.; Walsh, T.; Gelle, E.; Sabin, M.; Junker, U.; O'Sullivan, B.; Rabiser, R.; Dhungana, D.; Grunbacher, P.; Lehner, K.; Federspiel, C. & Naus, D. Configuration *IEEE Intelligent Systems, IEEE, 22*, 78-90*,* **2007**

Sher, F.; Jawawi, D. N. A.; Mohamad, R. & Babar, M. I. Requirements prioritization techniques and different aspects for prioritization a systematic literature review protocol *2014 8th. Malaysian Software Engineering Conference (MySEC)*, 31-36*,* **2014**

Soininen, T. & Stumptner, M. Special issue: configuration *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM), 17*, 1-2*,* **2003**

Soloway, E.; Bachant, J. & Jensen, K. Assessing the Maintainability of XCON-in-RIME: Coping with the Problem of very large Rule-bases *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, 824-829*,* **1987**

Sommerville, I. Software engineering **2010**

Souag, A.; Mazo, R.; Salinesi, C. & Comyn-Wattiau, I. Reusable knowledge in security requirements engineering: a systematic mapping study *Requir Eng*, **2015**

Stumptner, M. An Overview of Knowledge-based Configuration *AI Communications, 10*, 111-125*,* **1997**

Sun, C.; Lo, D.; Khoo, S.C. & Jiang, J. Towards more accurate retrieval of duplicate bug reports *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, 253-262, **2011**

Svahnberg, M.; van Gurp, J. & Bosch, J. A taxonomy of variability realization techniques *Software --- Practice and Experience, 35*, 705-754*,* **2005**

Svahnberg, M.; Gorschek, T.; Feldt, R.; Torkar, R.; Saleem, S. B. & Shafique, M. U. A systematic review on strategic release planning models *Information and Software Technology, 52*, 237 - 248, **2010**

Thakurta, R. Understanding requirement prioritization artifacts: a systematic mapping study *Requirements Engineering*, 1-36, **2016**

Tiihonen, J. Support for configuration of physical products and services. Aalto University publication series Doctoral dissertations, 153/2014, **2014**

Tiihonen, J.; Raatikainen, M.; Myllärniemi, V. & Männistö, T. Carrying Ideas from Knowledge-Based Configuration to Software Product Lines *Software Reuse: Bridging with Social-Awareness: 15th International Conference, ICSR 2016*, 55-62*,* **2016**

Tiihonen, J. & Soininen, T. Product configurators - information system support for configurable products *Laboratory of Information Processing Science Technical Report series, Helsinki University of Technology, Helsinki University of Technology, Helsinki University of Technology, TKO-B 137,* **1997**

Tiihonen, J. & Felfernig, A. An introduction to personalization and mass customization *Journal of Intelligent Information Systems, 49*, 1-7*,* **2017**

Tiihonen, J.; Felfernig, A.; Zanker, M. & Männistö, T. Special issue: advances in configuration systems: editorial *International Journal of Mass Customisation, 3*, 311-315*,* **2010**

Toval, A.; Nicolás, J.; Moros, B. & García, F. Requirements reuse for improving information systems security: a practitioner’s approach, *Requirements Engineering*, 6(4), **2002**

Tran, T.O.; Xuan, B.N.; Le, N.M. & Akira, S. Automated reference resolution in legal texts *Artif Intell Law*, 22(1), 29–60, **2014**

Verma, K. & Kass, A. Requirements Analysis Tool: A Tool for Automatically Analyzing Software Requirements Documents *The Semantic Web - ISWC 2008, Lecture Notes in Computer Science*, vol 5318, **2008**

Vidal, E.; Thollard, F.; de la Higuera, C.; Casacuberta, F. & Carrasco, R.C. Probabilistic finite-state machines part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7), 1013–1025, **2005**

Vogelsang, A. & Fuhrmann, S. Why feature dependencies challenge the requirements engineering of automotive systems: An empirical study *2013 21st IEEE International Requirements Engineering Conference (RE)*, 267-272*,* **2013**

Wahono, R.S. & Cheng, J. Extensible requirements patterns of web application for efficient web application development *First International Symposium on Cyber Worlds,* 412-418, **2002**

Wang, X.; Zhang, L.; Xie, T.; Anvik, J. & Sun, J. An approach to detecting duplicate bug reports using natural language and execution information *ACM/IEEE 30th International Conference on Software Engineering*, 461-470, **2008**

Wang, M. & Cer, D. Stanford: probabilistic edit distance metrics for STS *Sixth International Workshop on Semantic Evaluation (SemEval '12)*, 648-654, **2012**

Weiss, D. M. & Lai, C. T. R. Software Product-Line Engineering: A Family-Based Software Development Process *Addison-Wesley Longman Publishing Co., Inc.,* **1999**

Withall, S. Software requirement patterns *Microsoft Press*, **2007**

Zhang, H.; Li, J.; Zhu, L.; Jeffery, R.; Liu, Y.; Wang, Q. & Li, M. Investigating dependencies in software requirements for change propagation analysis *Information and Software Technology, 56*, 40-53*,* **2014**

Zhu, X. & Jin, Z. Inconsistency measurement of software requirements specifications: an ontology-based approach *10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, 402-410, **2005**

Zuccato, A.; Daniels, N. & Jampathom, C. Service Security Requirement Profiles for Telecom: How Software Engineers May Tackle Security *Sixth International Conference on Availability, Reliability and Security*, 521-526, **2011**

Appendix

Appendix 1: Full details of interdependency taxonomies

Pohl taxonomy

Type	**Meaning**
Condition type	
Constraint	Used to relate a constraint to a particular object.
Precondition	Conditions that must be fulfilled to enable implementation of the requirement.
Content type	
Similar	Similar objects
Compares	Links to a result of a comparison
Contradicts	Inconsistency between requirements
Conflicts	Negative influence to another requirement
Documents type	
Example_for	Example (real-world scenes or scenarios)
Test_case_for	Relate test case that validates requirement
Purpose	E.g. informal text
Background	More details
Comment	Arbitrary information
Evolutionary type	
Replaces	Requirement has been replaced with another
Satisfies	Satisfying target results in satisfying also source
Based_on	influences , e.g., causes creation
Formalizes	Target formalizes source. Specialization of based on.

Elaborates	A more comprehensive description gained later in the development process
Abstraction	
Generalization	Target is generalization
Refines	Target object is defined in more detail by another requirement

Carlshamre taxonomy

Type	**Meaning**
R1 AND R2	R1 requires R2 to function, and R2 requires R1 to function.
R1 REQUIRES R2	R1 requires R2 to function but not vice versa.
R1 TEMPORAL R2	Either R1 has to be implemented before R2 or vice versa.
R1 CVALUE R2	R1 affects the value of R2 for a customer. Value can be positive or negative.
R1 ICOST R2	R1 affects the cost of implementing R2. Value can be positive or negative.
R1 OR R2	Only one of {R1,R2} needs to be implemented

Dahlstedt taxonomy

Type	**Meaning**
Structural types	
Refined_to	Hierarchical structure for more specific requirements that provides further explanations, details or clarification. Synonyms: Elaborated, derived from, divide into parts, formalizes, (generalization). Practically optional part (cf. requires).
Changes_to	New version replaces the old one.
Similar_to	Similar to or overlapping with one or more other

	requirements. Similarly expressed; similar underlying idea what system should do; or similar solutions from which one has to be selected. That is, either similarity in requirements or potential solutions.
Constraining types	
Requires	One requirement depends on the fulfillment of another requirement; if one requirement is to be included into the system, it requires another requirement be included; one requirement cannot work without another; or temporal interdependency when one needs to be implemented before another. Can be used for hierarchies stronger than refined_to (mandatory part). Also weaker forms possible: support or enhance; positive effect.
Conflicts_with	Cannot exists at the same time; or increasing the satisfaction of one requirement decreases the satisfaction of another requirement. Thus, impossible to implement both requirements or negative effect.
Cost/value types	
Increases/Decreases_cost_of	If one requirement is chosen for implementation, then the costs of implementing another requirement increases or decreases.
Increases/Decreases_value_of	If one requirement is chosen for implementation, then the value of implementing another requirement increases or decreases Negative, e.g., by making functionality more complex.

Zhang taxonomy

Interdependency	Class[1]	Description	In other models
Constrain	B,I	One requirement is a constraint of another requirement. This kind of interdependency can represent crosscutting relationship among requirements	
Precede	B	If function A precedes function B, A is a precondition of B.	Precondition, require
Be_similar_to	I	If two requirements share similar data information, these two requirements are similar to each other. If two requirements complete similar tasks, these two requirements are similar to each other. Note: symmetric	Similar, Similar_to
Refine	S	One requirement is refined by more specific requirements.	Refines, Refines_to
Be_exception_ of	B,I	One requirement describes the exceptional event of another requirement.	(new)
Conflict	I	Implementation of one requirement negatively impacts another requirement	Conflicts, Conflicts_with
Evolve_into	E	If one requirement B is a new version of another requirement A, then A evolves into B.	Combine, Replaces, Satisfies, Based_on, Formalises, Elaborates, Changes_to
Increase/ Decrease_cost_of	C	The implementation of one requirement causes the increase/decrease of the implementation cost of another requirement.	
Increase/ Decrease_value_of	V	The implementation of one requirement causes the increase/decrease of the value to the customer of another requirement.	

[1] Business (B), implementation (I), structure (S), evolution (E), value (V), cost (C).

Appendix 2: Full details of RMS comparison

	Caliber	DOORS	Helix RM	Modern Requirements4TFS
Copy and Paste	The requirement or the requirement and its child requirements.	The requirement or the requirement and its properties, and any (or all) interdependencies are copied.	The requirement or the requirement and its properties, and interdependencies can be copied.	Although it seems possible it has been difficult to use this functionality.
Mapping of Requirements	Just synchronization of the requirement description. The requirement description of the mapped requirement changes without any notice nor notification. Mapped requirements visualization to see mapping links.	It seems possible by the documentation but it cannot be tested. It seems difficult to apply.	No	No
Use of Requirement Templates	No	Requirement templates are allowed to be created, which correspond to specific requirements that can be reused in different situations. It has not been possible to test if they can be used in different projects.	No	It is possible to apply reusable requirements that can be seen as requirement templates. They are associated with queries to be used during requirements elicitation and can be imported and exported from a template (see Requirement Libraries below).
Use of Project Template	No	Project templates can be used to establish a starting point for artifact types, property types, interdependency types, and folder structure. It is possible to create project templates.	Yes	The tool provides three project templates. It has not been possible to check if it is possible to create new project templates.

Use of Requirement Libraries	No.	No.	No.	A library of questions (named FAQ) that help requirements engineer to elicit requirements. These questions are organized by functional and non-functional characteristics of software. FAQs can be adapted and saved as a questions template for new projects. Reusable requirements can be defined as answers to the questions in the library.

	Jama	Jira	Polarion	TopTeam
Copy and Paste	A requirement.	A requirement.	The requirement and its interdependencies..	Yes, although we do not know the part of the requirement it clones since the demo version, that we could access, did not have the full functionality.
Mapping of Requirements	Mapping of specific requirements allowed and it can be chosen, what is included. Possible to define rules to state which parts of requirements must be mapped and when to apply the synchronization of mapped requirements when base requirements change.	A copied cloned requirement can be mapped to its original counterpart. Practically, this is an interdependency link. Changes to either requirement do not propagate, but they exist separately.	It is possible to define derived requirements. These requirements cannot be updated automatically. When original requirement is updated, the change is marked. The user can synchronize both requirements.	No
Use of Requirement Templates	No	No	No	No

Use of Project Template	A project can be configured to act as a template so that it can be reused and synchronized. This can be done in Jama through the duplicate project option that allows synchronization.	A Jira project is a collection of requirements adhering to requirements types (issues, user stories, bugs, tasks) that share notification settings, a common workflow and issue field configuration scheme. A project is distinguished in requirements as a prefix in the requirement's IDname (e.g. "QTBUG-".	Yes	When a project is created from an existing project Template, the project can then be used as-is or can be customized further to suit specific needs. It has not been possible to check if it is possible to create new project templates.
Use of Requirement Libraries	It can be done through the use of containers of reusable requirements or through the duplication of projects that can be considered reusable requirement libraries. However, the concept of library is not managed by the tool.	No.	No.	No.

	Caliber	DOORS	Helix RM	Modern Requirements4TFS
Term used	Trace	Link	Link	Link
Interdependencies Extraction	No	No	No	No
Interdependencies definition	-Parent / Child -Traces From /To It is not possible to add trace types	-Constraints -Extracted -Link To -References -Satisfies It is possible to add link types	-Business/ Functional -Business/Non Functional - Functional/Non-Technical -Related To It is possible to add link types	-Affects /Affected By -Duplicate of -Parent/Child - References/Reference d By -Related - Successor/Predecess or It was not possible to know if new link types can be defined.
Interdependency types semantics definition	No	Yes, visible in the link types catalogue. It is possible to define link constraints that have to be fulfilled when an interdependency is defined Link validity to check the compliance of the link constraints	Yes, visible when a link type is selected in creating a link. No control on the items for which a type of the link is specified	No
Interdependencies traceability	Traceability matrix Traceability diagram	Links explorer	Traceability matrix with multiple filters Impact analysis	Traceability matrix with multiple filters
Tagging of suspect interdependencies	In case of changes in a requirement, the trace where it is involved automatically become suspect.	Suspect Links. It is configurable whether changes make the links tagged as suspect links. Link validity analyses the interdependencies when there are multiple interdependencies among requirements, and different levels of links.	In a case of changes in a requirement, the user has to explicitly tag if a link that is involved has become suspect.	As far as we know, the tool does not give the functionality of tagging links as suspect.

	Jama	Jira	Polarion	TopTeam
Term	Relationship	Link	Link	Link
Interdependencies Extraction	No	No	No	No
Interdependencies definition	-Related to -Depends on -Derived from It is possible to create new relationship types.	-Clones / Cloned by -Blocks / Is blocked by -Requires/Is required by -Causes / Is caused by -Relates to	-Relates to/is related to -Derived from/derived by -Duplicates/duplicated by -Has parent/is parent of It's not possible to add link types.	-Trace In/Traces From -Used In/Uses -Impacts/Dependent It is possible to create new link types.
Interdependency types semantics definition	No.	No.	No.	It has not been possible to find a definition for the predefined link types, although it is possible to introduce a description when a new link types created. It is possible to define traceability rules to define constraints about the types of requirements for which an interdependency type can be defined.
Interdependencies traceability	Trace Matrix Trace View	There is an additional plug-in (TraceabilityX) for this.	Treeview (parent-child relationships). Traceability matrix.	Traceability Tab Trace Explorer Trace Diagram Traceability Network Diagram

Tagging suspect interdependencies	When changes occur in one requirement, the interdependencies where it is involved are automatically tagged as suspect.	The user can configure an email notification for “update” of an issue which should cover an interdependency change to that requirement also.	If the property is modified, the suspect property will be applied automatically to child work Item links, if the parent Work Item is modified.	Interdependencies, in which one of the involved requirements changes, can automatically be made suspect. It is necessary to turn on the automatic suspect behavior option. There are inconsistencies in documentation and it is not clear if changes can tag suspect links recursively or not. It could not be tested

	Caliber	DOORS	Helix RM	Modern Requirements4TFS
Glossary and natural language analysis support	Glossaries for individual projects. Highlight in text of ambiguous terms, and other words in the glossary.	Glossaries for individual projects. It is possible to create interdependencies of requirement parts with glossary terms	Spell check dictionary extendible with new words	No
Recommendation of Requirements	No	No	No	No
Search	Filtering in a grid view of requirements. Possibility to save the search, and save the filter once used.	Filter requirements in a view. Find/Replace. Possible to add tags to requirements to facilitate search and classification	Grid visualization and filtering.	Filter requirements in a view that contain certain terms. Possible to define and save queries that are filters of requirements that have specific values for their properties. Queries may be global to all users of a project or personal. Each query may have different properties for the included requirements. Possible to add tags to requirements to facilitate search and classification
Links to other Requirements in Requirement properties	No	Yes	No	No
Dashboards	No	Project and team events and information Requirement view Recent activity Configurable	Work left, work done in a stage of a project Metrics of the project Requirements assigned to the user Recent activity Configurable	Charts about requirements, number of recent changes, results from a query, work assigned to the user, etc. Configurable

Customization	An API is provided to create add-ons using either Java or .NET. It is possible to add a traceability add-in to provide traceability to an external tool.	Extensions can be developed by using a combination of JavaScript, HTML, and CSS files. The extensions access data by using an API.	Developers can currently use the REST API to retrieve information about artifacts managed by the tool.	No documentation of the existence of an SDK has been found.
Integration	Provision of traceability from requirements to HP Quality Center, among others by means of a traceability add-in.	Supports a wide variety of integrations with other IBM® and third-party products. Open Services for Lifecycle Collaboration (OSLC) provides artifact creation, linking, and data sharing across applications.	Integration with Jira, MS Excel, MS Outlook, QA Wizard Pro	Integration with other products of the same provider as Smart Office 4TFS
ReqIf Format	No	Yes	No	No

	Jama	Jira	Polarion	TopTeam
Glossary and natural language analysis support	No	No	No.	Glossaries for individual projects that can be imported from other glossaries. Glossary used to suggest words to include in requirements definition
Recommendation of Requirements	No	No	No	No
Search	Yes, with advanced search filters.	Yes	Yes.	Necessary to previously create a data filter to do the search. Also possible a global search by keywords and other properties.

Links to other Requirements in Requirement properties	No	No	Yes	Yes.
Dashboards	Bar or Pie charts with results of a filter, summary of projects, work assigned to the user, recent activity, etc. Yes, configurable.	Yes. Configurable and personalized dashboards.	Yes, configurable.	Dashboard view of Projects that displays status information to Business Analysts, Project Managers, Team Members, etc. Yes, configurable
Customization	Not customizable per se, offers restful API, allowing to build applications for exchanging and manipulating requirements data.	Versatile existing plugins for customizing functionality and user interface of the software. Offers both a SDK and Java&REST APIs.	A set of plugins are offered for enabling requirement templates, customizing workflow and reporting. SDK and API available.	No SDK and only basic API for exchanging requirements data available.
Integration	Offers an “Integration tasktop hub”, enabling support for e.g. Enterprise Architect, Jira and VersionOne.	Integration plug-ins available extensively for software development and project management tools.	Software plugins enable integration with Jira, Enterprise Architect and TeamCenter.	Manufacturer does not offer integrations, yet other systems support TopTeam.
ReqIf Format	No	No.	Yes	No

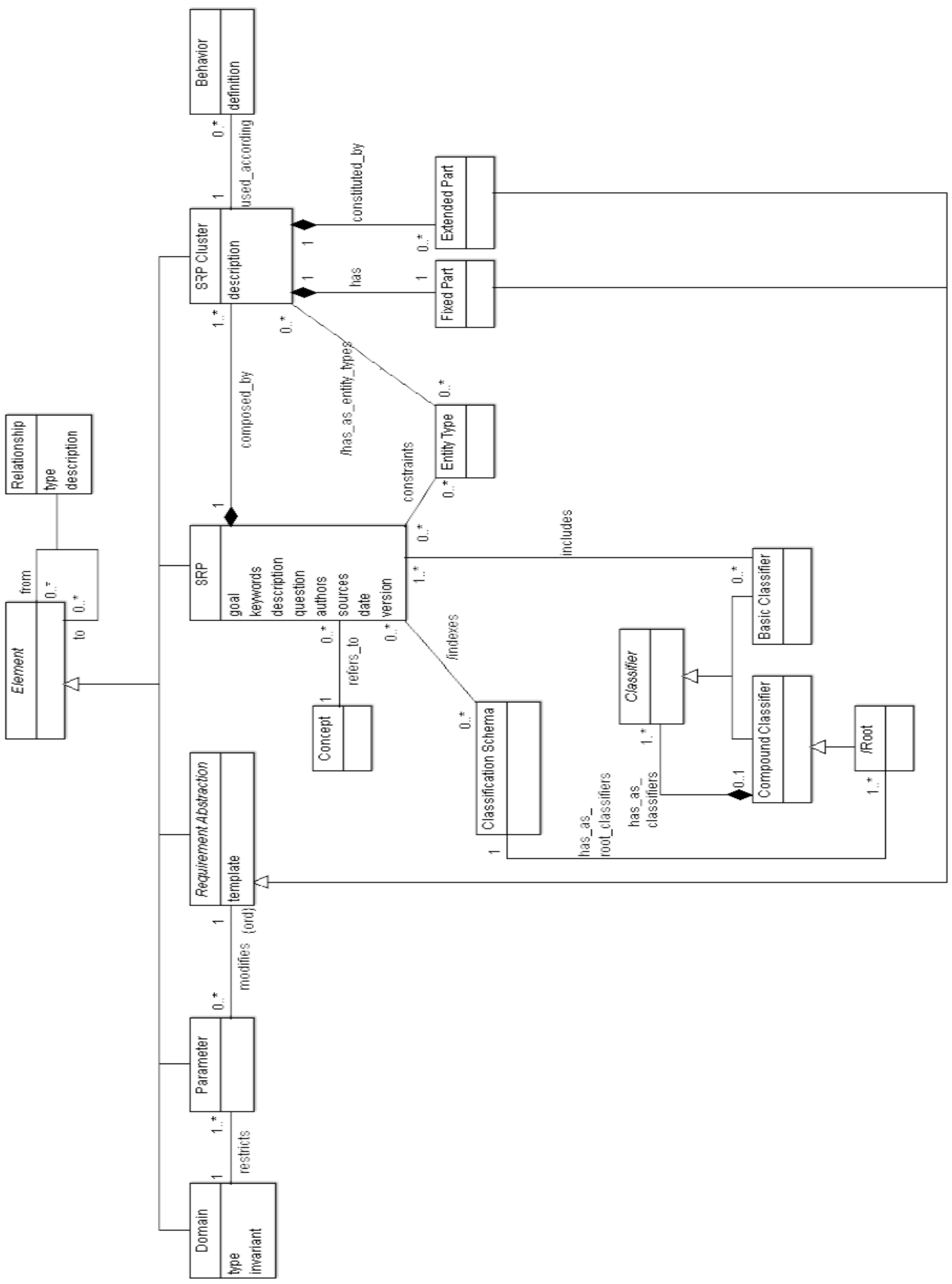
Appendix 3: SRP Metamodel

This appendix contains the whole metamodel of SRPs and their organization in a catalogue, as well as the glossary with all the concepts appearing on it.

Glossary of the SRP Metamodel

- Basic classifier. Category in the lower level of a classification schema (i.e. indexing SRPs).
- Behavior. Constraint to which the fixed and extended parts of an SRP cluster must conform to.
- Classification schema. Taxonomy that organizes SRPs.
- Classifier. Category in a classification schema.
- Compound classifier. Category in the middle level of a classification schema (i.e. containing other classifiers); it is used to create a hierarchical structure of classifiers.
- Concept. Main aspect to which an SRP refers to.
- Domain. Valid values that a parameter can take when applying in a project the requirement abstraction to which the parameter is associated.
- Element. Generalization of the atomic components of SRPs that could be involved in relationships.
- Entity type. General aspect of an IT project restricted by an SRP.
- Extended part. Requirement abstraction of an SRP cluster that allows defining a precise requirement, providing more detail to the fixed part of that same SRP cluster.
- Fixed part. Requirement abstraction of an SRP cluster that allows defining the minimal requirement that always holds in the SRP cluster.
- Parameter. Variable part in a requirement abstraction that takes a specific value of its domain during the application of that requirement abstraction in a project.
- Relationship. Connection, association, or involvement among two different elements of SRPs.
- Requirement abstraction. Generalization of the parts of an SRP cluster, which may be a fixed or extended part.
- Root. Category in the upper level of a classification schema.
- SRP. Pattern that, when applied in a project, produces software requirements that foster the achievement of the goal of that pattern.
- SRP cluster. Group of requirement abstractions that allow defining requirements to achieve the goal of that SRP using a specific solution.

SRP Metamodel


Behavior
definition
used_according
SRP Cluster
description
constituted_by
Extended Part
has
Fixed Part
/has_as_entity_types
Entity Type
composed_by
constraints
Relationship
type
description
from
to
Element
SRP
goal
keywords
description
question
authors
sources
date
version
includes
Basic Classifier
refers_to
/indexes
Classifier
Concept
Compound Classifier
/Root
Classification Schema
has_as_
root_classifiers
has_as_
classifiers
Requirement Abstraction
template
modifies
{ord}
Parameter
restricts
Domain
type
invariant
1
0..*
1..*
0..1