Information and Software Technology xxx (2014) xxx–xxx

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Change impact analysis for requirements: A metamodeling approach

Arda Goknil [a,*], Ivan Kurtev [b], Klaas van den Berg [c], Wietze Spijkerman [c]

[a] SnT Centre, University of Luxembourg, Luxembourg
[b] Nspyre, Dillenburgstraat 25-3, 5652 AM Eindhoven, The Netherlands
[c] Software Engineering Group, University of Twente, 7500 AE Enschede, The Netherlands

ARTICLE INFO

Article history:
Received 9 October 2013
Received in revised form 11 March 2014
Accepted 12 March 2014
Available online xxxx

Keywords:
Requirements metamodel
Change impact analysis
Proposing and propagating changes

ABSTRACT

Context: Following the evolution of the business needs, the requirements of software systems change continuously and new requirements emerge frequently. Requirements documents are often textual artifacts with structure not explicitly given. When a change in a requirements document is introduced, the requirements engineer may have to manually analyze all the requirements for a single change. This may result in neglecting the actual impact of a change. Consequently, the cost of implementing a change may become several times higher than expected.
Objective: In this paper, we aim at improving change impact analysis in requirements by using formal semantics of requirements relations and requirements change types.
Method: In our previous work we present a requirements metamodel with commonly used requirements relation types and their semantics formalized in first-order logic. In this paper the classification of requirements changes based on structure of a textual requirement is provided with formal semantics. The formalization of requirements relations and changes is used for propagating proposed changes and consistency checking of proposed changes in requirements models. The tool support for change impact analysis in requirements models is an extension of our Tool for Requirements Inferencing and Consistency Checking (TRIC).
Results: The described approach for change impact analysis helps in the elimination of some false positive impacts in change propagation, and enables consistency checking of changes.
Conclusion: We illustrate our approach in an example which shows that the formal semantics of requirements relations and change classification enables change alternatives to be proposed semi-automatically, the reduction of some false positive impacts and contradicting changes in requirements to be determined.

1. Introduction

At the present day, software systems get more and more complex. Following the evolution of the business needs, the requirements of software systems change continuously and new requirements emerge frequently. The integration of the new and/or modified requirements with the existing ones poses a need for adapting the architecture and source code of the software system in order to satisfy the new set of requirements. The integration of the new/modified requirements and adaptations to the software system is called *change management*. The size and complexity of software systems make change management costly and time consuming. It has been reported that 85–90 percent of software system budgets goes to operation and maintenance [27]. To reduce the cost of changes, it is important to apply change management as early as possible in the software development cycle. When changes in the requirements are proposed, the impact of these changes on other requirements, design elements and source code needs to be analyzed in order to determine parts of the software system to be changed. Determining the impact of changes on other development artifacts is called *change impact analysis*. In this paper we focus on change impact analysis in requirements only.

When a change is introduced to a requirement, the requirements engineer needs to find out if any other requirement related to the changed one is impacted. The structure of the requirements specification is crucial in this process. In practice, requirements documents are often textual artifacts with implicit structure. Most relations among requirements are not encoded explicitly [48]. It is time consuming and error prone to manually identify these

* Corresponding author. Tel.: +352 4666445572.
E-mail addresses: arda.goknil@uni.lu (A. Goknil), ivan.kurtev@nspyre.nl (I. Kurtev), vdberg.nl@gmail.com (K. van den Berg), weedz@frummel.org (W. Spijkerman).

http://dx.doi.org/10.1016/j.infsof.2014.03.002
0950-5849/

relations and to follow them in order to determine impacted requirements. Several commercial tools use a semi-structured format of requirements documents and provide support for automatic change impact analysis. IBM Rational RequisitePro [44] and DOORS [45] are some of the most commonly used tools with such support. Although these tools represent requirements relations explicitly, the provided information is often too general and the analysis results may be not satisfactory. For example, when a requirement is changed in RequisitePro, relations of the changed requirement are marked as *suspect* by the tool. RequisitePro provides two general relation types that indicate only the direction of relations: *traceFrom* and *traceTo*. These types do not indicate the meaning of the dependency between requirements. All requirements reachable from the changed requirement (by following the *suspect* relations) are potentially impacted. Without semantic information that precisely determines how a change propagates through the relations, the requirements engineer generally has to assume the worst case scenario where all related requirements are impacted. This type of analysis often produces a high number of false positive impacted elements.

Bohner [5–7] describes this situation as *explosion of impacts without semantics*. He states that change impact analysis must employ additional semantic information to increase the accuracy by eliminating false positives.

Several approaches use models to express requirements specifications. Most of them follow the goal-oriented paradigm [91], for example *i** and KAOS, where requirements are given as goal models. In these approaches, requirements relations have well defined semantics that allow a higher precision in change impact analysis. Unfortunately, this way of specification is not yet a part of the mainstream industrial practice. Many software companies still rely on specifications in the form of textual documents and use cases. In this paper we describe change impact analysis techniques on requirements that are still specified textually but are extended with several types of well-defined requirements relations.

In our previous work [38,36], we use a representation of requirements and the relations among them as models conforming to a requirements metamodel. The metamodel contains concepts commonly found in the literature and that reflect how most requirements documents are structured. The most important elements in the metamodel are requirements relations and their types. The semantics of these elements is given in First Order Logic (FOL) and allows two activities. First, new relations among requirements can be inferred from the initial set of relations. Second, requirements models can be automatically checked for consistency of the relations. Tool for *Requirements Inferencing and Consistency Checking* (TRIC) [38,92] is developed to support both activities. The details about the metamodel, the semantics and the tool are already reported in [38]. In [38] we also thoroughly study how to manually identify and assign initial relations between requirements. In this paper we extend these results with a technique for change impact analysis in requirements models.

The technique uses the formal semantics of requirements relations and a classification of requirements changes. Three activities for impact analysis are supported. First, the requirements engineer proposes changes according to the change classification before implementing the actual changes. Second, the requirements engineer identifies the propagation of the changes to related requirements. Possible alternatives in the propagation are determined based on the semantics of change types and requirements relations. Third, possible contradicting changes are identified. We extended the tool TRIC to support these activities. The tool automatically determines the change propagation paths, checks the consistency of the changes, and suggests alternatives for implementing the changes. There are different rationales for requirements changes (e.g., *refactoring* and *domain changes*). Our focus in this paper is the requirements changes fostered by the evolution of the business needs since they have an impact on other software development artifacts like software architecture, detailed design and source code. We name these changes as *domain changes*. The described approach for change impact analysis helps in the elimination of some false positive impacts in change propagation, and enables consistency checking of changes.

This paper is organized as follows. In Section 2 we illustrate an example impact analysis with and without using semantics. Section 3 briefly introduces the elements of requirements models. This is required for understanding the impact analysis process. Section 4 presents the classification of changes in requirements. Section 5 explains our solution for change impact analysis based on change propagation and consistency checking. The tool support for change propagation activities is illustrated in Section 6. Section 7 illustrates the approach with an example. In Section 8 we give a brief discussion for the overall approach. Section 9 compares our work with the existing results. Section 10 concludes the paper.

2. The importance of semantics for change impact analysis

We perform impact analysis on an example problem with and without using semantics of requirements relations. Table 1 gives some requirements in a requirements document for a Course Management System (CMS) used as a working example in this paper. The CMS is used in educational institutions and has features such as notification of students about exam grades and messaging for communication. The CMS requirements document contains 122 requirements in total. Part of the document is given in Appendix A.

Assume that the requirements and their relations are handled in a tool like IBM RequisitePro, that is, only very generic relation types like *traceFrom* and *traceTo* can be assigned. Fig. 1 shows the requirements directly and indirectly related to the requirement R7 with a distance 2. The distance is the number of relations between two requirements. The relations have to be assigned by the requirements engineer.

We will analyze the following change to R7.

> **R7**: The system shall provide a messaging facility.
> Proposed Change is the following.
> **Change**: Add a constraint to the property messaging facility
> **Description of Change**: Messaging facility should also contain sms and e-mail features

For this change, R18 is directly impacted and R74 has an indirect impact (depending on the change in R18). By using the transitive closure of the relations we can obtain all indirectly impacted requirements for a given change. It is clear that if the graph of the requirements model is connected then every requirement will be indirectly impacted by any change in any other requirement. In general, the distance between software artifacts has been proposed as a measure for the chance to have an impact [5]. The notion of distance explains how the number of impacts explodes for requirements (see Fig. 2).

Fig. 2 shows some of the requirements directly/indirectly related to R7 at distances of 1, 2, 3 and 4. The requirements indirectly related to R7 at distances of 2, 3 and 4 (see Fig. 2(b), (c) and (d)) are candidates for inspection in RequisitePro. As can be seen, the number of potentially impacted requirements (or *suspects*) might rapidly grow. The manual inspection may be unfeasible for large documents. In the following we give the analysis of two requirements directly related to R7.

Please cite this article in press as: A. Goknil et al., Change impact analysis for requirements: A metamodeling approach, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.03.002

Table 1
Some requirements for the CMS.

R1: The system shall provide static course information
R4: The system shall provide dynamic course information
R7: The system shall provide a messaging facility
R8: The system shall enable students to retrieve contact information of students and lecturers of subscribed courses
R11: The system shall enable students to subscribe/unsubscribe to courses
R16: The system shall allow messages to be sent to individuals, teams, or all course participants at once
R17: The system shall allow students to create teams
R18: Teams are created by students inviting other students in the same course using the messaging system
R24: The system shall notify students about events (new messages posted, team invites, scheduled exams etc.)
R25: The system shall allow students to customize the notification behavior
R72: The system shall allow only lecturers to manage student teams
R74: The system shall allow only lecturers to create new teams
R117: The system shall allow the student office to ask the students to evaluate courses by means of a web-survey

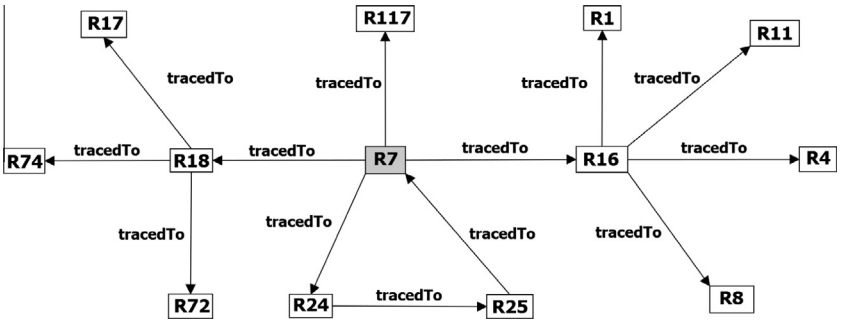


Fig. 1. Requirements related to R7 with distance of 2 in RequisitePro.

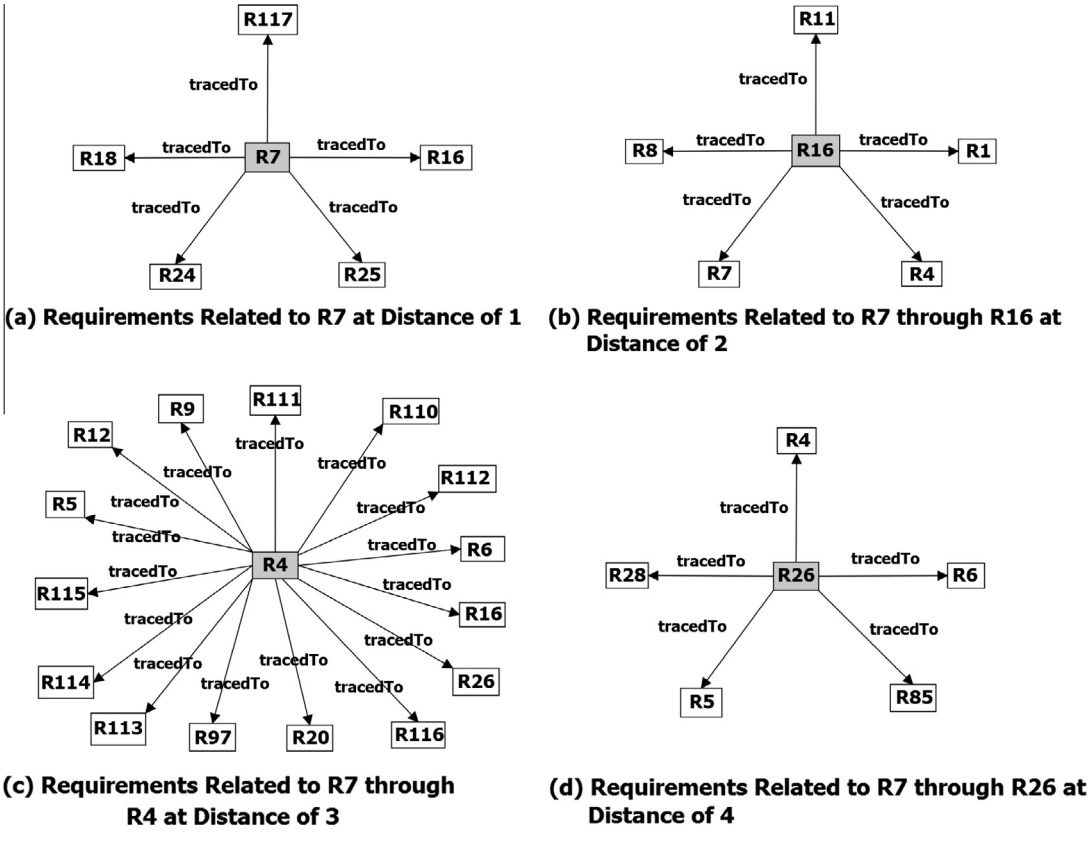


Fig. 2. Some of the requirements directly/indirectly related to R7 in RequisitePro.

Please cite this article in press as: A. Goknil et al., Change impact analysis for requirements: A metamodeling approach, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.03.002

R16: The system shall allow messages to be sent to individuals, teams, or all course participants at once.
R18: Teams are created by students inviting other students in the same course using the messaging system.

Sending messages via either sms or e-mail does not have any impact on the creation of teams. Therefore, R18 is not impacted by the change in R7. R16 is a refined version of R7. It adds details about who should receive messages. The new constraint may have an impact on R16. For example, if a mobile phone number is optional in the system, it may not be possible to send an sms message.

In this example for some cases changes are not propagated via a given relation. The meaning of the relation between R7 and R18 together with the type of the change in R7 can determine if there is any propagation. R18 requires R7 (a need for messaging) and the change in R7 does not remove the general messaging facility. Therefore we can conclude that the change in R7 does not lead to any changes in R18.

The example shows the importance of semantics of relations for change impact analysis. The impact analysis without semantics results in false positives. Using semantics, some of the false positives are removed. The challenge is to capture the semantics, the type of change and how they combine in order to detect change propagation.

3. Requirements metamodel

Our requirements metamodel contains common entities identified in the literature for requirements models. In order to construct our metamodel we investigated and benefited from several approaches which are commonly used to define and represent requirements: goal-oriented [91,67], aspect-driven [76], variability management [66], use-case [19], domain-specific [74,50], and reuse-driven techniques [60]. The main elements in the requirements metamodel are *Requirement* and *Relationship* (see Fig. 3). The metamodel defines the *Requirement* entity with its attributes and relations between requirements. In this paper we do not show other entities such as goals, stakeholders, and test cases.

Requirements and their *relations* are captured in a *requirements model*. Based on [85], a requirement is defined as follows:

Definition 1. *Requirement: A requirement* is a description of a system property or properties which need to be fulfilled.

We identified five types of relations: *requires*, *refines*, *contains*, *partially refines*, and *conflicts*. These relations are informally defined as follows.

Definition 2. *Requires relation:* A requirement R_1 *requires* a requirement R_2 if R_1 is fulfilled only when R_2 is fulfilled.

The required requirement can be seen as a pre-condition for the requiring requirement.

Definition 3. *Refines relation:* A requirement R_1 *refines* a requirement R_2 if R_1 is derived from R_2 by adding more details to its properties.

The refined requirement can be seen as an abstraction of the refining requirements.

Definition 4. *Contains relation:* A requirement R_1 *contains* requirements $R_2 \ldots R_n$ if $R_2 \ldots R_n$ are parts of the whole R_1 (part-whole hierarchy).

This relationship enables a complex requirement to be decomposed into parts. A composite requirement may state that the system shall do *A* and *B* and *C*, which can be decomposed into the requirements that the system shall do *A*, the system shall do *B*, and the system shall do *C*. For this relation, all parts are required in order to fulfill the composing requirement.

Definition 5. *Partially refines relation:* A requirement R_1 *partially refines* a requirement R_2 if R_1 is derived from R_2 by adding more details to properties of R_2 and excluding the unrefined properties of R_2.

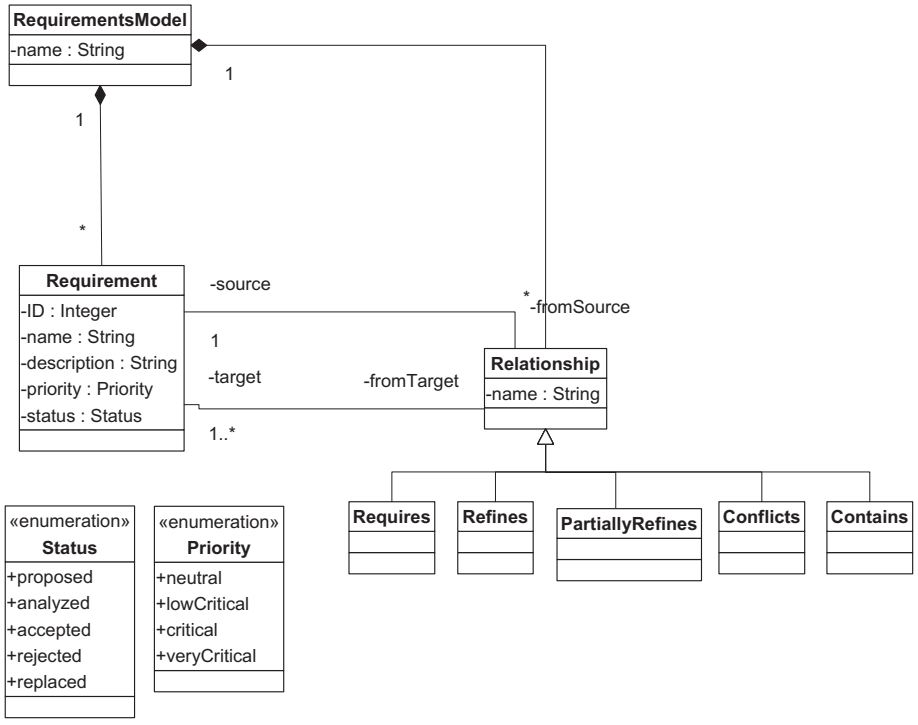


Fig. 3. Requirements metamodel.

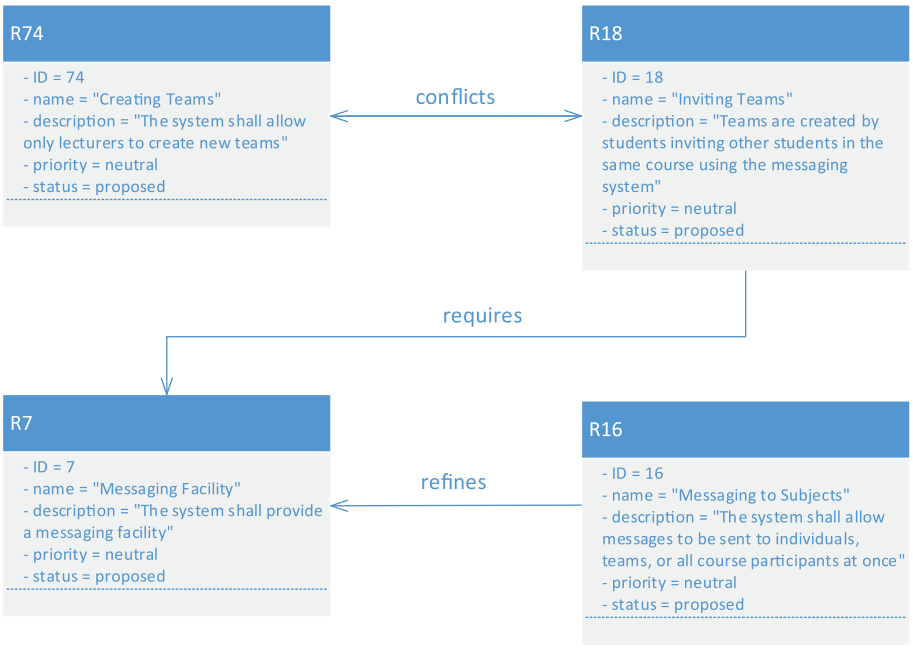


Fig. 4. Instance of the requirements metamodel for some CMS requirements.

Our assumption here is that R_2 can be decomposed into other requirements and that R_1 refines a subset of these decomposed requirements. This relation can be described as a combination of *contains* and *refines* relations. It is mainly drawn from the decomposition of goals (and-decomposition) in goal-oriented requirements engineering [91].

Definition 6. *Conflicts relation:* A requirement R_1 *conflicts with* a requirement R_2 if the fulfillment of R_1 excludes the fulfillment of R_2 and vice versa.

In this paper, we consider *conflicts* as a binary relation. Our approach can be extended to *n*-ary conflicts relations, that is, conflicts among multiple requirements, as a whole without excluding pairs of requirements to be fulfilled.

The definitions given above are informal. Supplementary Material A[1] presents the formal semantics of requirements and relations. The reader is also referred to [32,38] for the formalization.

Fig. 4 gives an instance of the requirements metamodel for some of the CMS requirements.

> **R7**: The system shall provide a messaging facility.
> **R16**: The system shall allow messages to be sent to individuals, teams, or all course participants at once.
> **R18**: Teams are created by students inviting other students in the same course using the messaging system.
> **R74**: The system shall allow only lecturers to create new teams.

It should be noted that realistic requirements specifications contain additional elements like definitions, dictionaries, assertions, identity, etc. [95]. The model used here is a simplification built for the purpose of analyzing requirements which represent system properties and their relations for change impact analysis.

4. Classification of changes in requirements

Our technique for change impact analysis uses a classification of changes in requirements and the semantics of requirements relations. Requirements changes are analyzed and classified based on an assumption about a very general structure of textual requirement. The change types are formalized by giving their effects in terms of changes in the formula that represents a requirement. For readability in this section we give the classification of changes with the semantics of only one change type. We discuss the rationale of changes at the end of the section. The formal semantics of the change types and their rationale can be found in Supplementary Material B.[2] The reader is also referred to [32] for the formalization.

4.1. Structure of a textual requirement

We need to consider a general enough structure of a requirement to determine the granularity of changes that can be applied. Our definition of a requirement is "a textual requirement is a description of a property or properties which must be exhibited by the system" (see Fig. 5).

> **Example**: *Structure of Requirement based on the definition*
> **R98**: The system shall allow only the administration to create new courses.
> We can identify the following structure of R98 by following Fig. 5:
> **Property**: The system shall provide the functionality of creating courses to only the administration
> **Constraint**: Only the administrator users can create courses

[1] http://www-sop.inria.fr/members/Arda.Goknil/impact/SupplementaryMaterialA.pdf.

[2] http://www-sop.inria.fr/members/Arda.Goknil/impact/SupplementaryMaterialB.pdf.

Please cite this article in press as: A. Goknil et al., Change impact analysis for requirements: A metamodeling approach, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.03.002

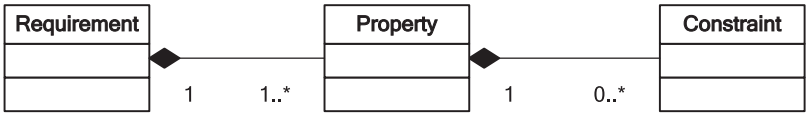


Fig. 5. Structure of a textual requirement based on the definition of a requirement.

Table 2
Requirements change types.

Change types
• Add a new requirements relation
• Delete requirements relation
• Update requirements relation
• Add a new requirement
• Delete requirement
• Update requirement
– Add property to requirement
– Add constraint to property of requirement
– Change property of requirement
– Change constraint of property of requirement
– Delete property of requirement
– Delete constraint of property of requirement

4.2. Change types for requirements models

The change types for requirements models are derived from the structure in Fig. 5 and from the requirements metamodel in Fig. 3 (see Table 2).

The first five change types in Table 2 are obvious manipulations over the requirements model. The subtypes of 'Update Requirement' are obtained from the structure of a textual requirement in Section 4.1.

4.2.1. Update requirement

We use the symbol $\mapsto$, to denote updates in requirements in the following way: $R \mapsto R^l$ denotes a change where R is the requirement before the change and R^l is the requirement after the change. The subtypes of 'Update Requirement' are denoted by using a notation over the symbol $\mapsto$. Update of a requirement R is done:

- By adding the property pt to the requirement R, denoted as $R \overset{+pt}{\mapsto} R^l$.
- By deleting the property pt of the requirement R, denoted as $R \overset{-pt}{\mapsto} R^l$.
- By changing the property pt of the requirement R with the property pt^l, denoted as $R \overset{pt \mapsto pt^l}{\mapsto} R^l$.
- By adding the constraint ct to the property pt of the requirement R, denoted as $R \overset{+ct}{\mapsto} R^l$.
- By deleting the constraint ct of the property pt of the requirement R, denoted as $R \overset{-ct}{\mapsto} R^l$.
- By changing the constraint ct of the property pt of the requirement R with the constraint ct^l, denoted as $R \overset{ct \mapsto ct^l}{\mapsto} R^l$.

4.3. Semantics of requirements changes

The semantics of the changes is given in first-order logic (FOL). Requirements are interpreted as formulae in a fragment of FOL where all the formulae are in a conjunctive normal form (CNF). In the following we sketch the formalization of requirements and we give the semantics of the change "Delete Property of Requirement". For the complete description of the semantics, the reader is referred to Supplementary Material B.[2]

Formalization of Requirements

We assume the general notion of requirement being "a property which must be exhibited by a system". We express the property as a formula P in FOL. We assume that requirements can always be expressed in the universal fragment of FOL and a requirement is expressed as a formula $\forall x \varphi$ with φ in conjunctive normal form (CNF). If the formula φ is a closed formula, then the universal quantifiers can be dropped. It is also possible that the formula contains free variables.

According to the model theoretic semantics of FOL the truth value of P is determined in a *model* $\mathcal{M}$ by using an interpretation for the function and predicate symbols in P.

Let $\mathcal{F}$ be a set of function symbols and P a set of predicate symbols, each symbol with a fixed arity. A model $\mathcal{M}$ of the pair $(\mathcal{F}, \mathcal{P})$ consists of the following items [43]:

- a non-empty set $\boldsymbol{A}$, the universe of concrete values
- for each $\in \mathcal{F}$ with n arguments, a function $^{\mathcal{M}}$: $A^n \to A$
- for each P $\in \mathcal{P}$ with n arguments, a set $\mathrm{P}^{\mathcal{M}} \subseteq A^n$.

A satisfaction relation between the model $\mathcal{M}$ and the formula P holds:

(1) $\mathcal{M} \mid =_\ell \mathrm{P}$

if P evaluates to True in the model $\mathcal{M}$ with respect to the environment ℓ (i.e., a look-up table for free variables in P). The model $\mathcal{M}$ together with ℓ in which P is true represents a system s that satisfies the requirement. From now on, all the formulae P that express properties will be in the form where $(\forall x = \forall x_1 \forall x_2 \ldots \forall x_k)$:

(2) $\mathrm{P} = \forall x (p_1 \wedge \cdots \wedge p_n)$, where $n \geqslant 1$

p_n is a disjunction of literals which are atomic formulae (atoms) or their negation. An atomic formula is a predicate symbol applied over terms. In the rest of the paper we use the notation $(p_1 \ldots p_n)$ for $(p_1 \wedge \cdots \wedge p_n)$.

Delete Property of Requirement

Let R be the requirement before deleting the property pt, and R^{l} be the requirement after deleting the property pt. P and P^{l} are formulae for R and R^{l}. P is in conjunctive normal form as follows:

(3) $\mathrm{P} = \forall x \, ((\mathrm{p}_1 \ldots \mathrm{p}_n) \wedge (\mathrm{q}_1 \ldots \mathrm{q}_m)); \quad \mathrm{m}, \mathrm{n} \geq 1$

$R \overset{-pt}{\mapsto} R^l$ iff P^{l} is derived from P such that the following two statements hold:

(4) $\mathrm{P}^{\mathrm{l}} = \forall x \, (\mathrm{p}_1 \ldots \mathrm{p}_n); \quad \mathrm{n} \geq 1$

(5) $(\neg(\mathrm{P}^{\mathrm{l}} \to \mathrm{P}))$ is satisfiable

where $\forall x (\mathrm{q}_1 \ldots \mathrm{q}_m)$ denotes the property that is captured in pt.

If every bounded occurrence of a variable is removed by deleting the property, then the quantifier for the variable is removed as well. Please note that if the requirement R is written as a formula $\forall x (\varphi \wedge \psi)$ with $(\varphi \wedge \psi)$ in CNF and P_{pt} (for the property captured in pt) is expressed as $\forall x \psi$ with ψ in CNF, we understand the following: $R \overset{-pt}{\mapsto} R^l$ iff $(\mathrm{P}^{\mathrm{l}} = \forall x \, \varphi)$, and $(\neg(\forall x (\varphi \to (\varphi \wedge \psi))))$ is satisfiable.

Please cite this article in press as: A. Goknil et al., Change impact analysis for requirements: A metamodeling approach, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.03.002

In the following we give the semantics of the change "Delete Property of Requirement".

4.4. Rationale of changes

The change types in Section 4.2 do not address why a change needs to be performed in the requirements model, that is, what is the rationale of changes. The impact of changes depends on their rationale. For instance, the requirements engineer may delete a property of a requirement because this property is not required any more from the business/stakeholder point of view. The property may be in other requirements in the model and it also has to be deleted from them. The requirements engineer may delete a property of a requirement to improve the structure of the model without modifying the overall system properties. In this situation, the property must still hold in the requirements model after the change. The property has to be kept at least in one of the requirements. Therefore, we need to know the rationale of requirements changes in order to determine the impact of changes in the whole model. We classify the rationale of changes as *refactoring* and *domain changes*.

Buckley et al. [12] classify changes as *semantics-preserving* and *semantics-modifying*. This classification concerns the semantics of software components, such as type hierarchy, scoping, visibility, accessibility, and overriding relationships. We adapt their classification for requirements changes.

Van Lamsweerde [54] introduces requirements description qualities such as good structuring and modifiability. The requirements engineer may change the requirements model to improve the quality of the requirements description. For instance, a requirement may be decomposed into multiple requirements. These changes are *semantics-preserving* according to [12] and we consider their rationale as *refactoring* (see [30] for refactoring).

The evolution of requirements also foster changes to the requirements model. We name these changes and their rationale as *domain changes*. With the term 'domain' we mean the problem/business domain. Consider a requirements model that contains a set of requirements for an online banking system in Europe. Here, the domain is banking and a change request to adapt the system to the USA is received. Then, all currency requirements in the domain of banking are changed and these changes should be reflected in the requirements model.

Refactoring. Refactoring is a change (changes) in the requirements model in order to improve the structure of the model without modifying the overall system properties [30]. Changes to the model caused by refactoring do not affect the properties in the whole requirements model.

Domain changes. Domain changes are the changes in the requirements model in order to modify the overall system properties. Changes to the model caused by domain changes do affect the properties in the whole requirements model.

The rationale of changes is important since it is a factor in determining the change alternatives for the change propagation. The usage of rationale in change propagation is illustrated in the next section.

5. Change propagation and change consistency checking

Change propagation is a process of deducing new proposed changes in a requirements model based on an initial set of proposed changes. The requirements relations and the change types are used to determine if a change in a requirement has an impact (is propagated) on the directly related requirements. *Change consistency checking* identifies contradicting proposed changes. In case of several paths from one requirement to another the propagation of changes via different paths can possibly result in contradicting proposed changes on a single requirement.

The focus in this paper is on domain changes since they have an impact on other software development artifacts like software architecture, detailed design and source code. These changes are fostered by the evolution of the business needs in order to modify the overall system properties. By using the formal semantics of requirements, relations and changes, it is possible to derive whether or not (possible) impacts are caused by a change.

The change propagation mechanism uses a change impact function. The change impact function takes a change type and a requirement to which the change is introduced as input, and produces a set of decision trees as output. A decision tree contains alternative changes identified during the change propagation by traversing the requirements model. The following is the definition of the change impact function:

$impact: SCT \times SR \times SSRR \rightarrow SSDT$

where *SCT* is the set of change types, *SR* is the set of requirements, *SSRR* is the set of sets of requirements relations, and *SSDT* is the set of sets of decision trees for changes.
A decision tree is expressed as a sentence in a language with the following grammar.

```
<DT-C> ::= <Change > | <Change > <And>''('' <DT-C> '')'' |
      <DT-C> <Boolean-Operator> <DT-C> | ''('' <DT-C> '')''
<Change> ::= <Change-Type> ID
<Change-Type> ::= ''No Impact in'' | ''Delete Requirement'' |
      ''Delete Property of Requirement'' |
      ''Delete Constraint of Property of Requirement'' |
      ''Add Requirement'' | ''Add Property to Requirement'' |
      ''Add Constraint to Property of Requirement'' |
      ''Change Property of Requirement'' |
      ''Change Constraint of Property of Requirement'' |
      ''Add Relation'' | ''Delete Relation''
<Boolean-Operator> ::= <Exclusive-or> | <And>
<Exclusive-or> ::= ''|''
<And> ::= ''&''
ID denotes identifiers.
```

The reader is referred to Supplementary Material C[3] for the algorithm of the change impact function. In the following we explain the main steps of the algorithm with a simple model. Fig. 6(a) gives an example requirements model where the change 'Delete Property of Requirement' is proposed for the requirement R2. Fig. 6(b) shows four paths created while the change impact algorithm traverses the model for the change in R2.

Fig. 7 gives the decision trees for the model in Fig. 6(a). The operator *Exclusive-or* in the grammar is represented as the branches of the decision trees in Fig. 7 while the operator *And* in the grammar is "&" in the nodes of the decision trees. Each node contains change(s) with(out) the operator *And*.

The main steps in the change impact function algorithm are the following:

- *Creating a decision tree for each unvisited requirement related to the starting requirement*. In Fig. 6(b), the change 'Delete Property of Requirement' is introduced to the R2. The algorithm creates a decision tree (*Decision* Tree for Path 1, Decision Tree for Path 2 and *Decision Tree for Path 4* in Fig. 7) for each unvisited directly related requirement (R1, R3 and R4 in Fig. 6(b)). Decision trees have a starting node 'Delete Property of Requirement R2'.

[3] http://www-sop.inria.fr/members/Arda.Goknil/impact/SupplementaryMaterialC.pdf.

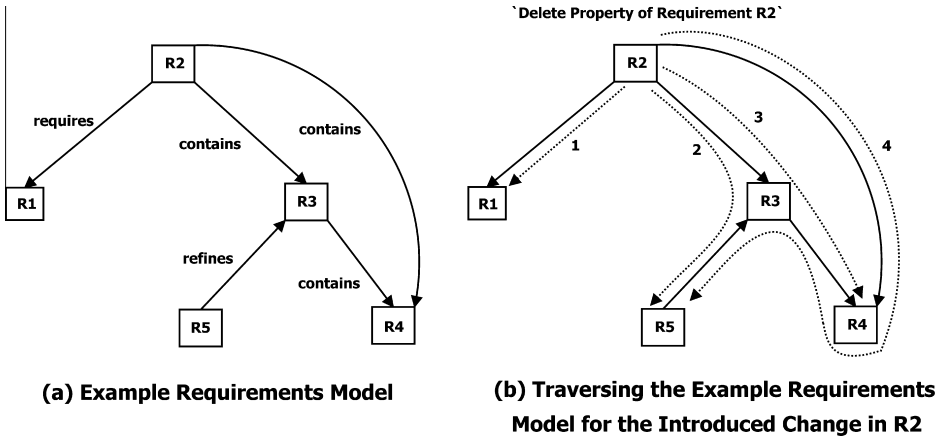


Fig. 6. Example requirements model and model traversal for the proposed change.

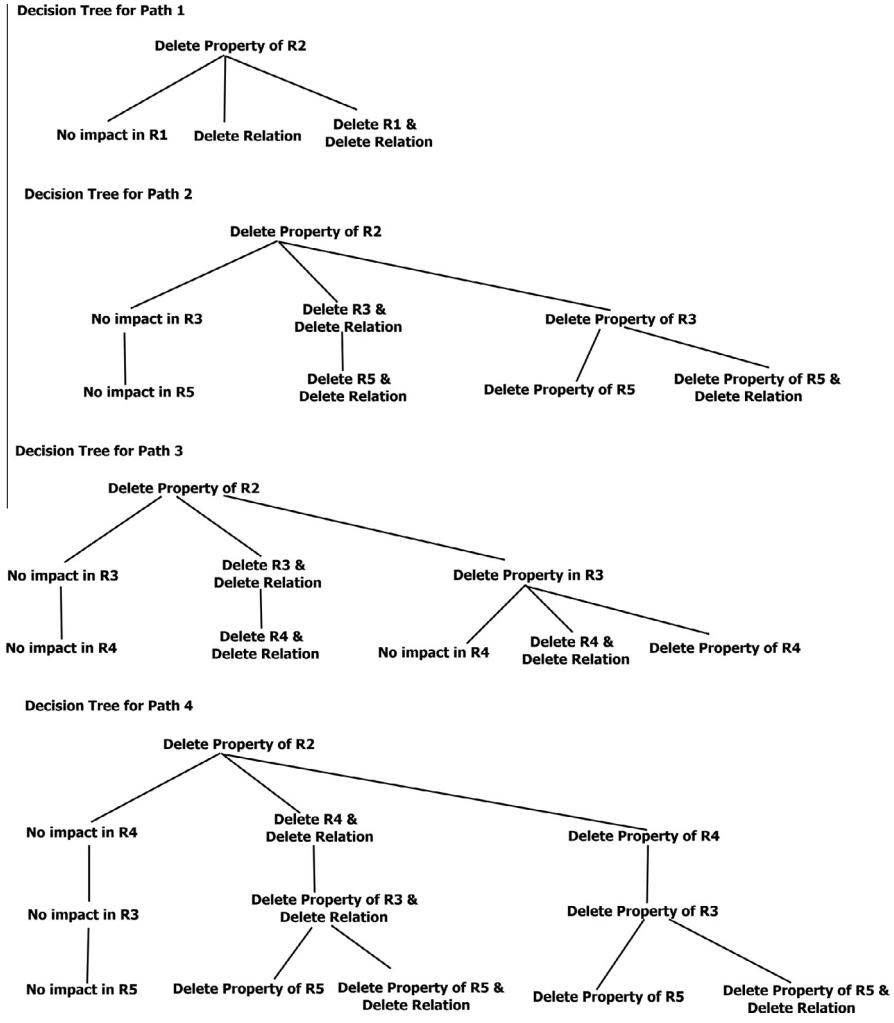


Fig. 7. Decision trees for the example model.

- *Propagating change for each unvisited related requirement*. Change alternatives are identified for the unvisited requirements (R1, R3 and R4) directly related to R2 in Fig. 6(b). For instance, R1 is related to R2 via the requires relation. The alternatives for propagating the change 'Delete Property of Requirement R2' from R2 to R1 are 'No impact in R1', 'Delete Relation' and 'Delete R1 & Delete Relation' (the *Decision Tree for Path 1* in Fig. 7). These alternatives are given in Table 3 where $(R_i \overset{-pt}{\mapsto} R_i^l)$ and (R_i requires R_k).
- *Expanding decision tree for each unvisited related requirement*. Each decision tree created for directly related requirements (*Decision* Tree for Path 1 for R2, *Decision Tree for Path 2* for R3 and *Decision Tree for Path 4* for R4 in Fig. 7) is expanded with alternative changes. For instance, the change alternatives 'No impact in R1', 'Delete Relation' and 'Delete R1 & Delete Relation' for R1 become the nodes of *Decision Tree for Path 1* in Fig. 7.

Please cite this article in press as: A. Goknil et al., Change impact analysis for requirements: A metamodeling approach, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.03.002

Table 3
Change impact alternatives for some change types with the domain change rationale.

Change types	Requirements relation types				
	R_i contains R_k	R_i refines R_k	R_i partially refines R_k	R_i requires R_k	R_i conflicts R_k
Add R_x	No impact	No impact	No impact	No impact	No impact
Delete R_i	Delete R_k & Delete relation	Delete R_k & Delete relation	Delete property of R_k	Delete relation | (Delete R_k & Delete relation)	Delete relation
$R_i \overset{+pt}{\mapsto} R_i^l$	No impact	Add property to R_k | Delete relation	Delete relation	No impact	No impact
$R_i \overset{-pt}{\mapsto} R_i^l$	No impact | Delete relation | (Delete R_k & Delete relation) | Delete property of R_k	Delete property of R_k | (Delete property of R_k & Delete relation)	Delete property of R_k	No impact | Delete relation | (Delete R_k & Delete relation)	No impact | Delete relation
$R_i \overset{pt_1 \to pt'}{\mapsto} R_i^l$	No impact | Change property of R_k	Change property of R_k | (Change property of R_k & Delete relation)	Change property of R_k | (Change property of R_k & Delete relation)	No impact | Delete relation | (Delete R_k & Delete relation)	No impact | Delete relation
$R_i \overset{+ct}{\mapsto} R_i^l$	No impact | Add constraint to property of R_k | Delete relation	No impact	No impact	No impact	No impact
$R_i \overset{-ct}{\mapsto} R_i^l$	No impact | Delete constraint of property of R_k	No impact | Delete relation | Delete constraint of property of R_k | (Delete constraint of property of R_k & Delete relation)	No impact | Delete relation | Delete constraint of property of R_k | (Delete constraint of property of R_k & Delete relation)	No impact | Delete relation | (Delete R_k & Delete relation)	No impact | Delete relation
$R_i \overset{ct_1 \to ct'}{\mapsto} R_i^l$	No impact | Change constraint of property of R_k	No impact | Change constraint of property of R_k	No impact | Change constraint of property of R_k	No impact | Delete relation | (Delete R_k & Delete relation)	No impact | Delete relation

- *Iterating*. Directly related requirements (R1, R3 and R4 in Fig. 6(b)) become the starting requirement and the algorithm is reinitiated for each of them. For R1, there is no unvisited directly related requirement and *Decision Tree for Path 1* in Fig. 7 is not expanded further. For R3, there are two unvisited directly related requirements (R4 and R5) and *Decision Tree for Path 2* is copied (see Decision Tree for Path 3). *Decision Tree for Path 2* is expanded with change alternatives for R5 and *Decision Tree for Path 3* is expanded with change alternatives for R4.

The output of the change impact function is a set of decision trees that contains all alternatives for a change propagated in the whole model. For instance, the output of the impact function for the proposed change in Fig. 6(a) is the set of decision trees in Fig. 7. Having decision trees is beneficial to generate all alternatives based on all possible propagation paths in the model and to see the overall impact of any change in the whole model. On the other hand, the size of the trees and their number can explode. The requirements engineer can also select among the change alternatives to propagate the change from one requirement to another step-by-step. Therefore, our tool gives an option to propagate the change step-by-step effectively by cutting branches of the tree. The tool (see Section 6) supports both decision tree generation and step-by-step change propagation. In the following we explain the change alternatives based on the semantics of change types, rationale of changes and requirements relations.

Change propagation. Change alternatives are determined based on the semantics of the change types, change rationale and requirements relations. Domain changes are the requirements changes fostered by the evolution of the business needs. For the change propagation in this paper we consider only the changes with the domain change rationale since they have an impact on other software development artifacts like software architecture, detailed design and source code. Table 3 gives the change impact alternatives for some of the change types with the domain change rationale. Each cell in the table gives change alternatives in order to propagate the changes in the rows by using the relations in the columns. The reader is referred to [32] for the complete table of the change impact alternatives.

The following is a change propagation example that illustrates the usage of alternatives in Table 3.

Change Propagation Example
Consider two requirements for the course management system:
R61: The system shall allow lecturers to specify enrolment policies based on grade, first-come first-serve (fcfs), and department.
R62: The system shall allow lecturers to specify enrolment policies based on grade.
where (R61 contains R62).
The stakeholder poses a change: specifying enrolment policies based on grade is not needed any more. One of the properties given in R61 is allowing lecturers to specify enrolment policies based on grade. Therefore, we propose the change 'Delete property of Requirement' for R61.
Proposed Change: Delete the Property of R61
Description of Change: Specifying enrolment policies based on grade is not needed any more.
The proposed change is propagated from R61 to R62 through the *contains* relation as follows:
Propagation from R61 to R62: According to Table 3 the alternatives to propagate the change 'Delete Property of R61' to R62 are (No impact | Delete R62 | Delete Property of R62).
The property to be deleted from R61 is specifying enrolment policies based on grade. It should also be deleted from R62. Since this property is the only property in R62, we choose the change 'Delete R62' among the change alternatives.

Please cite this article in press as: A. Goknil et al., Change impact analysis for requirements: A metamodeling approach, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.03.002

The following is the justification of change alternatives for R_k where $R_i \overset{-pt}{\mapsto} R_i^I$ and R_i *contains* R_k.

Change Alternatives:

Change alternatives for R_k where $(R_i \overset{-pt}{\mapsto} R_i^I)$ and $(R_i \text{contains} R_k)$

$=$ No impact | Delete R_k | Delete Property of R_k

Justification:

Let R_i and R_k be requirements. P_i and P_k are formulae for R_i and R_k.

$= \{By\ using\ formalization\ of\ the\ contains\ relation\}$

R_i contains R_k iff P_i is derived from P_k as follows:

$P_i = P_k \wedge P^I$

where $P_i = \forall x((p_1 \dots p_n) \wedge (q_1 \dots q_m))$; $m, n \geqslant 1$ and P^I denotes properties that are not captured in P_k

$= \{By\ using\ formalization\ of\ the\ change\ type\}$

$R_i \overset{-pt}{\mapsto} R_i^I$ iff P_i^I is derived from P_i as follows:

$P_i^I = \forall x(p_1 \dots p_n); \quad n \geqslant 1$

where $\forall x(q_1 \dots q_m)$ denotes properties that are captured in pt.

$= \{By\ using\ the\ formalization\ of\ domain\ changes\}$

Properties $\forall x(q_1 \dots q_m)$ that are captured in pt should be deleted from the requirements model RM.

$= \{By\ using\ formalization\ of\ the\ contains\ relation\}$

There are three alternatives for P_k and impact on R_k

(i) $P_k = \forall x(z_1 \dots z_t)$; $z \geqslant 1$, $\{z_1, \dots, z_t\} \subseteq \{p_1, \dots, p_n\}$ then *No Impact*.

(ii) $P_k = \forall x(q_1 \dots q_m)$; $m \geqslant 1$ then $\forall x(q_1 \dots q_m)$ should also be deleted. It means *Delete R_k & Delete Relation*.

(iii) $P_k = \forall x((z_1 \dots z_t) \wedge (q_1 \dots q_m))$; $t, m \geqslant 1$ then $\forall x(q_1 \dots q_m)$ should also be deleted. It means *Delete Property of R_k*.

All change alternatives given in Table 3 are derived from the semantics of the change types, the requirements relations and the rationale of changes. The reader is referred to Supplementary Material D[4] for the explanation of some of the change alternatives. The change propagation is implemented in a rule-based form in TRIC (see Section 6).

Proposed changes and propagated proposed changes may cause a conflict. In the following we explain how conflicts between proposed changes are identified.

Change consistency checking. Stakeholders may require changes that contradict with each other or the requirements engineer may propagate multiple changes to the same requirement in which the propagations cause a contradiction. Table 4 gives the pairs of contradicting changes based on the semantics of the change types and the domain change rationale. The rows and columns of the table are the change types. Two changes for the same requirement might cause a contradiction (the cells marked as *maybe* in Table 4) and these changes should be inspected, or there is a contradiction for sure (the cells marked as *yes*) caused by the changes. The cells in Table 4 are marked as *no* if there is no contradiction.

[4] http://www-sop.inria.fr/members/Arda.Goknil/impact/SupplementaryMaterialD.pdf.

The following is an example of an ensured inconsistency.

Ensured Inconsistency

R7: The system shall provide a messaging facility.

In the following there are two proposed domain changes for R7.

Proposed Change 1: Delete R7

Description of Proposed Change 1: There is no need for a messaging facility any more.

Proposed Change 2: Add Constraint to the Property of R7

Description of Proposed Change 2: An sms messaging facility should be provided.

The second change states that the sms messaging is a new constraint for the messaging facility while the first change states that the messaging facility is not needed at all. Therefore, there is an inconsistency for sure for the proposed changes (see Table 4).

The following is an example of a possible inconsistency.

Possible Inconsistency

R61: The system shall allow lecturers to specify enrolment policies based on grade, first-come first-serve (fcfs), and department.

The requirement R61 can be interpreted as three distinct properties: (i) allow lecturers to specify enrolment policies based on grade, (ii) allow lecturers to specify enrolment policies based on first-come first-serve (fcfs), and (iii) allow lecturers to specify enrolment policies based on department.

In the following there are two proposed domain changes for R61.

Proposed Change 1: Delete the Property of R61

Description of Proposed Change 1: There is no need of specifying enrolment policies based on grade any more.

Proposed Change 2: Add Constraint to the Property of R61

Description of Proposed Change 2: Lecturers should be allowed to specify enrolment policies based on departments only if they are affiliated with that department.

The first change states specifying enrolment policies based on grade is not needed any more. The second change states a constraint about the enrolment policies based on department. There is a need of checking if the two changes are referring to the same property or not. Since they refer to different properties, there is no inconsistency.

The reader is referred to Supplementary Material D[4] for the explanation of some of the contradicting changes. Table 4 is implemented in a rule based form in TRIC. The consistency rules are checked for the proposed and propagated changes (see Section 6).

6. Tool support

The tool TRIC was extended with features for change impact analysis in requirements [81]. The tool can be downloaded from [87]. In this section we describe the most important features of the tool: *proposing changes*, *propagating changes*, *displaying inconsistent proposed changes*, *implementing proposed changes in the requirements model*, and *predicting the impact of proposed changes*.

Proposing changes. TRIC provides a GUI for proposing changes which lists all the requirements in the model. The requirements

Table 4
Contradicting changes based on the semantics of the change types and the domain change rationale.

Change type	Delete R	$R \overset{+pt}{\mapsto} R^l$	$R \overset{-pt}{\mapsto} R^l$	$R \overset{pt_i \mapsto pt^l}{\mapsto} R^l$	$R \overset{+ct}{\mapsto} R^l$	$R \overset{-ct}{\mapsto} R^l$	$R \overset{ct_i \mapsto ct^l}{\mapsto} R^l$	No impact
Delete R	No	Yes	No	Yes	Yes	No	Yes	No
$R \overset{+pt}{\mapsto} R^l$	Yes	No	No	No	No	No	No	No
$R \overset{-pt}{\mapsto} R^l$	No	No	No	Maybe	Maybe	No	Maybe	No
$R \overset{pt_i \mapsto pt^l}{\mapsto} R^l$	Yes	No	Maybe	Maybe	Maybe	Maybe	Maybe	No
$R \overset{+ct}{\mapsto} R^l$	Yes	No	Maybe	Maybe	No	Maybe	Maybe	No
$R \overset{-ct}{\mapsto} R^l$	No	No	No	Maybe	Maybe	No	Maybe	No
$R \overset{ct_i \mapsto ct^l}{\mapsto} R^l$	Yes	No	Maybe	Maybe	Maybe	Maybe	Maybe	No
No impact	No	No	No	No	No	No	No	No

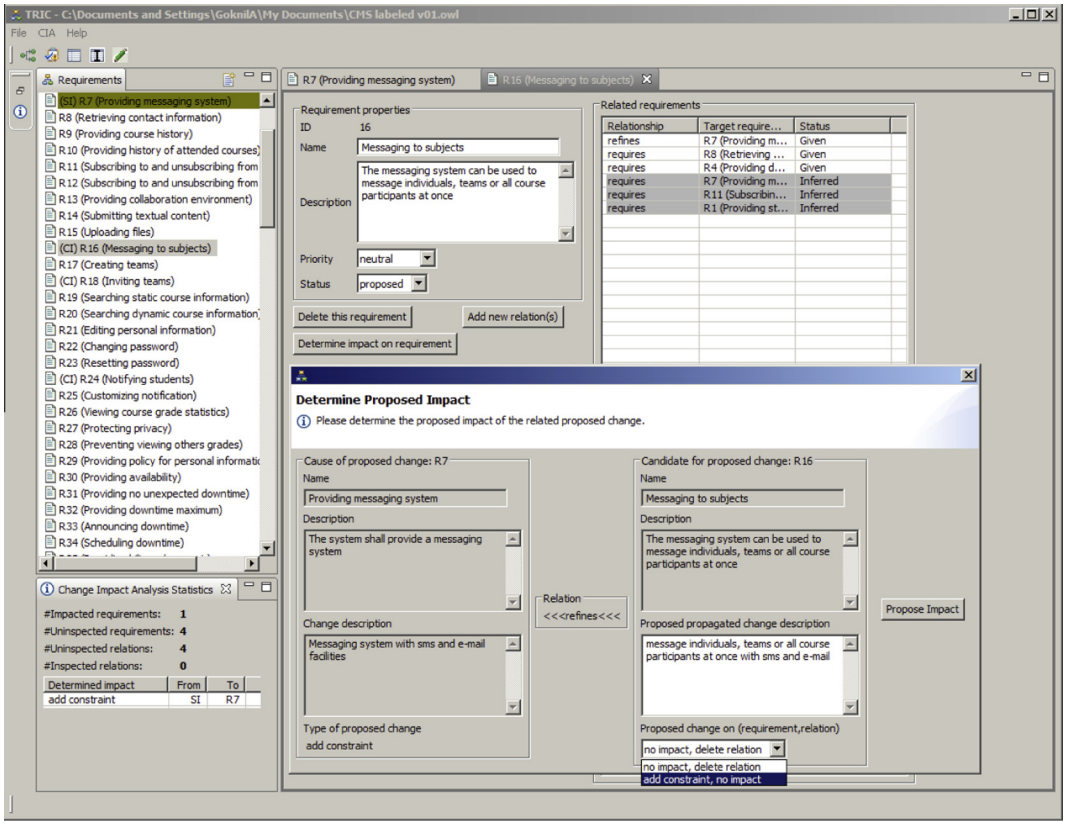


Fig. 8. GUI for propagating proposed changes.

engineer proposes a change with its type and description for the requirement selected among the listed requirements.

After proposing the change, the tool lists the candidate requirements to which the requirements engineer can propagate the change.

Propagating changes. Fig. 8 gives the GUI for change propagation. The *Determine Proposed Impact* window is opened by clicking on one of the candidate requirements (R16) for the impact. The change alternatives are provided to the requirements engineer by the tool based on the change impact alternatives for each relation given in Table 3.

TRIC also provides a matrix view in order to represent and propagate changes. Such a view is also available in commercial requirements management tools, such as RequisitePro in order to determine the impacted requirements.

The GUI for propagating proposed changes in Fig. 8 and the matrix view do not allow analyzing multiple proposed changes simultaneously. To support simultaneous analysis of multiple impact propagations, TRIC provides interactive decision tree builder (see Fig. 9).

Each arrow indicates a decision captured by the target node of the arrow. The decision tree can be expanded by making decisions (the *Make Decision* button). Once the analysis of the interactive decision tree is concluded, the requirements engineer can select a node and apply decisions captured by the path from the tree root to the selected node (the *Use Analysis* button).

Displaying inconsistencies in proposed changes. Fig. 10 gives the GUI for displaying inconsistent proposed changes.

The window in Fig. 10 has a list view including three columns. The first column gives the requirements that have contradicting proposed changes. The second column shows if the inconsistency is *certain* or *possible*. The third column lists the proposed changes that cause any inconsistency for the given requirement. The tool

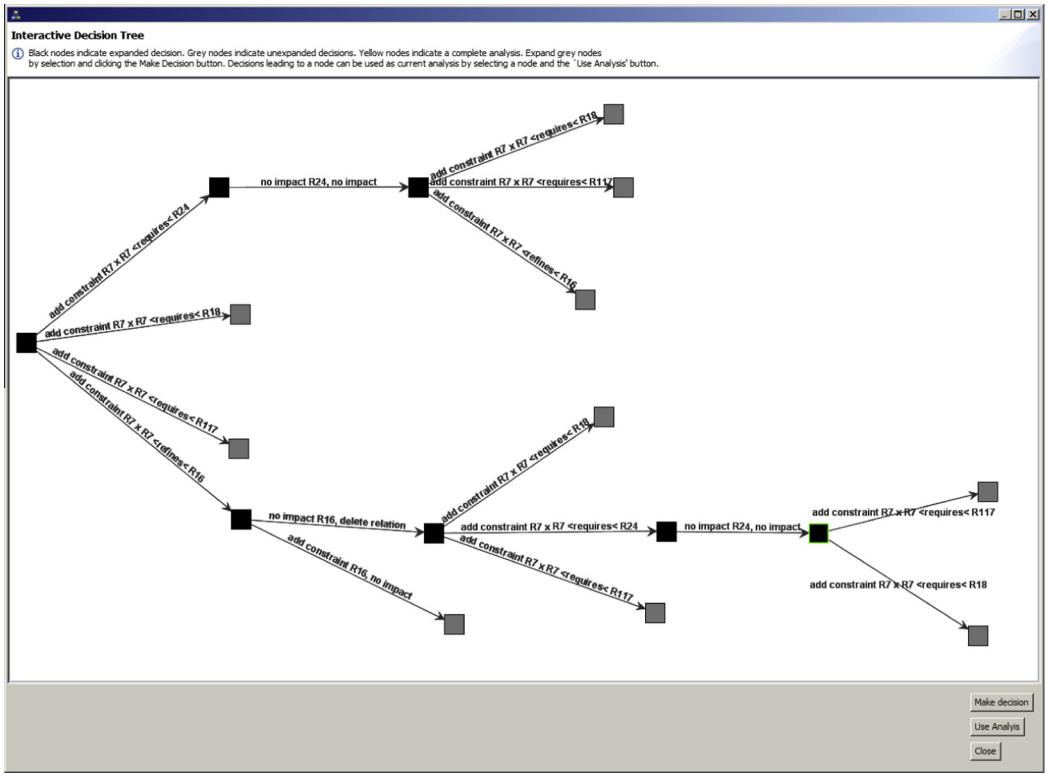


Fig. 9. Interactive decision tree builder for propagation of changes.

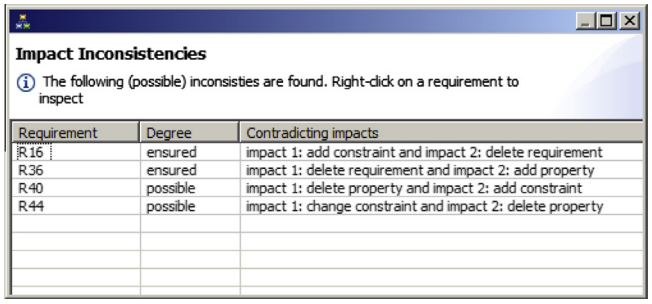


Fig. 10. GUI for displaying inconsistent proposed changes.

also provides an explanation of the contradicting proposed changes.

Implementing proposed changes in the requirements model. The tool allows the requirements engineer to implement the proposed and propagated changes in their propagation path.

Predicting the impact of proposed changes. In the impact prediction all possible propagation paths in the model are traversed by using the change impact function algorithm in order to determine change alternatives for the propagation of any selected proposed change. Fig. 11 gives the output of the impact prediction for the change "Add Constraint to Property of Requirement" in R7.

The columns list the requirements in the model, if the requirement is impacted, and the impact type respectively. The result of the impact prediction in Fig. 11 is that only R16 in the whole requirements model ***might*** be impacted with the change "Add Constraint to the Property of the Requirement" for the proposed change "Add Constraint to Property of Requirement" to R7. The tool also provides all possible propagation paths for the change alternatives listed in Fig. 11. Fig. 12 gives the propagation paths for the alternatives in R16. The first part of the window gives the alternatives for R16. There are two change alternatives to be propagated for R16: "Add Constraint to Property of Requirement" or "No Impact". The second part of the window shows the requirements in the propagation path. TRIC also supports the visualization of the propagation paths.

The impact prediction allows showing the impact of the proposed changes for the whole model. It is useful for large models to see which requirements are not impacted at all and which requirements are (or *might be*) impacted.

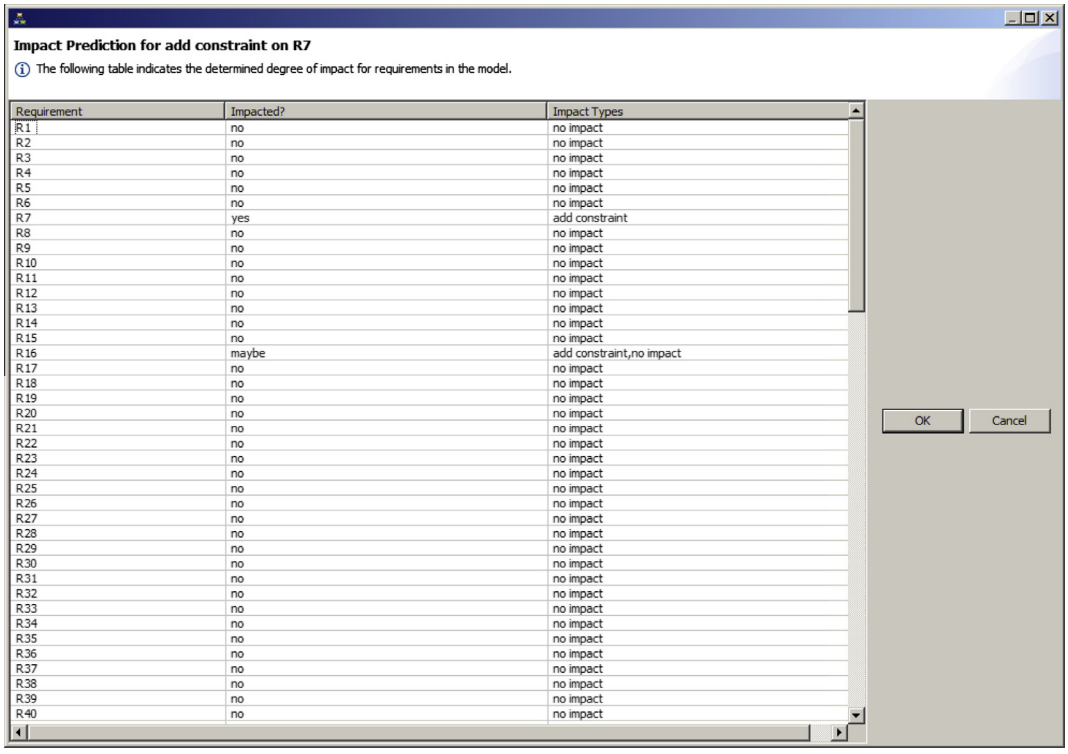
7. Example: course management system

In this section, we illustrate our approach and tool support with the CMS example. The change impact alternatives, the elimination of some false positive impacts and the consistency checking of changes are the benefits of the approach illustrated in this section. The main limitation is that the approach depends on the requirements relations. False requirements relations assigned by the requirements engineer result in wrong propagation alternatives. In our previous work [38], we provide an approach for reasoning about requirements. It supports inferencing new relations and checking consistency of relations to increase the correctness of the requirements relations in the model. In [38] we also thoroughly study how to manually identify and assign the initial relations among requirements.

7.1. Proposing and propagating requirements changes

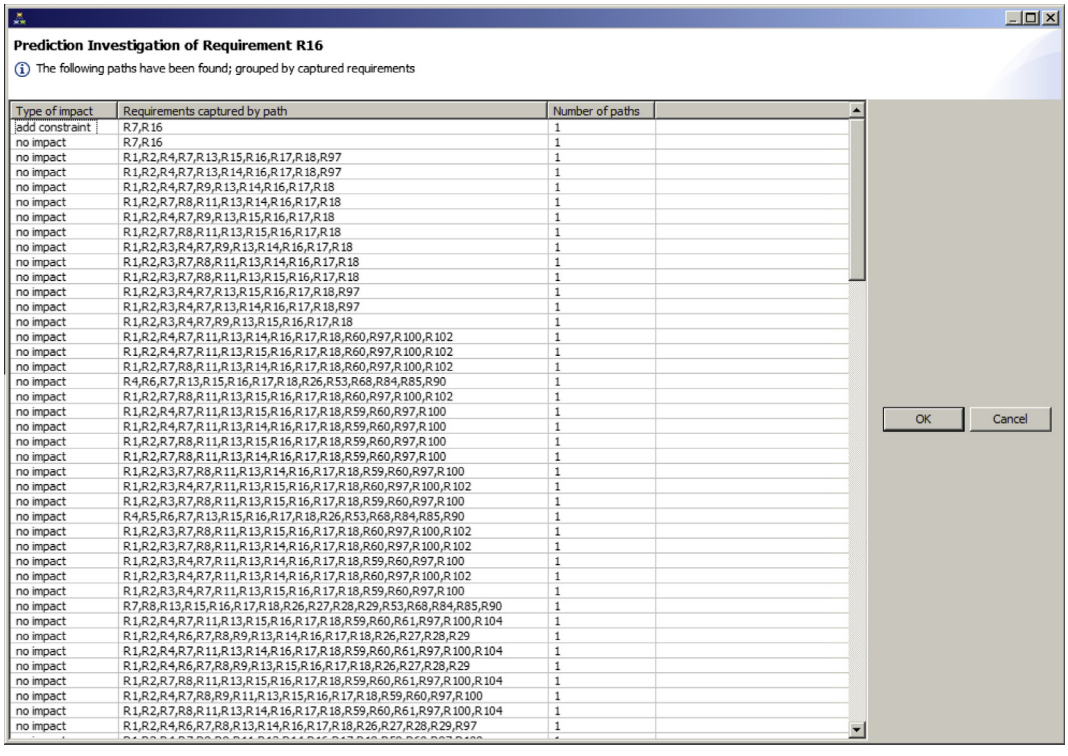
Consider the change in R7 (Add a constraint to the property messaging facility in R7) that we already used in Section 2.

R7 states a messaging facility where the sms and e-mail features are introduced as message types for the messaging facility



Requirement	Impacted?	Impact Types
R1	no	no impact
R2	no	no impact
R3	no	no impact
R4	no	no impact
R5	no	no impact
R6	no	no impact
R7	yes	add constraint
R8	no	no impact
R9	no	no impact
R10	no	no impact
R11	no	no impact
R12	no	no impact
R13	no	no impact
R14	no	no impact
R15	no	no impact
R16	maybe	add constraint,no impact
R17	no	no impact
R18	no	no impact
R19	no	no impact
R20	no	no impact
R21	no	no impact
R22	no	no impact
R23	no	no impact
R24	no	no impact
R25	no	no impact
R26	no	no impact
R27	no	no impact
R28	no	no impact
R29	no	no impact
R30	no	no impact
R31	no	no impact
R32	no	no impact
R33	no	no impact
R34	no	no impact
R35	no	no impact
R36	no	no impact
R37	no	no impact
R38	no	no impact
R39	no	no impact
R40	no	no impact

Fig. 11. Output of the impact prediction for the proposed change in R7.



Type of impact	Requirements captured by path	Number of paths
add constraint	R7,R16	1
no impact	R7,R16	1
no impact	R1,R2,R4,R7,R13,R15,R16,R17,R18,R97	1
no impact	R1,R2,R4,R7,R13,R14,R16,R17,R18,R97	1
no impact	R1,R2,R4,R7,R9,R13,R14,R16,R17,R18	1
no impact	R1,R2,R7,R8,R11,R13,R14,R16,R17,R18	1
no impact	R1,R2,R4,R7,R9,R13,R15,R16,R17,R18	1
no impact	R1,R2,R7,R8,R11,R13,R15,R16,R17,R18	1
no impact	R1,R2,R3,R4,R7,R9,R13,R14,R16,R17,R18	1
no impact	R1,R2,R3,R7,R8,R11,R13,R14,R16,R17,R18	1
no impact	R1,R2,R3,R7,R8,R11,R13,R15,R16,R17,R18	1
no impact	R1,R2,R3,R4,R7,R13,R15,R16,R17,R18,R97	1
no impact	R1,R2,R3,R4,R7,R13,R14,R16,R17,R18,R97	1
no impact	R1,R2,R3,R4,R7,R9,R13,R15,R16,R17,R18	1
no impact	R1,R2,R4,R7,R11,R13,R14,R16,R17,R18,R60,R97,R100,R102	1
no impact	R1,R2,R4,R7,R11,R13,R15,R16,R17,R18,R60,R97,R100,R102	1
no impact	R1,R2,R7,R8,R11,R13,R14,R16,R17,R18,R60,R97,R100,R102	1
no impact	R4,R6,R7,R13,R15,R16,R17,R18,R26,R53,R68,R84,R85,R90	1
no impact	R1,R2,R7,R8,R11,R13,R15,R16,R17,R18,R60,R97,R100,R102	1
no impact	R1,R2,R4,R7,R11,R13,R15,R16,R17,R18,R59,R60,R97,R100	1
no impact	R1,R2,R4,R7,R11,R13,R14,R16,R17,R18,R59,R60,R97,R100	1
no impact	R1,R2,R7,R8,R11,R13,R15,R16,R17,R18,R59,R60,R97,R100	1
no impact	R1,R2,R7,R8,R11,R13,R14,R16,R17,R18,R59,R60,R97,R100	1
no impact	R1,R2,R3,R7,R8,R11,R13,R14,R16,R17,R18,R59,R60,R97,R100	1
no impact	R1,R2,R3,R4,R7,R11,R13,R15,R16,R17,R18,R60,R97,R100,R102	1
no impact	R1,R2,R3,R7,R8,R11,R13,R15,R16,R17,R18,R59,R60,R97,R100	1
no impact	R4,R5,R6,R7,R13,R15,R16,R17,R18,R26,R53,R68,R84,R85,R90	1
no impact	R1,R2,R3,R7,R8,R11,R13,R15,R16,R17,R18,R60,R97,R100,R102	1
no impact	R1,R2,R3,R7,R8,R11,R13,R14,R16,R17,R18,R60,R97,R100,R102	1
no impact	R1,R2,R3,R4,R7,R11,R13,R14,R16,R17,R18,R59,R60,R97,R100	1
no impact	R1,R2,R3,R4,R7,R11,R13,R14,R16,R17,R18,R60,R97,R100,R102	1
no impact	R1,R2,R3,R4,R7,R11,R13,R15,R16,R17,R18,R59,R60,R97,R100	1
no impact	R7,R8,R13,R15,R16,R17,R18,R26,R27,R28,R29,R53,R68,R84,R85,R90	1
no impact	R1,R2,R4,R7,R11,R13,R15,R16,R17,R18,R59,R60,R61,R97,R100,R104	1
no impact	R1,R2,R4,R6,R7,R8,R9,R13,R14,R16,R17,R18,R26,R27,R28,R29	1
no impact	R1,R2,R4,R7,R11,R13,R14,R16,R17,R18,R59,R60,R61,R97,R100,R104	1
no impact	R1,R2,R4,R6,R7,R8,R9,R13,R15,R16,R17,R18,R26,R27,R28,R29	1
no impact	R1,R2,R7,R8,R11,R13,R15,R16,R17,R18,R59,R60,R61,R97,R100,R104	1
no impact	R1,R2,R4,R7,R8,R9,R11,R13,R15,R16,R17,R18,R59,R60,R97,R100	1
no impact	R1,R2,R7,R8,R11,R13,R14,R16,R17,R18,R59,R60,R61,R97,R100,R104	1
no impact	R1,R2,R4,R6,R7,R8,R13,R14,R16,R17,R18,R26,R27,R28,R29,R97	1

Fig. 12. All possible propagation paths for the change alternative in R16.

in the description of the change. Since these features are constrains for the property 'messaging facility', the type of the change is 'Add constraint to Property of Requirement'. Then, the proposed change is propagated to the requirements related to R7. Fig. 13 gives the requirements related to R7 with distance of 2 in TRIC (dotted arrows are inferred relations).

Please cite this article in press as: A. Goknil et al., Change impact analysis for requirements: A metamodeling approach, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.03.002

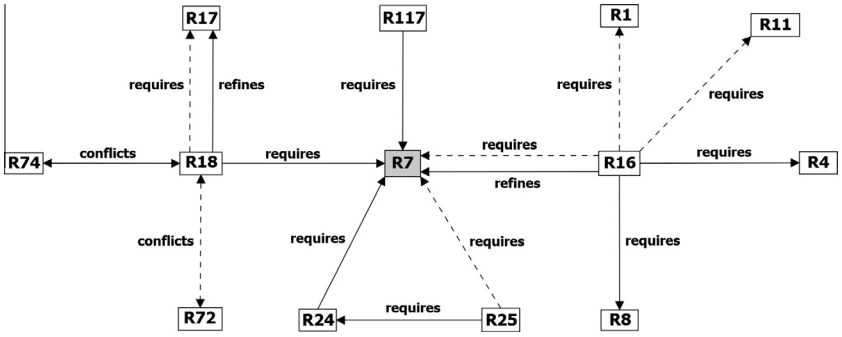


Fig. 13. Requirements related to R7 with distance of 2.

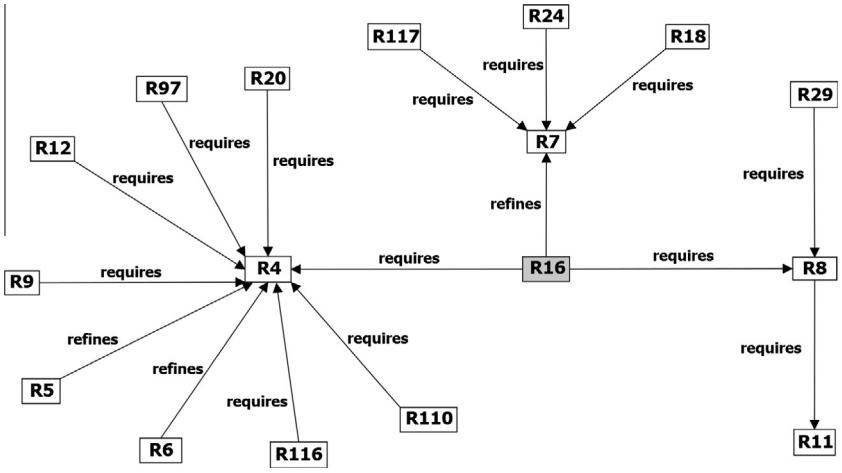


Fig. 14. Requirements related to R16 with distance of 2.

R16: The system shall allow messages to be sent to individuals, teams, or all course participants at once.
R18: Teams are created by students inviting other students in the same course using the messaging system.
R24: The system shall notify students about events (new messages posted, etc.).
R25: The system shall allow students to customize the notification behavior.
R117: The system shall allow the administration to evaluate courses through students by means of a web-survey.

According to Table 3 in Section 5, there is no impact for R18, R24, R25 and R117, which *require* R7, for the proposed change in R7. Then, we do not have to check some of the indirectly related requirements. For instance, we do not check R17, R74 and R72 since the change can be propagated to them via only R18. Please note that with the *impact prediction* feature these requirements are automatically identified as '*not impacted*' (see Fig. 11).

There are two change alternatives to propagate the change from R7 to R16 via the *refines* relation: 'Add Constraint to Property of Requirement' or 'Delete Relation'. The change type 'Add Constraint to Property of Requirement' is chosen for R16 since the constraint for R7 is also a constraint for R16.

The proposed change for R16 is the following.
Change: Add Constraint to the Property of R16
Description of Change: Messages to be sent to individuals, teams, or all course participants at once with both sms and e-mail.

The next propagation is from R16 to its related requirements. Fig. 14 gives the requirements related to R16 with distance of 2 (inferred relations are not shown for simplicity).

R4: The system shall provide *dynamic course information*.
R8: The system shall enable students to retrieve contact information of students and lecturers of subscribed courses.

According to Table 3, for the proposed change in R16, there is no impact for R4 and R8 which are related to R16 via the *requires* relation. Then, we do not have to check R5, R6, R9, R11, R12, R20, R29, R97, R110 and R116 since they are indirectly related to R7 via R4 and R8. Please also note that with the *impact prediction* feature these requirements are automatically identified as '*not impacted*' (see Fig. 11). There is no other requirement related to R16 and the change propagation is over.

Please cite this article in press as: A. Goknil et al., Change impact analysis for requirements: A metamodeling approach, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.03.002

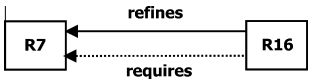


Fig. 15. Propagation path of the proposed change for R16 in the inconsistency.

7.2. Checking consistency

In this section, we discuss the inconsistencies which are detected by our tool. In the following we give R16 with some inconsistent proposed changes.

> **Change 1**: Add Constraint to the Property of R16 (by propagating the change in R7)
> **Description of Change 1**: Messages to be sent to individuals, teams, or all course participants at once with sms and e-mail.
> **Change 2**: Delete R16 (by directly proposing)
> **Description of Change 2**: Messaging individuals, teams, or all course participants is not required any more.

According to Table 4, the changes "Add Constraint to the Property of Requirement" and "Delete Requirement" cause an inconsistency for sure. Since the change "Add Constraint to the Property of Requirement" is a propagated change, we also need to analyze its change propagation path (see Fig. 15).

According to the propagation path, the proposed change in R16 is caused by propagating the change in R7 via the *refines* relation. In order to fix the inconsistency, the requirements engineer has three options. He/she might decide that the proposed change "Delete Requirement" in R16 is not valid, or the proposed change "Add Constraint to the Property of Requirement" in R7 is not valid. The third option is that the change propagation is reconsidered and the change alternative "Delete Relation" is chosen instead of the change "Add Constraint to the Property of Requirement" for R16 (see Section 7.1). This decision has to be made as a result of the negotiation between the requirements engineer and the stakeholder who proposes the change request.

7.3. Comparison of the results in our approach and RequisitePro

The goal of the comparison is to show that our approach produces less false positives and provides better guidance what and how to be changed in impact analysis. We compare our approach with one of the industrial requirements management tools IBM Rational RequisitePro [44] since it is a good representative of the tools and approaches which perform impact analysis without semantic information. By using the semantics we provide a more precise impact analysis because of the following features:

- elimination of some of the false positive impacts in change propagation,
- consistency checking of changes.

We compare our approach with RequisitePro based on these features.

Elimination of some of the false positive impacts in change propagation. Our approach provides a classification of changes in requirements models (see Table 3). Propagation alternatives are provided to be chosen by the requirements engineer. Change alternatives provide information about what to change in impacted requirements. Table 5 gives some of the change impact alternatives in TRIC and IBM RequisitePro.

In Table 5 there are three change types and their propagation alternatives provided by TRIC and RequisitePro. RequisitePro has only two relation types (*traceFrom* and *traceTo*) with informal definitions. As shown in Table 5, for each change type, RequisitePro provides two alternatives (*No impact* or *Change* R_k) since there is only one change type (*Change requirement*). Therefore, the requirements engineer has to inspect the impacted requirement to determine the type of the change without any semantic information. In our approach, the requirements engineer inspects the impacted requirement based on the change alternatives derived from the semantics of relations and change types. On the other hand, the requirements engineer has to spend some effort to model requirements and determine their relations before performing change impact analysis. Fig. 1 gives the requirements related to R7 in the CMS requirements model with distance of 2 in RequisitePro (see Fig. 13 for the correspondence model in TRIC).

Please note that RequisitePro does not provide any visualization similar to the one in Fig. 1. We converted the part of the matrix view of the CMS requirements model in RequisitePro to the graph visualization. Please again consider the following change to R7 in RequisitePro.

> **R7**: The system shall provide a messaging facility.
> **Description of Change**: Messaging facility should also contain sms and e-mail features.

Table 5
Some change impact alternatives in TRIC and IBM rational RequisitePro.

Change types	Requirements relation types						
	Relation types in our approach					Relation types in RequisitePro	
	R_i contains R_k	R_i refines R_k	R_i partially refines R_k	R_i requires R_k	R_i conflicts R_k	R_i trace from R_k	R_i trace to R_k
$R_i \overset{+pt}{\mapsto} R_i^l$	No impact	Add property to R_k | Delete relation	Delete relation	No impact	No impact	No impact | Change R_k	No impact | Change R_k
$R_i \overset{-pt}{\mapsto} R_i^l$	No impact | Delete relation | (Delete R_k & Delete relation) | Delete property of R_k	Delete property of R_k | (Delete property of R_k & Delete relation)	Delete property of R_k	No impact | Delete relation | (Delete R_k & Delete relation)	No impact | Delete relation	No impact | Change R_k	No impact | Change R_k
$R_i \overset{pt_1 \mapsto pt^l}{\mapsto} R_i^l$	No impact | Change property of R_k	Change property of R_k | (Change property of R_k & Delete relation)	Change property of R_k | (Change property of R_k & Delete relation)	No impact | Delete relation | (Delete R_k & Delete relation)	No impact | Delete relation	No impact | Change R_k	No impact | Change R_k

Please cite this article in press as: A. Goknil et al., Change impact analysis for requirements: A metamodeling approach, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.03.002

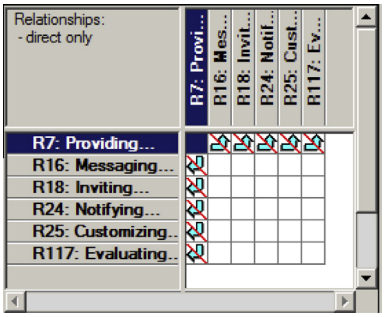


Fig. 16. Suspended relations for impacted requirements by the change in R7.

Since RequisitePro does not support proposing changes, the change is implemented by updating R7. The requirements relations for R7 get suspect. Fig. 16 shows the suspended relations in the matrix view.

R16: The system shall allow messages to be sent to individuals, teams, or all course participants at once.
R18: Teams are created by students inviting other students in the same course using the messaging system.
R24: The system shall notify students about events (new messages posted, etc.).
R25: The system shall allow students to customize the notification behavior.
R117: The system shall allow the administration to evaluate courses through students by means of a web-survey.

All directly related requirements are with the suspended relations (R16, R18, R24, R25 and R117) and they are candidate in the impact analysis. For the same change with the change type 'Add Constraint to Property of Requirement' in TRIC, 'No Impact' is automatically identified for R18, R24, R25 and R117 (see Section 7.1). TRIC identifies two alternatives for R16 via the *refines* relation: 'Add Constraint to Property of Requirement' or 'Delete Relation'. The requirements engineer inspects R16 to propose a change among these two alternatives.

Without employing any semantic information, all requirements directly related to the changed requirement are identified as candidate in the impact analysis. The requirements engineer has to check all these requirements manually to identify which requirements are actually not impacted (false positive impacts). For some change and relation types, our approach identifies 'No Impact' for the related requirements. For instance, any change in one of the conflicting requirements does not have any impact in another conflicting requirement ('R_i conflicts R_k' column in Table 3). For the change types 'Add Property to Requirement' and 'Add Constraint to Property of Requirement', 'No Impact' is automatically detected via some relations (see the rows in Table 3).

As we depicted above for the change 'Add Constraint to Property of Requirement' in R7, TRIC automatically identifies 'No Impact' for R18, R24, R25 and R117 which are actually false positive impacts in RequisitePro. Apart from directly related requirements, there might be other candidate indirectly related requirements (see Fig. 2). The requirements indirectly related to R7 at distances of 2, 3 and 4 (see Fig. 2(b–d)) are candidate to be inspected in RequisitePro.

Our approach provides automatic impact prediction for any proposed change (see Section 6). The output of the impact prediction is the impacted requirements including both directly and indirectly requirements with change alternatives. For instance, for the change in R7, the output of the impact prediction is that only R16 might be impacted with the change type 'Add Constraint to Property of Requirement' (see Fig. 11). All other impacts identified by following directly and indirectly related requirements in RequisitePro are false positives. We reduce the number of elements to be inspected. Table 6 gives the number of candidate CMS requirements for some changes performed in RequisitePro and TRIC.

The first and second columns in Table 6 give the number of directly and indirectly related requirements to be inspected by the requirements engineer in RequisitePro. The third column shows the number of candidate requirements detected by TRIC (impact prediction feature) and the fourth column gives the number of actual impacted requirements determined by our manual investigation in the whole model. All actual impacted requirements are also detected by TRIC. With some change types ($R_7 \overset{+ct}{\mapsto} R_7^I, R_8 \overset{+ct}{\mapsto} R_8^I, R_{14} \overset{+pt}{\mapsto} R_{14}^I$ and $R_{55} \overset{+pt}{\mapsto} R_{55}^I$) TRIC significantly reduces the number of false positive impacts for indirectly related requirements.

As seen in Table 6 (see the row for $R_{55} \overset{+pt}{\mapsto} R_{55}^I$) our approach still may produce some false positives. On the other hand, TRIC does not produce any false negatives; we were able to detect all actual impacted requirements with TRIC. In order to apply the approach, the requirements engineer needs some effort to model requirements and their relations. TRIC provides a reasoning framework [38] to infer new relations and check the consistency of the given relations. Also in the beginning of impact analysis he/she needs to determine the type of the proposed change based on our classification.

Consistency checking of changes. Our approach provides consistency checking of changes based on the formal semantics of requirements, relations and changes (see Section 7.2). RequisitePro does not support any consistency checking activity for requirements changes.

Table 6
Number of candidate CMS requirements for change impact in RequisitePro and TRIC.

Changes	Number of candidate/impacted requirements			
	Number of directly related candidate requirements in RequisitePro	Number of indirectly related candidate requirements with a distance of 5 in RequisitePro	Number of candidate requirements in TRIC	Number of actual impacted requirements
$R_7 \overset{+ct}{\mapsto} R_7^I$	5	72	1	1
$R_8 \overset{+ct}{\mapsto} R_8^I$	4	73	0	0
$R_{22} \overset{-ct}{\mapsto} R_{22}^I$	1	2	0	0
$R_{23} \overset{-ct}{\mapsto} R_{23}^I$	1	2	0	0
$R_{14} \overset{+pt}{\mapsto} R_{14}^I$	3	79	0	0
$R_{55} \overset{+pt}{\mapsto} R_{55}^I$	4	50	3	2
$R_{21} \overset{-pt}{\mapsto} R_{21}^I$	1	2	0	0
$R_{65} \overset{-pt}{\mapsto} R_{65}^I$	1	2	0	0

Please cite this article in press as: A. Goknil et al., Change impact analysis for requirements: A metamodeling approach, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.03.002

8. Discussion of the approach

The applicability of the approach depends on the availability of an explicit structural information in the requirements models that gives the relations and their types. Encoding such information often leads to an additional effort from the requirements engineer. Fortunately, requirements documents already provide some of this information. Our metamodel is generic enough to accommodate most of the requirements structuring strategies used in the today's practices. This section elaborates more on different aspects of the applicability of the approach

Chosen formalization, requirements metamodel and change classification. We chose a formalization of requirements, their relations and the change classification in FOL. There are other formalizations of requirements, for example, in modal logic and deontic logic [61]. The formalization in FOL allows the expression of commonly occurring requirement descriptions, including for example real-time or performance requirements. However, there are limitations of the expressivity of FOL. For instance, imperfect requirements can be modeled by fuzzy sets [70]. Dealing with imperfection is out of scope of our formalization. We do not cover modalities in requirements like possibility, probability, and necessity or logic operators like "in the next state" and "sometime in the future" which can be used to describe the evolution of requirements. Our formalization should be extended with temporal logic, modal logic or fuzzy sets in order to cover these types of requirements. Under these limitations, the expressiveness of FOL is sufficient for change impact analysis.

Since the focus of our approach is on the commonly occurring requirements relations, we investigated and benefited from several approaches which are commonly used to define and represent requirements: goal-oriented [91,67], aspect-driven [76], variability management [66], use-case [19], domain-specific [74,50], and reuse-driven techniques [60]. The main activity for constructing the metamodel is to choose the most commonly occurring requirements relations. The selected relation types are compatible with the results of the industrial case study conducted by Zhang et al. [96] to evaluate the applicability of existing dependency types in the literature. The change classification presented in this paper is a result of the chosen formalization and requirements metamodel. The idea behind the classification is based on the structure of a textual requirement as system property and constraints on this system property. The formalization of the requirements changes is aligned with the formalization of the requirements.

Scalability. TRIC provides an impact prediction feature which traverses all possible propagation paths in the model by using the change impact function algorithm. The change impact function may need more efficient graph traversal algorithms [29] to improve its scalability for large models.

The time for model exploration can be reduced by combining manual and automatic inspection. For example, some of the paths can be excluded early by manually detecting lack of change propagation. In this way the propagation algorithm works on a submodel instead on the whole model. We already explained how the elimination of some propagation paths in an early phase significantly reduces the number of the impacted elements thus avoiding the need to inspect them.

Expressing requirements and their relations. Our approach heavily depends on the requirements relations initially given by the requirements engineer. In our previous work we provide a reasoning facility in order to enhance the correctness of requirements models. However, the practicality of our approach has still some implications in the way to express requirements with their relations and the ability of the requirements engineer to analyze them. The requirements engineer may spend a substantial effort to analyze the requirements relations as an initial input. By using TRIC's reasoning features (inferencing and consistency checking of relations) the requirements engineer iterates over the requirements model. Similarly to our approach, in commercial tools like IBM RequisitePro and DOORS the requirements engineer has to give effort to identify requirements relations in order to perform change impact analysis. Existing relations need further classification according to our metamodel. Very often, some of the relations that we support are already present in requirements documents, for example, in hierarchically structured documents that follow part-whole decomposition. In order to reduce the effort, natural language processing (NLP) techniques can also be applied to automatically identify some of the relations in requirements specifications which conform to some requirement boilerplates [3].

Integration of the approach with existing requirements engineering tools. TRIC is a prototype tool that supports our approach. Alternatively, the used algorithms can be implemented as an extension of an industrial requirements management tool. The extension should be implemented with the following main features: (1) managing requirements and relation types given in our metamodel, (2) reasoning over requirements models (inferring new requirements relations and checking consistency of relations), (3) propagating changes based on the impact alternatives given in Table 3, and (4) checking consistency of changes based on the rules given in Table 4. Inferencing new relations and checking consistency of relations are not part of the change impact analysis but they are important to improve the accuracy of requirements relations which are the main input of our approach. Since our models are stored as OWL ontologies for reasoning purposes, in the extension there is a need to use an existing API such as Jena [47] or to implement one from scratch to process RDF and OWL ontologies. There is also a need to transform requirements models in the requirements management tool's own model storage format to OWL/RDF.

Some requirements management tools provide their own APIs to enable implementing extensions. IBM Rational DOORS provides its own extension language (the Rational DOORS Extension Language – DXL) that can be used to control and extend IBM DOORS functions. This scripting language allows adding custom options to DOORS menus and responding to events by triggering custom programs. Therefore, in IBM DOORS the main features of our approach can be implemented as custom programs and controlled from the added options in DOORS menus.

TRIC uses Eclipse platform to manipulate models. The metamodel is implemented as an Ecore metamodel on top of the Eclipse Modeling Framework (EMF) API. This facilitates the application of our approach to existing approaches for requirements modeling based on EMF. OMG SysML [74,80] is an example language that defines requirements modeling constructs. Requirements relations defined in SysML are either already present in our metamodel or can be defined as specializations. Situations like this provide an opportunity for an easy adaptation and integration of our approach and tooling. Our work on applying TRIC and the change impact analysis algorithm on SysML requirements models is described in [33,86].

Adapting TRIC to industrial environments. Applying research prototype tools in industrial context usually requires significant investment in tool usability and robustness. TRIC provides a matrix view and a visual editor for managing requirements relations. The usability of the tool is already improved for large models with the visual editor which enables selecting requirements to be shown. Most of the requirements management tools allows grouping/clustering requirements based on their priority or category. By having the grouping feature, the usability of the matrix view in TRIC can be increased for its use in change impact analysis.

Please cite this article in press as: A. Goknil et al., Change impact analysis for requirements: A metamodeling approach, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.03.002

TRIC provides an import/export mechanism for IBM RequisitePro in which the requirements engineer can export a RequisitePro model to a Microsoft Excel file. The exported model can be imported by TRIC and vice versa. The import/export mechanism for other requirements management tools like IBM DOORS, Borland Caliber [8] and TopTeam Analyst [88] can be implemented to increase the interoperability of TRIC with industrial tools.

Our approach is mainly based on the requirements relations and their semantics. The requirements engineer may have to define his/her own relation types. In TRIC, defining new relation types requires customization of the requirements metamodel followed by a formalization of the newly defined types. Therefore, TRIC needs to provide practical ways to support the reuse of the formalization for the metamodel customization. In [33] we show how our requirements metamodel can be customized for different requirements modeling approaches and notations such as Product-line and SysML. However, the customization of the metamodel with the formalization is still a challenge for the practitioners.

9. Related work

We classify the related work in five categories: *Requirements Relations, Requirements Metamodeling, Change Classification, Change Impact Analysis in Requirements and Change Impact Analysis in UML models*.

9.1. Requirements relations

We studied literature about requirements relation types and their semantics. Dahlstedt and Persson [21] address requirements relations (they call a relation an "interdependency") from a traceability perspective. They give an overview of requirements relations research and present a model of fundamental relation types. There is a classification (*structural*, *constrain*, and *cost/value interdependencies*) of fundamental interdependency types which includes some of the relations (*refines*, *requires*, and *conflicts*) we also use in our approach. The need to understand the nature of requirements relations and their influence on software development activities such as change management are stated. However, there is no formal semantics for the relations. Carlshamre et al. [13] run an industrial survey of requirements in software product release planning. Their aim is to learn about the nature of interdependencies in general, to be able to classify them, and to assess the relative frequency of different classes. The results show that roughly 20% of the requirements are responsible for 75% of the interdependencies and only a few requirements are singular. Although the two studies mentioned above motivate the need for requirements relations, no much attention is paid for how to give formal semantics of the relations and their use for change impact analysis.

Lee et al. [55] studies relationships between soft functional requirements based on fuzzy logic. The types of relations between soft functional requirements are classified as *conflicting*, *irrelevant*, *cooperative*, *counterbalance* and *independent*. These relation types are formalized by using fuzzy logic. Contrary to our approach, the relation types in [55] are specialized for imprecise requirements and they are used for trade-off analysis.

The survey in [78] introduces Requirements Interaction Management (RIM), which is concerned with the analysis and management of dependencies among the requirements. One of the activities in RIM, is reasoning on requirements interactions. Conflict detection methods for reasoning are introduced in five categories: *domain model*, *theorem model*, *scenario analysis*, *modeling checking* and *executing monitoring* methods. The domain model method is summarized in the survey that a domain model of system requirements interactions is used to identify interactions at the requirement level. We consider that our requirements metamodel is our domain model of requirements relations which stand for requirements interactions to identify relations between requirements.

Zhang et al. [96] conducted a case study to evaluate the usefullness and applicability of existing dependency types given in Dahlstedt's dependency model (*D*-model) [21] and Pohl's dependency model (*P*-model) [75] in change propagation with a real-world industry project. The findings in the conducted case study are (1) the definitions of some dependency types are confusing; (2) some dependency types from literature are not common and seldom found in real projects; and (3) in existing dependency models, five dependency types propagate changes, but their definitions need to be clarified [96]. It is also stated that dependency discovery and change impact analysis are subjective processes and to alleviate the subjectivity the definitions of dependency types need to be very specific, explicit and clear. Based upon the empirical evaluation of existing dependency models, Zhang et al. also propose a new dependency model including dependency types for change impact analysis (*Constrain*, *Precede*, and *Be_similar_to*). They state that the description of the new dependency types still needs improvement in their dependency model. The relation types in our metamodel covers their dependency types with a concrete interpretation of their descriptions based on formal semantics.

9.2. Requirements metamodeling

Vicente-Chicote et al. [93] describe a requirements metamodel and a modeling environment. The environment supports: graphical requirements models, their validation against the metamodel and against a set of constraints written in OCL, and automatic generation of a navigable Software Requirements Specification document (SRS). In the requirements metamodel, there are three types of trace links between requirements: *DependenceTrace*, *InfluenceTrace*, and *ParentChildTrace*. The relations are defined informally.

Monperrus et al. [65] provide a requirements metamodel that supports the specification and implementation of requirements metrics in the literature. The metamodel is centered on the notion of requirement and a requirement can be refined in several types (*CapabilityRequirement*, *ConstraintRequirement*, etc.). It does not support any type of requirements relations. For the life cycle of requirements the metamodel encodes types of requirements changes as *Modification*, *Addition* and *Deletion*. No formalism is provided for the change types. Baudry et al. [4] introduce a metamodel for requirements and present how they use it on top of a constrained natural language for requirements definitions. The requirements metamodel captures functional requirements as use cases with pre-conditions and post-conditions that constrain the activation of use cases. Operations are added in the metamodel in order to simulate requirements models. The goal of the simulation is to instantiate the use cases, replacing the formal parameters with actual values defined in an initial configuration. The metamodel does not capture the static part of requirements. It does not have the notion of requirements relations. On the other hand, our approach covers the static aspects of requirements including non-functional requirements and change impact analysis. In [11], a model-driven mechanism is proposed to merge different requirement specifications and reveal inconsistencies between them by using a requirements metamodel. The requirements metamodel is mainly used to produce a requirements model from a given requirements document. Requirements relations are not typed and lack semantics. There is no support for change impact analysis.

Some authors [42,82] use the UML profiling mechanism in a goal-oriented requirements engineering approach. Heaven et al. [42] introduce a profile that allows the KAOS model [91] to be

Please cite this article in press as: A. Goknil et al., Change impact analysis for requirements: A metamodeling approach, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.03.002

represented in UML. They also provide an integration of requirements models with lower level design models. Supakkul et al. [82] use the UML profiling mechanism to provide an integrated modeling language for functional and non-functional requirements that are mostly specified by using different notations. These two works aim at a metamodel for goal-oriented requirements engineering rather than reasoning over requirements and change impact analysis.

SysML [74,80] uses the UML profiling mechanism to provide modeling constructs that represent text-based requirements and relate them to other modeling elements. The relation types for requirements in SysML are *derive*, *copy*, and *contain*. SysML also provides a stereotype mechanism that allows the requirements engineer to specify their own relation types. Formal semantics of relation types is not considered. The definitions of the relations tend to be ambiguous. No reasoning facility for requirements is provided.

Vogel and Mantell [94] provides a UML profile that allows the modeling of stakeholders, requirements and test cases. The profile has two parts: *Stakeholders* and *Requirements*. The first part includes entities for types of stakeholders such as *User*, *Project Stakeholder*, *Supplier* and *Customer*. The second part of the profile contains entities for TestCase and types of requirements such as *Performance Requirement* and *Functional Requirement*. The profile contains entities similar to entities in our requirements metamodel. However, there is no requirements relation in [94].

COMET [20], a requirements modeling method, provides a requirements metamodel which is an extension to the use case concept of UML. COMET considers the UML use cases as the only requirements specification method. The requirements metamodel includes a use case entity with interacting roles, scenario which is the detailed description of the use case, goal entity, and the requirement entity represented by the use case. Requirements relations are not represented in the requirements metamodel of COMET.

Navarro et al. [68] propose a customization approach for requirements metamodels. They propose a core requirements metamodel which is generic and considers only *Artifact* and *Dependency* as core entities. The metamodel does not contain concrete types for requirements relations. This disallows the application of any change impact decision table for the core relations to customized entities. The Requirements Interchange Format (RIF) [77] structures requirements and their attributes, types, access permissions, and relationships. It is defined as an XML schema. Its data model has generic entities and relations like *Information Type*, *Association*, and *Generalization*. These entities can be formalized to perform change impact analysis.

Some papers address domain-specific requirements models. Koch et al. [50] propose a requirements metamodel specialized for Web systems. They identify the general structure of Web systems in order to define the requirements metamodel. The requirements metamodel for web requirements, presented by Escalona and Aragon [28], is divided into two packages: the *Behavior* and the *Structure*. In the behavior package, concepts such as *WebActor* and *WebUseCase* related to the behavior of the system presented. In the structure package, any information storage for the system is represented. Molina et al. [63,64] propose another web engineering requirements metamodel as an extension that can be integrated with existing web engineering methodologies. A tool is provided as an eclipse plug-in that accompanies the metamodel presented in [63,64]. The metamodel is extended with general security concepts in [79] in order to define a domain specific language for security requirements. In [62], Molina presents a measurable requirements metamodel which extends the requirements metamodel in [63,64]. The measurable requirements metamodel supports the elicitation of measurable requirements based on the explicit connection of goals, requirements, and measures. Moon et al. [66] propose a methodology for producing requirements that can be considered as a core asset in the product line. Ceron et al. [14] discuss requirements modeling in the context of product lines. They propose a metamodel for requirements that contains both the common and variable parts. Lopez et al. [60] propose yet another metamodel for requirements reuse as a conceptual schema to integrate semiformal requirement diagrams into a reuse strategy. The requirements metamodel is used to integrate different abstraction levels for requirements definitions. All these domain-specific approaches aim at providing a structure for representing requirements and their relations. Some of them do not contain types of requirements relations and most of them only provide informal definitions of their relations.

9.3. Change classification

Buckley et al. [12] classify change types in software systems as *structural* and *semantic* changes. Another distinction is *semantics-preserving* and *semantics-modifying* changes. Our classification for the change rationale is based on [12] and adapted to requirements.

Kitchenham et al. [49] propose an ontology to identify a number of factors that influence maintenance. The ontology has *Modification Activity* as an entity, specialized by *Enhancement* and *Correction* entities. In *Corrections*, a defect such a discrepancy between the required behavior of a product/application and the observed behavior is corrected [49]. *Enhancements* might be changes in the implementation or they might be requirements changes which are *adding new requirements* or *changing existing requirements*. According to Kitchenham, "Add a new Requirement" and "Update an Existing Requirement" can be aquated to Swanson's adaptive and perfective maintenance change types [83,84] in turn. However, the requirements change types in [49] have no formal semantics.

Aizenbud-Reshef et al. [2] present an approach for defining operational semantics for a trace in UML. The semantic property of a trace is a triplet (*event*, *condition* and *actions*). An event indicates a change. Conditions help to differentiate among events. Actions describe what should and should not be done when a specific event has occurred. There are event types (*delete events*, *update events*, and *create events*) which can be considered as change types. The main goal is to achieve automated consistency management of UML class diagrams. We use a similar approach but derive our change classification from the structure of requirements whereas the change types (event types) in [2] are for UML models.

Lee et al. [56] provide a change impact analysis approach using a goal-driven traceability-based technique. There is no explicit change classification in the approach although change types such as *modify an existing requirement* and *add a new requirement* are introduced in the example section of [56]. Instead of providing a change classification, Nurmuliani et al. [71] focus on establishing how practitioners classify change requests. The *Card Sorting*, a knowledge elicitation method, is used to identify categories of change requests in practice. For instance, requirements changes are categorized as *high effort*, *medium effort*, *low effort* and *no effort changes* based on the *magnitude of effort involved* criterion by the practitioners. Harker et al. [40] describe a classification of changing requirements where each changing requirement type could be reformulated as a change type. Lam et al. [53] propose a change maturity model that reflects an organization's capability at managing change. In this maturity model, a change classification is provided with three types of change: *screen change*, *report change* and *data change*. The change classification in [53] is specialized for Customer Complaints Systems (CCCs). Ackermann and Lindvall [1] classify change requests as *data flow change*, *program flow change* and *application domain change*. Contrary to our approach,

Please cite this article in press as: A. Goknil et al., Change impact analysis for requirements: A metamodeling approach, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.03.002

none of the change classifications given above except the work in [2] has formal semantics.

9.4. Change impact analysis in requirements

A number of approaches in the literature address change impact analysis in requirements. Jonsson and Lindvall [48] present common impact analysis strategies from a requirements engineering perspective. They categorize strategies as *automatable* (traceability/dependency analysis and slicing techniques) and *manual* (design documentation and interviews). Automatable impact analysis strategies often employ algorithmic methods for change propagation [48]. Traceability analysis is an automatable strategy that examines relations among all types of software development artifacts. We consider our approach as traceability analysis.

Event-Based Traceability (EBT) [17] supports change impact analysis by automating trace generation and maintenance. In EBT, requirements and other traceable artifacts, such as design models, are linked through publish-subscribe relationship based on the *Observer design pattern* [31]. The main purpose of EBT is to determine candidate elements and maintain traces for these elements. Contrary to our approach, in EBT all elements directly/indirectly related to the changed element are candidate. EBT does not support any change impact alternatives, identification of false positives or consistency checking of changes.

A goal-driven requirements traceability approach is proposed by Lee et al. [56] to analyze requirements change impacts through goals and use cases. Traces among goals and use cases are established and evaluated. Lee et al. provide trace types without formal semantics. Contrary to our approach, this approach does not provide change alternatives. Cleland-Huang et al. [18] introduce another goal-centric approach for managing impact of a change in non-functional requirements. Non-functional requirements and their dependencies are modeled with a Softgoal Interdependency Graph (SIG). The impact detection is limited to identifying a set of directly impacted SIG elements without any change type.

Ibrahim et al. [46] present an approach for change impact analysis of object oriented software. Change impact analysis is performed from requirements to design, test case or source code. Ibrahim et al., however, do not explain how to propagate a change from one requirement to another requirement. Turver et al. [89] describe a technique dealing with the ripple effects of a change based on a graph-theoretic model. This technique can be applied not only for source code but also for design and requirements documents. The technique, however, calculates the ripple effects by using relations without any semantic information and it does not provide any change alternative.

O'Neal [72,73] proposes a change impact analysis method to evaluate requirement changes. Complementary to our approach, O'Neal addresses the identification of the consequences of a change, such as how much change should be done. Hassine et al. [41] provide change impact analysis approach for requirements described as detailed scenarios. Dependencies between scenarios are used to identify the impacted scenarios. However, the approach does not provide any change alternative.

Cheng et al. [16] propose a method of requirements change management based on keyword mapping. Each requirement is defined as a keyword and a keyword sentence is used to arrange all the keywords according to a certain kind of order. When a change request is received for a keyword, the relations of keywords are analyzed as part of the impact analysis. However, the requirements engineer is not supported for how the change is propagated.

Chen et al. [15] introduce a holistic approach to change impact analysis in handling not only software contents but also other items such as requirements, documents and data. The approach relates heterogeneous items by using attributes and linkages. The linkages relate items classified as design document, software document, external data, and requirement. The linkages are between different types of items such as requirements-to-component but there is no mention of any linkage between two requirements.

Lock et al. [57–59] provide an approach that integrates different traceability extraction methods (pre-recorded traceability, dependency, plain experience, etc.) to determine impacted requirements. Impact propagation structure, similar to propagation path in our approach, is used with propagation probability to propagate a proposed change from one requirement to another. The only output is the candidate requirements. Lai et al. [51,52] provide a model-based approach for propagating changes between requirements and design models (particularly activity and sequence diagrams). None of the approaches given above supports consistency checking of requirements changes.

Our approach is based on our first attempt [34] for change impact analysis. The work in [34] does not support a detailed change classification. It does not consider the change rationale, in particular the difference between refactoring and domain changes. Therefore, the decision table in [34] has some subtle difference compared to the output of RequisitePro. Our previous work in [86] provides a formal semantics of SysML requirements relations for change impact analysis. Since the descriptions of SysML relations are highly ambiguous, we had to take our own very specific interpretation for the relations. Also, the SysML relations are limited and a subset of our own relations given in our metamodel.

9.5. Change impact analysis in UML models

In the literature there are numerous works on checking consistency of changes for UML diagrams/models. Egyed [25] presents an approach for automatically deciding what consistency rules to evaluate when a UML model changes. The approach is an adaptation to incremental consistency checking. Egyed shows it is possible to detect inconsistencies quickly by building a model profiler for observing which model elements a consistency checker accessed during the evaluation of consistency rules. In [25] the profiling data forms the basis for deciding *when* to re-evaluate *what* consistency rule. In [24] Egyed shows the use of an expanded version of this profiling in understanding where to fix inconsistencies, that is, identifying all the model elements that potentially fix an inconsistency. The approach relies on the model profiler to determine the location of the fix in the model. Nentwich et al. [69] introduces the Xlinkit framework which does the inconsistency location determination via white-box analysis of consistency rules. The Xlinkit framework is not able to identify dependencies among inconsistencies that can be detected in [24]. In [26] Egyed introduces another approach based on his previous works [25,24] which is for deciding how to fix inconsistencies. The approach explored all fix choices in a trial-and-error exploration. As a continuum of the inconsistency checking work by Egyed [26,25,24], Groher and Egyed [39] uses the incremental consistency checking approach for selective undoing of model changes where the designer decides which model elements to undo.

Briand et al. [9,10] propose another approach to detect which model elements to modify as a result of a change in UML design models. They identify change propagation rules for each change type to compute change actions. There is no guarantee that the set of these rules is complete. Dam and Winikoff [22] shows how the approach [23] they developed for change propagation within design models of intelligent systems is applied to UML design models. Different from the work by Egyed [26,25,24], this approach provides the designer not only just single change actions, but a series of repair actions to make the system consistent.

Our approach determines the impact on other requirements when a change occurs in a requirement. Here, the requirements

Please cite this article in press as: A. Goknil et al., Change impact analysis for requirements: A metamodeling approach, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.03.002

changes are fostered by the evolution of the business needs. Therefore, with our approach the requirements engineer mainly tries to make the requirements model consistent with the business needs reflected by the requirements in the model. In all these UML based approaches the main aim is to keep the UML diagrams consistent with each other regardless of the change rationale.

10. Conclusion

In this paper, we presented a change impact analysis approach for requirements captured in requirements models with requirements relations. We provided a classification of requirements changes with formal semantics. The formal semantics of relations and change types enables new proposed changes to be deduced and contradicting proposed changes to be determined. Most of the approaches and tools do not focus on the formal semantics of requirements relations and change types. The formal semantics in our approach provides a more precise change impact analysis with a support of change alternative identification, elimination of false positive impacts and change consistency checking. None of the industrial requirements management tools support change impact alternatives and consistency checking of changes. The main advantage of our approach is that propagation alternatives are provided to be chosen by the requirements engineer. By providing change alternatives with impact prediction we determine some of the false positive impacts that usually occur in the industrial tools. Once the requirements engineer analyzes the impact of a change in the requirements model with our approach, by using traces between requirements and architecture [35,37] the requirements engineer/the software architect can identify the impact of this change in the software architecture.

The task of identifying and classifying relations during requirements modeling is vital to our approach. Actually whatever impact analysis we might do with or without semantic information on the potentially incorrect relations is not going to give correct results. In our previous work [38,36] we thoroughly studied how to manually identify and assign the initial relations among requirements. The requirements reasoning framework given in [38] also provides a semi-automatic tool support (the reasoning features of TRIC) to infer new relations from the initial set of relations and check the consistency of the given and inferred relations. Especially, the consistency checking feature improves the correctness of the requirements relations in the model [38].

Our approach has limitations for some change types and relation types. Change alternatives in Table 3 are used only if there is any requirement related to the changed requirement. For instance, adding a new requirement (Add R_x) has no impact on other requirements in the requirements models according to Table 3. The requirements engineer has to determine relations for the added requirement and find if there is any impact on other requirements.

There might be multiple relations between two requirements. The priority is given to the intensionally defined relations for propagation of changes through multiple relations. For instance, in the formal semantics of the relations [38,36] we stated that the *refines* and *contains* relations imply the *requires* relation. Therefore, our approach uses *refines* and *contains* to determine the change alternatives.

In the implementation of change propagation and change consistency checking, change impact alternatives in Table 3 and contradicting changes in Table 4 are hard-coded. When there is a new relation and/or change type, additional manual proofs have to be implemented in the current tool support.

Our current support is for textual requirements only. There is a variety of other forms of requirements that is used in practice, e.g. Product-line, SysML requirements diagrams, use cases, user tasks and goals. Some of these forms can be mapped to our requirements metamodel and formalization. In [33] we show how our requirements metamodel can be specialized for different requirements modeling approaches and notations such as Product-line and SysML. Mainly, the requirements relations in the metamodel are specialized to support relations in different forms of requirements. The specialization allows using the same semantics and reasoning mechanism of our requirements metamodel for multiple forms of requirements.

The empirical evaluation of the approach and tool is an important issue. In an earlier stage of the research, we conducted a controlled experiment (described in a master thesis [90]). Although the results were generally positive we did not achieve statistical significance due to low number of participants and low experience of the subjects. Because our research has evolved and the tool has been improved, the empirical validation is pending as a future work.

Acknowledgments

This work has been carried out when the first author was carrying his PhD work at the University of Twente. The work has been supported by NWO (www.nwo.nl) in the Jacquard Programme and by the National Research Fund, Luxembourg (FNR/P10/03).

Appendix A. Part of the CMS requirements document

In this appendix, we give an overview of the requirements of the Course Management System (CMS) as used in this paper. The full requirements document is available at http://www-sop.inria.fr/members/Arda.Goknil/cms/.

Requirements (partial)

Stakeholder general

R4: The system shall provide *dynamic course information*.
R5: The system shall be able to store *dynamic course information*.
R6: The system shall be able to represent *dynamic course information*.
R7: The system shall provide a messaging facility.

Stakeholder students

R8: The system shall enable students to retrieve contact information of students and lecturers of subscribed courses.
R11: The system shall enable students to subscribe to and unsubscribe from courses.
R16: The system shall allow messages to be sent to individuals, teams, or all course participants at once.
R24: The system shall notify students about events (new messages posted, team invites, scheduled exams, etc.).
R26: The system shall allow students to view course grade statistics per semester.
R29: The system shall provide a user-customizable visibility policy for the personal information.

Stakeholder lecturers

R48: The system shall allow lecturers to create courses.
R49: The system shall allow lecturers to create entirely new courses.
R59: The system shall allow lecturers to manage *static course information*.
R60: The system shall allow lecturers to limit the number of students subscribing to a course.
R61: The system shall allow lecturers to specify enrolment

(*continued on next page*)

Part of the CMS requirements document (*continued*)

Requirements (partial)

policies based on grade, first-come first-serve (fcfs), and department.

R62: The system shall allow lecturers to specify enrolment policies based on grade.

R74: The system shall allow only lecturers to create new teams.

Stakeholder administration

R97: The system shall allow only the administration to *manage* courses.

R98: The system shall allow only the administration to create new courses.

R100: The system shall allow only the administration to update *static course information*.

R102: The system shall allow only the administration to specify the minimum number of students for a course. If there are too few subscriptions in a semester, that course will not be given during that semester.

R103: The system shall have no maximum limit on the number of course participants ever.

Glossary (partial)

Static Course Information: Information about a course which does not change while a course is given but does change between semesters. This includes the lecturer, number of ECTS credits, and study material.

Dynamic Course Information: Information about a course which changes while a course is given. This includes news messages, archived files, and roster.

Manage Courses: Managing courses involves the creation, reading, updating, and deleting of courses.

References

[1] C Ackermann, M. Lindvall, Understanding change requests to predict software impact, in: 30th Annual IEEE/NASA Software Engineering Workshop, 2006, pp. 66–75.
[2] N. Aizenbud-Reshef, R.F. Paige, J. Rubin, Y. Shaham-Gafni, D.S. Kolovos, operational semantics for traceability, ECMDA-TW 2005, pp. 7–14.
[3] C. Arora, M. Sabetzadeh, L. Briand, F. Zimmer, R. Gnaga, Automatic checking of conformance to requirement boilerplates via text chunking: an industrial case study, in: 7th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2013), 2013, pp. 35–44.
[4] B. Baudry, C. Nebut, Y. Le Traon, Model-driven engineering for requirements analysis, EDOC 2007, pp. 459–466.
[5] S.A. Bohner, Extending software change impact analysis into COTS components, in: 27th Annual NASA Goddard Software Engineering Workshop, 2002, pp. 175–182.
[6] S.A. Bohner, Software change impacts – an evolving, perspective, ICSM'02, 2002, pp. 263–271.
[7] S.A. Bohner, D. Gracanin, Software impact analysis in a virtual environment. in: 28th Annual NASA Goddard Software Engineering Workshop, 2003, pp. 143–151.
[8] Borland Caliber Analyst. <http://www.borland.com/us/products/caliber/index.html>.
[9] L.C. Briand, Y. Labiche, L. O'Sullivan, Impact analysis and change management of UML models, in: International Conference on Software Maintenance, 2003, pp. 256–265.
[10] L.C. Briand, Y. Labiche, L. O'Sullivan, M. Sowka, Automated impact analysis of UML models, J. Syst. Softw. 79 (3) (2006) 339–352.
[11] E. Brottier, B. Baudry, Y. Le Traon, D. Touzet, B. Nicolas, Producing a global requirement model from multiple requirement specifications, EDOC 2007, 2007, pp. 390–404.
[12] J. Buckley, T. Mens, M. Zenger, A. Rashid, G. Kniesel, Towards a taxonomy of software change, J. Softw. Mainten. Evol.: Res. Pract. 17 (5) (2005) 309–332.
[13] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, J. Natt och Dag, An industrial survey of requirements interdependencies in software product release planning, in: Proceedings of the 5th International Symposium on Requirements Engineering, 2001, pp. 84–91.
[14] R. Ceron, J.C. Duenas, E. Serrano, R. Capilla, A meta-model for requirements engineering in system family context for software process improvement using CMMI, PROFES 2005, 3547, 2005, pp. 173–178.
[15] C.Y. Chen, P.C. Chen, A holistic approach to managing software change impact, J. Syst. Softw. 82 (12) (2009) 2051–2067.
[16] H. Cheng, Y. Xia, X. Hu, Requirements change management of information system based on the keyword mapping, in: The Sixth Wuhan International Conference on E-Business, 2007, pp. 135–140.
[17] J. Cleland-Huang, C.K. Chang, M. Christensen, Event-based traceability for managing evolutionary change, IEEE Trans. Softw. Eng. 29 (9) (2003) 796–810.
[18] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhanskaya, S. Christina, Goal-centric traceability for managing non-functional requirements, in: Proceedings of the 27th International Conference on Software Engineering (ICSE'05), 2005, pp. 362–371.
[19] A. Cockburn, Writing Effective Use Cases, Addison-Wesley, 2000.
[20] COMET (Component and Model Based Development Methodology). <http://modelbased.net/methods/comet/>.
[21] A.G. Dahlstedt, A. Persson, Requirements Interdependencies: state of the art and future challenges, in: A. Aurum, C. Wohlin (Eds.), Engineering and Managing Software Requirements, Springer, Berlin, 2005, pp. 95–116.
[22] H.K. Dam, M. Winikoff, Supporting change propagation in UML models, in: IEEE International Conference on Software Maintenance (ICSM), 2010, pp. 1–10.
[23] H.K. Dam, M. Winikoff, L. Padgham, An agent-oriented approach to change propagation in software evolution, in: Proceedings of the Australian Software Engineering Conference (ASWEC), 2006, pp. 309–318.
[24] A. Egyed, Fixing inconsistencies in UML design models, in: 29th International Conference on Software Engineering (ICSE'07), 2007, pp. 292–301.
[25] A. Egyed, Instant consistency checking for the UML, in: 28th International Conference on Software Engineering (ICSE'06), 2006, pp. 381–390.
[26] A. Egyed, E. Letier, A. Finkelstein, Generating and evaluating choices for fixing inconsistencies in UML design models, in: 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 2008, pp. 99–108.
[27] E. Erlikh, Leveraging legacy system dollars for E-business, IT Profess. 2 (3) (2000) 17–23.
[28] M.J. Escalona, G. Aragon, NDT. A model-driven approach for web requirements, IEEE Trans. Soft. Eng. 34 (3) (2008) 377–390.
[29] R. Fleischer, G. Trippen, Experimental studies of graph traversal algorithms, WEA 2003, LNCS(2647), 2003, pp. 120–133.
[30] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999.
[31] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1995.
[32] A. Goknil, Traceability of Requirements and Software Architecture for Change Management, PhD Thesis, University of Twente, Enschede, 2011.
[33] A. Goknil, I. Kurtev, J. V. Millo, A metamodeling approach for reasoning on multiple requirements models, EDOC'13, 2013, pp. 159–166.
[34] A. Goknil, I. Kurtev, K. van den Berg, change impact analysis based on formalizations of trace relations for requirements, ECMDA-TW'08, SINTEF Report, 2008, pp. 59–75.
[35] A. Goknil, I. Kurtev, K. van den Berg, Generation and validation of traces between requirements and architecture based on formal trace semantics, J. Syst. Softw. 88 (2014) 112–137.
[36] A. Goknil, I. Kurtev, K. van den Berg, A metamodeling approach for reasoning about requirements, in: European Conference on Model Driven Architecture Foundations and Applications (ECMDA-FA'08), LNCS(5095), 2008, pp. 311–326.
[37] A. Goknil, I. Kurtev, K. van den Berg, Tool support for generation and validation of traces between requirements and architecture, ECMFA-TW 2010, 2010, pp. 39–46.
[38] A. Goknil, I. Kurtev, K. van den Berg, J.W. Veldhuis, Semantics of trace relations in requirements models for consistency checking and inferencing, Softw. Syst. Model. 10 (1) (2011) 31–54.
[39] I. Groher, A. Egyed, Selective and consistent undoing of model changes, MODELS 2010, LNCS (6395), 2010, pp. 123–137.
[40] S.D.P. Harker, K.D. Eason, J.E. Dobson, The change and evolution of requirements as a challenge to the practice of software engineering, in: Proceedings of IEEE International Symposium on Requirements Engineering 1993, 1993, pp. 266–272.
[41] J. Hassine, J. Rilling, J. Hewitt, Change impact analysis for requirement evolution using use case maps, in: Eighth International Workshop on Principles of Software Evolution, 2005, pp. 81–90.
[42] W. Heaven, A. Finkelstein, UML profile to support requirements engineering with KAOS, IEE Proc. Softw. 151 (1) (2004) 10–27.
[43] M.R.A. Huth, M.D. Ryan, Logic in Computer Science: Modeling and Reasoning about Systems, Cambridge University Press, Cambridge, 2000.
[44] IBM Rational RequisitePro. <http://www-01.ibm.com/software/awdtools/reqpro/>.
[45] IBM Telelogic Doors. <http://www.telelogic.com/Products/doors/doors/index.cfm>.
[46] S. Ibrahim, M. Munro, A. Deraman, A requirements traceability to support change impact analysis, Asian J. Inf. Technol. 4 (4) (2005) 329–338.
[47] Jena. A Semantic Web Framework for JAVA. <http://jena.sourceforge.net/>.

Please cite this article in press as: A. Goknil et al., Change impact analysis for requirements: A metamodeling approach, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.03.002

[48] P. Jonsson, M. Lindvall, Impact analysis, in: A. Aurum, C. Wohlin (Eds.), Engineering and Managing Software Requirements, Springer, Berlin, 2005, pp. 117–142.
[49] B.A. Kitchenham, G.H. Travassos, A. von Mayrhauser, F. Niessink, N.F. Schneidewind, J. Singer, et al., Towards an ontology of software maintenance, J. Softw. Mainten.: Res. Pract. 11 (6) (1999) 365–389.
[50] N. Koch, A. Kraus, Towards a common metamodel for the development of web applications, ICWE 2003, 2003, pp. 497–506.
[51] W. Lai, Relationship-Based Change Propagation: A Case Study. M.Sc. Thesis, University of Toronto, Toronto, 2009.
[52] W. Lai, S. Nejati, J. Cabot, Z. Diskin, S. Easterbrook, M. Sabetzadeh et al., Relationship-based change propagation: a case study, in: Proceedings of ICSE'09 Workshop on Modeling in, Software Engineering (MiSE'09), 2009.
[53] W. Lam, V. Shankararaman, Managing change in software development using a process improvement approach, in: Proceedings of 24th Euromicro Conference 1998, 1998, pp. 779–786.
[54] A.v. Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications, John Wiley & Sons, 2009.
[55] J. Lee, J.Y. Kuo, New approach to requirements trade-off analysis for complex systems, IEEE Trans. Knowl. Data Eng. 10 (4) (1998) 551–562.
[56] W.T. Lee, W.Y. Deng, J. Lee, S.J. Lee, Change impact analysis with a goal-driven traceability-based approach, Int. J. Intell. Syst. 25 (8) (2010) 878–908.
[57] S. Lock, A Hybrid Approach to Requirement Level Impact Analysis, PhD Thesis, Lancaster University, 2001.
[58] S. Lock, G. Kotonya, An integrated framework for requirement change impact analysis, in: Proceedings of the 4th Australian Conference on Requirements Engineering, 1999, pp. 29–42.
[59] S. Lock, G. Kotonya, An integrated, probabilistic framework for requirement change impact analysis, Aust. J. Inf. Syst. 6 (2) (1999) 38–63.
[60] O. Lopez, M.A. Laguna, F.J. Garcia, Metamodeling for requirements reuse, in: Anaisdo WER02—Workshop em Engenharia de Requisitos, 2002, pp. 76–90.
[61] J.J.C. Meyer, R. Wieringa, F. Dignum, The Role of Deontic Logic in the Specification of Information Systems. Logics for Databases and Information Systems, 1998, pp. 71–115.
[62] F. Molina, J. Pardillo, C. Cachero, A. Toval, An MDE modeling framework for measurable goal-oriented requirements, Int. J. Intell. Syst. 25 (8) (2010) 757–783.
[63] F. Molina, J. Pardillo, A. Toval, Modelling web-based systems requirements using WRM, in: Web Information Systems Engineering – WISE 2008 Workshops, LNCS(5176), 2008, pp. 122–131.
[64] F. Molina, A. Toval, Integrating usability requirements that can be evaluated in design time into model driven engineering of web information systems, Adv. Eng. Softw. 40 (12) (2009) 1306–1317.
[65] M. Monperrus, B. Baudry, J. Champeau, B. Hoeltzener, J.M. Jezequel, Automated measurement of models of requirements, Softw. Qual. J. 21 (1) (2013) 3–22.
[66] M. Moon, K. Yeom, H.S. Chae, An approach to developing domain requirements reuse as a core asset based on commonality and variability analysis in a product line, IEEE Trans. Softw. Eng. 31 (7) (2005) 551–569.
[67] J. Mylopoulos, L. Chung, E. Yu, From object-oriented to goal oriented requirements analysis, ACM Commun. 42 (1) (1999) 31–37.
[68] E. Navarro, J.A. Mocholi, P. Letelier, I. Ramos, A metamodeling approach for requirements specification, J. Comput. Inf. Syst. 46 (5) (2006) 67–77.
[69] C. Nentwich, W. Emmerich, A. Finkelstein, Consistency management with repair actions, ICSE'03, 2003, pp. 455–464.
[70] J. Noppen, P. van den Broek, M. Aksit, Imperfect requirements in software development, REFSQ 2007, 4542, 2007, pp. 247–261.
[71] N. Nurmuliani, D. Zowghi, S.P. Williams, Using card sorting technique to classify requirements change, in: Proceedings of 12th IEEE International Requirements, Engineering Conference 2004, 2004, pp. 240–248.
[72] J.S. O'Neal, Analyzing the Impact of Changing Software Requirements: A Traceability-based Methodology, Ph.D. Dissertation, Louisiana State University, 2003.
[73] J.S. O'Neal, D.L. Carver, Analyzing the impact of changing requirements, in: International Conference on Software Maintenance, 2001, pp. 190–195.
[74] OMG. SysML Specification. <http://www.sysml.org/specs.htm> (retrieved 05.01.10).
[75] K. Pohl, Process-Centered Requirements Engineering, John Wiley & Sons, 1996.
[76] A. Rashid, A. Moreira, J. Araujo, Modularization and composition of aspectual requirements, AOSD 2003, 2003, pp. 11–20.
[77] RIF. Requirements Interchange Format. <http://www.automative-his.de/rif/doku.php>.
[78] W.N. Robinson, S.D. Pawlowski, V. Volkov, Requirements interaction management, ACM Comput. Surv. 35 (2) (2003) 132–190.
[79] O. Sanchez, F. Molina, J. Garcia-Molina, A. Toval, ModelSec: a generative architecture for model-driven security, J. Univ. Comput. Sci. 15 (15) (2009) 2957–2980.
[80] M.S. Soares, J. Vrancken, Model-driven user requirements specification using SysML, J. Softw. 3 (6) (2008) 57–68.
[81] W. Spijkerman, Tool Support for Change Impact Analysis in Requirement Models, MSc Thesis, University of Twente, Enschede, 2010.
[82] S. Supakkul, L. Chung, A UML profile for goal-oriented and use case driven representation of NFRs and FRs, SERA 2005, 2005, pp. 112–119.
[83] E.B. Swanson, The dimensions of maintenance, in: Proceedings of the 2nd International Conference on Software Engineering, 1976, pp. 492–497.
[84] E.B. Swanson, N. Chapin, Interview with E. Burton Swanson, J. Softw. Mainten.: Res. Pract. 7 (5) (1995) 303–315.
[85] SWEBOOK, Guide to Software Engineering Body of Knowledge, IEEE Computer Society.
[86] D. ten Hove, A. Goknil, I. Kurtev, K. van den Berg, K. de Goede, Change impact analysis for SysML requirements models based on semantics of trace relations, ECMDA-TW 2009, 2009, pp. 17–28.
[87] Tool for Requirements Inferencing and Consistency Checking (TRIC). <http://trese.cs.utwente.nl/tric/>.
[88] TopTeam Analyst. <http://www.technosolutions.com/topteam_requirements_management.html>.
[89] R.J. Turver, M. Munro, An early impact analysis technique for software maintenance, J. Softw. Mainten. Res. Pract. 6 (1) (1994) 35–52.
[90] R.S.A. van Domburg, Empirical Validation of Representation and Interpretation of Software Requirements in Requirements Models Master Thesis, University of Twente, Enschede, 2009.
[91] A. van Lamswerdee, Goal-oriented requirements engineering: a roundtrip from research to practice, in: Invited Minitutorial, Proceedings RE'01—5th International Symposium Requirements Engineering, 2001, pp. 249–263.
[92] J.W. Veldhuis, Tool Support for a Metamodeling Approach for Reasoning about Requirements, MSc Thesis, University of Twente, Enschede, 2009.
[93] C. Vicente-Chicote, B. Moros, A. Toval, REMM-Studio: an integrated model-driven environment for requirements specification, validation and formatting, J. Object Technol. 6 (9) (2007) 437–454.
[94] R. Vogel, K. Mantell, MDA Adoption for a SME: Evolution, not Revolution – Phase II. 2nd Workshop on From Code Centric to Model Centric Software Engineering: Practices, Implications and ROI, 2006.
[95] P. Zave, M. Jackson, Four dark corners of requirements engineering, ACM Trans. Softw. Eng. Methodol. (TOSEM) 6 (1) (1997) 1–30.
[96] H. Zhang, J. Li, L. Zhu, R. Jeffery, Y. Liu, Q. Wang, et al., Investigating dependencies in software requirements for change propagation analysis, Inf. Softw. Technol. 56 (1) (2014) 40–53.

Please cite this article in press as: A. Goknil et al., Change impact analysis for requirements: A metamodeling approach, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.03.002