Change Impact Analysis based on Formalization of Trace Relations for Requirements

Arda Goknil, Ivan Kurtev, Klaas van den Berg

Software Engineering Group, University of Twente, 7500 AE Enschede, the Netherlands
{a.goknil, kurtev, k.g.van.den.berg}@ewi.utwente.nl

Abstract. Evolving customer needs is one of the driving factors in software development. There is a need to analyze the impact of requirement changes in order to determine possible conflicts and design alternatives influenced by these changes. The analysis of the impact of requirement changes on related requirements can be based on requirements traceability. In this paper, we propose a requirements metamodel with well defined types of requirements relations. This metamodel represents the common concepts extracted from some prevalent requirements engineering approaches. The requirements relations in the metamodel are used to trace related requirements for change impact analysis. We formalize the relations. Based on this formalization, we define change impact rules for requirements. As a case study, we apply these rules to changes in the requirements specification for Course Management System.

Keywords: change impact analysis, requirements traceability, requirements metamodels

1 Introduction

Change management is a prerequisite for high-quality software development. Changes may be caused by changing user requirements and business goals or be induced by changes in implementation technologies. There is a need to analyze the impact of requirement changes in order to determine possible conflicts and design alternatives influenced by these changes.

The analysis of the impact of requirement changes on other requirements can be based on requirements traceability. Requirements relations can be used as trace links to determine the impact of requirements change. Current trace metamodels and mechanisms consider relations between model elements mostly without assigning any semantics. The lack of semantics in trace links causes imprecise results in change impact analysis and explosion of impacts problem [3].

We propose a requirements metamodel with well defined types of requirements relations. This metamodel represents the common concepts extracted from some prevalent requirements engineering approaches. The requirements relations are used to trace related requirements for change impact analysis. We formalize these requirements relations. Based on this formalization, we define change impact rules for

requirements. We aim at more precise analysis with these rules. As a case study, the rules are applied to changes in the requirements specification for Course Management System.

The paper is structured as follows. Section 2 gives details of the requirements metamodel. Section 3 gives the formalization for the requirements relations and consistency constraints. In Section 4, we describe the change impact rules derived from the formalization of requirements relations. In Section 5, we give a case study to illustrate the change impact analysis. Section 6 presents the related work. Section 7 concludes the paper and describes future work.

2 Requirements Metamodel

The requirements metamodel contains common concepts identified in existing requirements modeling approaches [26] [12] [19] [14] [27]. The metamodel in Fig. 1 includes entities such as *Requirement*, *Stakeholder* and *Relationship* in order to model general characteristics of requirements artifacts.

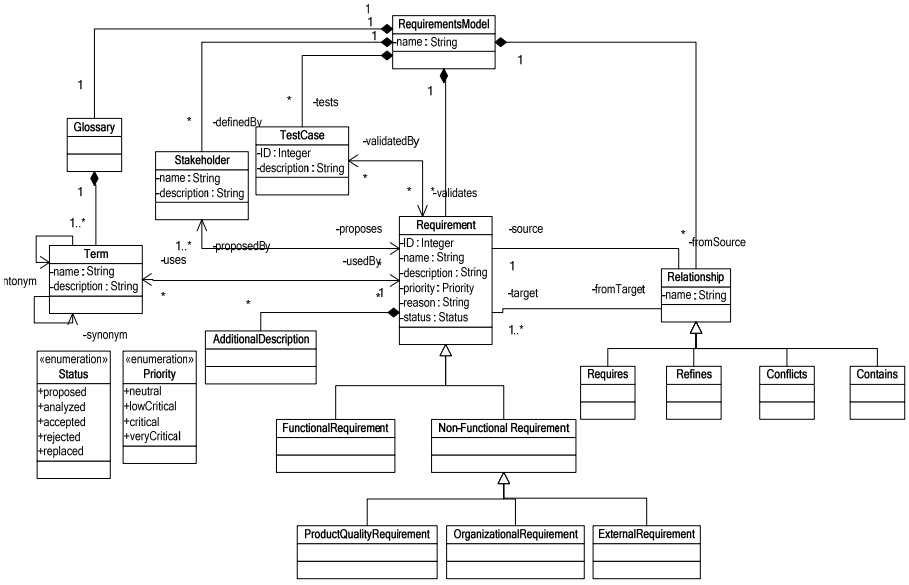

Fig. 1. Requirements Metamodel

In this metamodel, all requirements are captured in a requirements model (*RequirementModel*). A requirements model is characterized by a name property and contains requirements instances of the *Requirement* entity. All requirements have a unique identifier (*ID* property), a name, a textual description (*description* property), a priority, a rationale (*reason* property), and a status. Requirements may have additional descriptions (*AdditionalDescription* entity) such as a use case or any other formalization. Usually, requirements are classified as functional and non-functional

requirements. Non-functional requirements may come from required characteristics of the software (product quality requirements), the organization developing the software (organizational requirements) or from external sources [22]. Requirements can be related with each other. We recognize four types of relations: *Refines*, *Requires*, *Conflicts*, and *Contains*. These core relations can be specialized and new relations may be added as specializations of the *Relationship* concept. The metamodel includes the entities *Stakeholder*, *TestCase*, *Glossary* and *Term*. Test cases are not always considered as parts of requirements specifications. However, they are important to validate or verify requirements. Some metamodels [19] [27] consider test cases as a part of the requirements specification.

3 Formalization of Requirements Relations

In this section, we give the definitions and formalizations of requirements relations in Fig. 1. These formalizations make it possible to understand various types of dependency between requirements provided by the requirements relations. This understanding helps us to specify more precise change impact rules for requirements. The relations in the requirements metamodel are defined and formalized as follows:

- *Definition 1. Requires relation*: A requirement R_1 *requires* a requirement R_2 if R_1 is fulfilled only when R_2 is fulfilled. R_2 can be treated as a pre-condition for R_1 [27].
- *Definition 2. Refines relation*: A requirement R_1 *refines* a requirement R_2 if R_1 is derived from R_2 by adding more details to it [26].
- *Definition 3. Contains relation*: A requirement R_1 *contains* requirements $R_2..R_n$ if R_1 is the conjunction of the contained requirements $R_2..R_n$. This relation enables a complex requirement to be decomposed into child requirements [19].
- *Definition 4. 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 [25].

The definitions given above are intuitive and informal. In the remaining part of this section we give a formal definition of requirements and relations among them in order to derive sound change impact rules.

We assume the general notion of requirement being "a property which must be exhibited by a system" [8]. We define a requirement R as a tuple <P, S> where P is a predicate (the property) and S is a set of systems that satisfy P, i.e. $\forall s \in S : P(s)$.

- *Formalization of Requires*

Let R_1 and R_2 are requirements such that $R_1 = <P_1, S_1>$ and $R_2 = <P_2, S_2>$. R_1 *requires* R_2 iff for every $s_1 \in S_1$ then $s_1 \in S_2$.

From this definition we conclude that $S_1 \subset S_2$. The subset relation between the systems S_1 and S_2 gives us the properties of *non-reflexive*, *non-symmetric*, and *transitive* for the *requires* relation.

- *Formalization of Refines*

Let R_1 and R_2 are requirements such that $R_1 = <P_1, S_1>$ and $R_2 = <P_2, S_2>$. We assume that P_1 and P_2 are formulas in first order logic (there may be formalizations of requirements in other types of logics such as modal and deontic logic [18]) and P_2 can be represented in a conjunctive normal form in the following way:

$$P_2 = p_1 \wedge p_2 \wedge ... \wedge p_{n-1} \wedge p_n \wedge q_1 \wedge q_2 \wedge ... \wedge q_{m-1} \wedge q_m$$

Let $q^1_1, q^1_2, \ldots, q^1_{m-1}, q^1_m$ are the predicates such that $q^1_i \rightarrow q_i$ for $i \in 1..m$
R_1 *refines* R_2 iff P_1 is derived from P_2 by replacing every q_i in P_2 with $q^1_i \quad i \in 1..m$ such that the following two statements hold:

(a) $P_1 = p_1 \wedge p_2 \wedge ... \wedge p_{n-1} \wedge p_n \wedge q^1_1 \wedge q^1_2 \wedge ... \wedge q^1_{m-1} \wedge q^1_m$
(b) $\exists s \in S_2 : s \notin S_1$

From the definition we conclude that if P_1 holds for a given system s then P_2 also holds for s. Therefore $S_1 \subset S_2$. Similarly to the previous relation we have the properties *non-reflexive*, *non-symmetric*, *transitive* for the *refines* relation. Obviously, if R_1 *refines* R_2 then R_1 *requires* R_2.

- *Formalization of Contains*

Let R_1, R_2 and R_3 are requirements such that $R_1 = <P_1, S_1>$, $R_2 = <P_2, S_2>$, and $R_3 = <P_3, S_3>$. We assume that P_2 and P_3 are formulas in first order logic and can be represented in a conjunctive normal form in the following way:

$P_2 = p_1 \wedge p_2 \wedge ... \wedge p_{m-1} \wedge p_m$
$P_3 = p_{m+1} \wedge p_{m+2} \wedge ... \wedge p_{n-1} \wedge p_n$

R_1 contains R_2 and R_3 iff P_1 is derived from P_2 and P_3 as follows:

$P_1 = P_2 \wedge P_3 \wedge P'$ where P' denotes properties that are not captured in P_2 and P_3 (i.e. we do not assume completeness of the decomposition [26])

From the definition we conclude that if P_1 holds then P_2 and P_3 also hold. Therefore, $S_1 \subset S_2$ and $S_1 \subset S_3$. Obviously, the *contains* relation is *non-reflexive*, *non-symmetric*, and *transitive.*

- *Formalization of Conflicts*

Let R_1 and R_2 are requirements such that $R_1 = <P_1, S_1>$ and $R_2 = <P_2, S_2>$. Then, R1 *conflicts with* R2 iff $\neg\exists s : s \in S_1 \wedge s \in S_2 : P_1(s) \wedge P_2(s)$. The *conflicts* relation is *symmetric*.

It should be noted that the definition of *requires* is given in extensional terms as a subset relation between the systems that satisfy the requirements. The definitions of *refines* and *contains* are given in intensional terms, that is, they take into account the form of the requirement specification as a predicate. If we would interpret *refines* in an extensional way then we will conclude that *requires* and *refines* are both interpreted as a subset relation and therefore are equivalent. Apparently in our formalization, *refines* and *requires* are different.

Several constraints for the consistency of relations can be specified based on the formalizations of the relations for the requirements metamodel. These inconsistencies are different from the *conflicts* relation between requirements. Inconsistencies, here,

indicate that relations between requirements are violating their constraints. Some of the constraints for the consistency of relations and their proofs (proof by contradiction) are given below:

Constraint 1: (R_1 Refines R_2) → ¬ (R_2 Requires R_1)
Proof: Let R_1 *refines* R_2 and suppose R_2 *requires* R_1. According to the formalization of the refines relation (R_1 Refines R_2) → (R_1 Requires R_2). The *requires* relation is non-symmetric. Contradiction.

Constraint 2: (R_1 Requires R_2) → ¬ (R_1 Conflicts R_2)
Proof: Let R_1 *requires* R_2, then, by the formalization of the *requires* relation $S_1 \subset S_2$. Suppose R_1 *conflicts* R_2, then, by the formalization of the *conflicts* relation $\neg\exists s : s \in S_1 \wedge s \in S_2 : P_1(s) \wedge P_2(s)$. Contradiction.

Constraint 3: (R_1 Requires R_2) → ¬ (R_2 Contains R_1)
Proof: Let R_1 *requires* R_2, then, by the formalization of the *requires* relation $S_1 \subset S_2$. Suppose R_2 *contains* R_1, then, by the formalization of the *contains* relation $S_2 \subset S_1$. Contradiction.

Constraint 4: (R_1 Refines R_2) → ¬ (R_1 Contains R_2)
Proof: Let R_1 *refines* R_2. According to the formalization of the *refines* relation, q^1_1, q^1_2, …, q^1_{m-1}, q^1_m are the predicates such that $q^1_i \rightarrow q_i$ for $i \in 1..m$. Suppose R_1 *contains* R_2, then, by the formalization of the *contains* relation $P_1 = P_2 \wedge P'$. Contradiction.

Constraint 5: (R_1 Refines R_2) → ¬ (R_2 Contains R_1)
Proof: Let R_1 *refines* R_2. According to the formalization of the *refines* relation, q^1_1, q^1_2, …, q^1_{m-1}, q^1_m are the predicates such that $q^1_i \rightarrow q_i$ for $i \in 1..m$. Suppose R_2 *contains* R_1, then, by the formalization of the *contains* relation $P_2 = P_1 \wedge P'$. Contradiction.

Constraint 6: (R_1 Refines R_2) → ¬ (R_1 Conflicts R_2)
Proof: Let R_1 *refines* R_2, then, by the formalization of the refines relation $S_1 \subset S_2$. Suppose R_1 *conflicts* R_2, then, by the formalization of the *conflicts* relation $\neg\exists s : s \in S_1 \wedge s \in S_2 : P_1(s) \wedge P_2(s)$. Contradiction.

Constraint 7: (R_1 Contains R_2) ∧ (R_1 Refines R_3) → ¬ (R_2 Refines R_3)
Proof: Let R_1 contains R_2 and R_1 refines R_3 then, by the formalization of the *contains* and the *refines* relations:

(a) $P_1 = P_2 \wedge P'$
(b) $P_3 = p_1 \wedge p_2 \wedge ... \wedge p_{n-1} \wedge p_n \wedge q_1 \wedge q_2 \wedge ... \wedge q_{m-1} \wedge q_m$
(c) $P_1 = p_1 \wedge p_2 \wedge ... \wedge p_{n-1} \wedge p_n \wedge q^1_1 \wedge q^1_2 \wedge ... \wedge q^1_{m-1} \wedge q^1_m$ where q^1_1, q^1_2, …, q^1_{m-1}, q^1_m are the predicates such that $q^1_i \rightarrow q_i$ for $i \in 1..m$.

Suppose R_2 refines R_3, then, by the formalization of the *refines* relation:

(a) $P_3 = p_1 \wedge p_2 \wedge ... \wedge p_{n-1} \wedge p_n \wedge q_1 \wedge q_2 \wedge ... \wedge q_{m-1} \wedge q_m$

(b) $P_2 = p_1 \wedge p_2 \wedge ... \wedge p_{n-1} \wedge p_n \wedge q^2_1 \wedge q^2_2 \wedge ... \wedge q^2_{m-1} \wedge q^2_m$ where q^2_1, q^2_2, …, q^2_{m-1}, q^2_m are the predicates such that $q^2_i \rightarrow q_i$ for $i \in 1..m$.

Since $P_1 = P_2 \wedge P'$, P_1 includes all predicates in P_2. Contradiction.

These constraints above are derived from the formalization of the relations. There may be other types of constraints which originated from domain. For instance, one product quality requirement requires at least one functional requirement in the requirements model since product quality requirements come from the required characteristics of the software. These kinds of constraints are not explicitly stated in the requirements metamodel in Fig. 1. These can be defined using the OCL (Object Constraint Language) [30]:

```
1 -- all product quality requirements require at least one functional
2 -- requirement
3  context ProductQualityRequirement
4  inv: fromSource->size()>0 and
5       fromSource->forAll(rl|rl.oclIsTypeOf(Requires) and
6       rl.target->forAll(rq|rq.oclIsTypeOf(FunctionalRequirement))
```

In [7], we propose a prototyping that can perform reasoning on requirements that may detect implicit relations and inconsistencies on the basis of the formalization of relations and constraints. We also propose an approach for customizing the requirements metamodel in order to support different requirements specifications. Furthermore, our approach for customization keeps the semantics of the core concepts intact and thus allows reuse of tools and reasoning over the customized metamodel. We express the metamodels as OWL [5] ontologies. The composition operator is also expressed in OWL since this language allows direct mapping from set operations to language constructs. By using OWL we can use the reasoning capabilities of the ontology tools. We specified OWL [5] ontologies for each metamodel with Protégé [6] environment. Inference rules were expressed in SWRL [10]. The rules to check the consistency of relations were implemented as SPARQL [24] queries. The inference rules are executed by Jess rule engine [11] available as a plug-in in Protégé. To reason upon the requirements, the user specifies them as individuals (i.e., instances) in ontology. The inference and consistency checking rules are executed on this ontology.

4 Change Impact Analysis based on the Formalization of Relations

In this section, we give change impact rules for requirements based on the formalism of requirements relations. A change introduced to a model element can be in one of two phases [4]: "*A proposed change implies that impact analysis should be performed to determine how change would impact the existing system, whereas an implemented change implies that all impacted artifacts and their related links should be updated to reflect the change*". In this paper, we aim at giving some rules to the requirements engineer about the possible impacts of a proposed requirements change. For the

change impact analysis we also propose a distinction for impacted elements and candidate impacted elements: "*A candidate impacted element is the element identified as possibly impacted by a proposed change and it should be checked*". Another classification for impacted elements is direct/indirect impact. A direct impact occurs when the model element affected is related by one of the dependencies that fan-in/out directly to/from the changed model element [3]. An indirect impact occurs when the element is related by the set of dependencies representing an acyclic path between the changed and effected elements [3]. This type of impact is also referred as an N-level impact where N is the number of intermediate relationships. We propose step by step process to analysis the impacted elements. Requirements engineer first consider the directly impacted elements by using the impact rules with tool support, then do the changes in the impacted elements if needed. The previous indirectly impacted elements in 2-level impact are directly impacted elements in the second step. N-level impact analysis can be done by processing direct impact N-times in this way. These classifications for impacted elements are orthogonal. Table 1 gives the classification of impacted elements in the context of requirements modeling.

Table 1 Classification of Impacted Elements in the Context of Requirements Modeling

	Directly Impacted Elements	**Indirectly Impacted Elements**
Candidate Impacted Elements	All relations and requirements in 1-level impact should be considered to be updated.	All relations and requirements in N-level (N > 1) impact should be considered to be updated.
Actual Impacted Elements	All relations and requirements in 1-level impact are impacted and they must be updated.	All relations and requirements in N-level (N > 1) impact are impacted and they must be updated.

There are two types of changes in the requirements model:

- Changes in the requirements entities
  - A new requirement is added.
  - A requirement is deleted.
  - A requirement is modified (one or more of the predicates are deleted, or new predicates are added, or both)
- Changes in the requirements relations
  - A new relation is added.
  - A relation is deleted.
  - A relation is modified (type of the relation is changed)

Modifying a requirement is mainly about changing the text of requirement which is depicted by 'description' attribute of the Requirement entity in Fig. 1. There are other attributes of the Requirement entity such as 'name', 'priority', 'status' and 'reason'. However, we do not take into account changes in these attributes of a requirement in the paper. Since the Relation entity has only a type, changes in relations is only about changing types of the relation. Table 2 gives the change impact rules for requirements.

Table 2**.** Change Impact Rules for Requirements

		Case 1	**Case 2**	**Case 3**	**Case 4**
		R1 contains R2 and R3	**R1 refines R2**	**R1 requires R2**	**R1 conflicts R2**
sub a.	**R1 is deleted**	R2 and R3 are the candidate impacted requirements. The relation is impacted and it must be deleted.	R2 is not impacted. The refines relation is impacted and it must be deleted.	R2 is not impacted. The relation is impacted and it must be deleted.	R2 is not impacted. The conflicts relation is impacted and it must be deleted.
sub b.	**R2 is deleted**	R1 and R3 are candidate impacted requirements. The relation is impacted and it must be deleted.	R1 is the candidate impacted requirement. The relation is impacted and it must be deleted.	R1 is the impacted requirement. The relation is impacted and it must be deleted.	R1 is not impacted. The conflicts relation is impacted and it must be deleted.
sub c.	**R1 is modified**	R2 and R3 are the candidate impacted requirements. The relation is the candidate impacted relation.	R2 is not impacted. The relation is the candidate impacted relation.	R2 is not impacted. The relation is the candidate impacted relation.	R2 is not impacted. The conflicts relation is the candidate impacted relation.
sub d.	**R2 is modified**	R1 and R3 are the candidate impacted requirements. The relation is the candidate impacted relation.	R1 is the candidate impacted requirement. The relation is the candidate impacted relation.	R1 is the candidate impacted requirement. The relation is the candidate impacted relation.	R1 is not impacted. The relation is the candidate impacted relation.
sub e.	**New R added**	If R is a Product Quality Requirement then R1, R2, and R3 are candidate container requirements for R.	If R is a Product Quality Requirement then R1, R2 and R3 are candidate container requirements for R	If R is a Product Quality Requirement then R1, R2, and R3 are candidate container requirements for R.	If R is a Product Quality Requirement then R1, R2, and R3 are candidate container requirements for R.
sub f.	**Relation between R1 and R2 is deleted**	There is no impacted requirement. The relations inferred from it are the impacted relations.	There is no impacted requirement. The relations inferred from it are the impacted relations.	There is no impacted requirement. The relations inferred from it are the impacted relations.	There is no impacted requirement. The relations inferred from it are the impacted relations.
sub g.	**Relation between R1 and R2 is modified**	There is no impacted requirement. The relations inferred from it are the impacted relations.	There is no impacted requirement. The relations inferred from it are the impacted relations.	There is no impacted requirement. The relations inferred from it are the impacted relations.	There is no impacted requirement. The relations inferred from it are the impacted relations.
sub h.	**New relation is added to R1**	There is no impacted requirement.	There is no impacted requirement.	There is no impacted requirement.	There is no impacted requirement.

It should be noted that we only consider direct impacts for the change impact rules given in Table 2. We derive the change impact rules given in Table 2 from the formalizations of relations. Due to space limitation, we only explain some of them in the following.

Case 2 sub a: R_1 is deleted while R_1 *refines* R_2
Impact: Let R_1 *refines* R_2, then, by the formalization of the *refines* relation q^1_1, q^1_2, …, q^1_{m-1}, q^1_m are the predicates such that $q^1_i \rightarrow q_i$ for $i \in 1..m$. Deleting R_1 means deleting the predicates of R_1 including q^1_1, q^1_2, …, q^1_{m-1}, q^1_m. This change does not imply any impact on the predicates of R_2. We conclude that R_2 is not impacted. Since there should not be any dangling relation in the model, the *refines* relation is impacted and it must be deleted.

Case 2 sub b: R_2 is deleted while R_1 *refines* R_2
Impact: Let R_1 *refines* R_2, then, by the formalization of the *refines* relation q^1_1, q^1_2, …, q^1_{m-1}, q^1_m are the predicates such that $q^1_i \rightarrow q_i$ for $i \in 1..m$. Since R_1 includes more general predicates ($q^1_i \rightarrow q_i$), deleting the predicates of R_1 including p_i and q_i may imply that the predicates (p_i and q^1_i) in R_1 are impacted. We conclude that R_1 is the candidate impacted requirement. Since there should not be any dangling relation in the model, the *refines* relation is impacted and it must be deleted.

Case 3 sub b: R_2 is deleted while R_1 *requires* R_2
Impact: Since the definition of *requires* is given in extensional terms as a subset relation between the systems that satisfy the requirements, for every $s_1 \in S_1$ then $s_1 \in S_2$ when R_1 *requires* R_2. Deleting the predicates of R_2 may imply deleting or changing some of the predicates of R_1. We conclude that R_1 is the candidate impacted requirement. Since there should not be any dangling relation in the model, the *requires* relation is impacted and it must be deleted.

Case 1 sub c: R_1 is modified while R_1 *contains* R_2 and R_3
Impact: Let R_1 *contains* R_2 and R_3, then, by the formalization of the *contains* relation $P_1 = P_2 \wedge P_3 \wedge P'$. Modifying the predicate P_1 affects either the equation or the predicates P_2 or P_3. We conclude that R_2 or R_3 are the candidate impacted requirements and also the *contains* relation is the candidate impacted relation.

Case 2 sub c: R_1 is modified while R_1 *refines* R_2
Impact: Let R_1 *refines* R_2, then, by the formalization of the *refines* relation q^1_1, q^1_2, …, q^1_{m-1}, q^1_m are the predicates such that $q^1_i \rightarrow q_i$ for $i \in 1..m$. Changing the predicates of R_1 (p_i, q^1_i) does not affect the predicates of the more general requirement R_2 but may have an impact on the refines relation. The refines relation may not be valid anymore since the predicates of R_1 and R_2 may not ensure the formalism of the *refines* relation. We conclude that R_2 is not impacted. The refines relation is the impacted relation. It should be noted that we assume if there is a change in the general and refined requirements, requirements engineer always start changing requirements from the general ones.

Case 3 sub c: R_1 is modified while R_1 *requires* R_2
Impact: Since the definition of *requires* is given in extensional terms as a subset relation between the systems that satisfy the requirements, for every $s_1 \in S_1$ then $s_1 \in S_2$ when R_1 *requires* R_2. According to this subset relation R_1 has no implication on R_2. We conclude that R_2 is not impacted by modifying R_1. Since the subset relation may not be valid anymore, the *requires* relation is the candidate impacted relation.

Case 4 sub d: R_2 is modified while R_1 *conflicts* R_2
Impact: Since the definition of *conflicts* is given in extensional terms as an exclusive disjunction relation between the systems that satisfy the requirements ($\neg\exists s : s \in S_1 \wedge s \in S_2 : P_1(s) \wedge P_2(s)$), the changes in R_2 has no impact on R_1. Since the exclusive disjunction relation may not be valid anymore because of the change, the *conflicts* relation is the candidate impacted relation.

Case 2 sub d: R_2 is modified while R_1 *refines* R_2
Impact: Let R_1 *refines* R_2, then, by the formalization of the *refines* relation q^1_1, q^1_2, …, q^1_{m-1}, q^1_m are the predicates such that $q^1_i \rightarrow q_i$ for $i \in 1..m$. Modifying the predicates of R_2 (p_i, q_i) may have an impact on the predicates of R_1 (p_i, q^1_i) since there should be implication relation ($q^1_i \rightarrow q_i$) between refined predicates of R_1 & R_2. Or the refines relation may not be valid anymore. We conclude that R_1 is the candidate impacted requirement and *refines* is the candidate impacted relation.

Case 1 sub d: R_2 is modified while R_1 *contains* R_2 and R_3
Impact: Let R_1 *contains* R_2 and R_3, then, by the formalization of the *contains* relation $P_1 = P_2 \wedge P_3 \wedge P'$. Modifying the predicate P_2 affects either the equation or the predicates P_2 or P_3. We conclude that R_2 and R_3 are the candidate impacted requirements and also the *contains* relation is the candidate impacted relation.

5 Case Study Course Management System

In this section we apply the proposed approach in a case study. An existing requirements specification document is represented as a model instance of the requirements metamodel in Section 2. We also compared the benefits of our approach for change impact analysis with the benefits of having limited type of relations provided by some commercial tools such as IBM Rational RequisitePro. RequisitePro provides only two relations between requirements: *traceFrom* and *traceTo*. The relations in our requirements metamodel (e.g., the refines relation) must all be mapped to one of those two relations. The case study is about the requirements for Course Managements System. This system supports the basic facilities such as enrolling for a course, uploading roster and course materials, grading students, sending e-mails to students. The system supports three types of end users: administrator, student and lecturer. Fig. 2 gives the requirements model of the partial requirements specification of the course management system. Requirements for the model can be found in the appendix. Due to the page limitation and to simplify the

case study, we do not give the whole requirement specification in Fig. 2 and appendix.

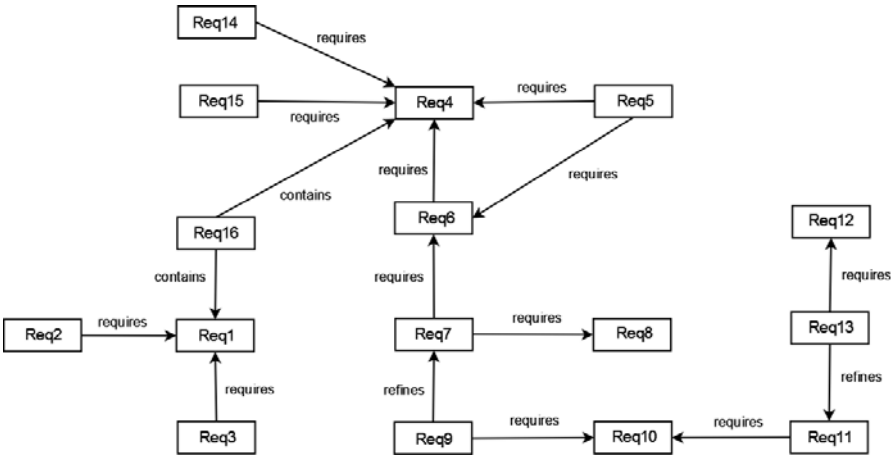

Fig. 2. Partial Requirements Model for the Course Management System Requirements Specification

We have three change scenarios (deleting Req6, modifying Req9 and Req11) in the requirements model given in Fig. 2 for the following requirements:

Req6. The system shall maintain a list of events the students can be notified about.

Req9. The system shall notify students about the events of the lectures they are enrolled for.

Req11. The system shall allow lecturers to send e-mail to students enrolled for the lecture given by that lecturer.

These are some of the change scenarios and impacts of the changes according to the change impact analysis given in Table 2:

Change 1: Deleting Req6

Impact: For the change in Req6, we consider the outgoing and incoming relations (Req6 requires Req4, Req5 requires Req6, Req7 requires Req6) of Req6 and the requirements (Req4, Req5, and Req7) related to Req6. According to the change impact rules given in Table 2, we determine the following impacts:

- *Req6 requires Req4 (Case 3 sub a)*: Since the required requirement is not impacted by deleting the requiring requirement, Req4 is not impacted. There should not be any dangling relation in the model. Therefore, the *requires* relation is impacted and it must be deleted.
- *Req5 requires Req6 (Case 3 sub b)*: Since the requiring requirement is the impacted requirement by deleting the required requirement, Req5 is impacted. It can not be satisfied by the system without specifying and satisfying Req6.
- *Req7 requires Req6 (Case 3 sub b):* Req7 is impacted. The *requires* relation is impacted and it must be deleted.

The directly impacted elements for *change 1* are Req5 and Req7. The second step is to determine indirectly impacted elements by *change 1*. The indirectly impacted elements in the first-step are directly impacted in this step.

- In the first step, we determine that Req5 is impacted by deleting Req6. When we analyze the impact Req5 can not be satisfied without satisfying Req6. We also decide to delete Req5. According to this change, we have the following impact:
  - *Req5 requires Req4 (Case 3 sub a)*: Req4 is not impacted. The *requires* relation is impacted and it must be deleted.
- We also determine that Req7 is impacted by deleting Req6. Req7 can not be satisfied without satisfying Req6. We also decide to delete Req7. We have the following impact:
  - Req7 *requires Req8 (Case 3 sub a)*: Req8 is not impacted. The *requires* relation is impacted and it must be deleted.
  - *Req9 refines Req7 (Case 2 sub b)*: Req9 is the candidate impacted requirement. The *refines* relation is impacted and it must be deleted.

When we analyze the direct impacts of *change 1* in RequisitePro, we do not have a distinction for types of relations. Therefore, we can not eliminate some of the related requirements to Req6. We identify all requirements (Req4, Req5 and Req7) related to Req6 as impacted requirements.

Change 2: Modifying Req9. We have the following requirement by modifying Req9:
Req9. *The system shall notify students about the events of school activities and lectures they are enrolled for*.
Impact: For the change in Req6, we consider the outgoing and incoming relations (Req9 refines Req7, Req9 requires Req10) of Req9 and the requirements (Req7 and Req10) related to Req9. We determine the following impacts:

- *Req9 refines Req7 (Case 2 sub c)*: Since the refined requirement is not impacted by modifying the refining requirement, Req7 is not impacted. Modifying the predicates of Req9 does not affect the predicates of the more general requirement Req7 but may have an impact on the refines relation. The refines relation may not be valid anymore since the predicates of Req9 and Req7 may not ensure the formalism of the *refines* relation. The refines relation is the candidate impacted relation.
- *Req9 requires Req10 (Case 3 sub c)*: Since the required requirement is not impacted by deleting the requiring requirement, Req10 is not impacted. Since the subset relation between Req9 and Req10 derived from the formalization of *requires* may not be valid anymore, the *requires* relation is the candidate impacted relation.

The candidate directly impacted elements for *change 2* are the *requires* and *refines* relations. The second step is to determine indirectly impacted elements by *change 2*. When we analyze the modification of Req9, we determine that these two relations are not impacted actually. Since we do not have any directly impacted requirements and relations, there are no indirectly impacted requirements and relations.

In RequisitePro, we identify all requirements (Req7 and Req10) related to Req9 as impacted requirements when we analyze the directly impacted elements. We can not identify the candidate impacted relations because we do not have the semantics of relations.

Change 3: Modifying Req11. We have the following requirement by modifying **Req11**: *The system shall allow lecturers to send e-mail and sms messages to students enrolled for the lecture given by that lecturer*.
Impact: For the change in Req11, we consider the outgoing and incoming relations (Req11 requires Req10, Req13 refines Req11) of Req11 and the requirements (Req10 and Req13) related to Req11. We determine the following impacts:

- *Req11 requires Req10 (Case 3 sub c)*: Since the required requirement is not impacted by deleting the requiring requirement, Req10 is not impacted. Since the subset relation between Req11 and Req10 derived from the formalization of *requires* may not be valid anymore, the *requires* relation is the candidate impacted relation.
- *Req13 refines Req11 (Case 2 sub d)*: Modifying the predicates of Req11 may have an impact on the predicates of Req13 since there must be implication relation between refined and refining predicates of R_1 & R_2. Or the refines relation may not be valid anymore. Req13 is the candidate impacted requirement and *refines* is the candidate impacted relation.

The candidate directly impacted elements for *change 3* are Req13 and *requires* & *refines* relations. When we analyze the modification of Req11, we determine that Req13 is impacted and we should add the feature of sms messages sending to Req13. Refines and requires relations which we identified as candidate impacted relations are still valid for *change 3*. The new Req13 is the following: *The system shall allow lecturers to send e-mail and sms messages to students in the same group*.

The second step is to determine indirectly impacted elements by *change 3*. Since the only impacted requirement is Req13, we analyze the impacted elements for Req13:

- *Req13 requires Req12 (Case 3 sub c)*: Since the required requirement is not impacted by deleting the requiring requirement, *Req12* is not impacted. *requires* relation is the candidate impacted relation.

In our approach, we can eliminate Req10 as not impacted but this requirement should be checked in RequisitePro since we identify all requirements (Req10 and Req13) related to Req11.

6 Related Work

Several authors address change impact analysis in the context of requirements modeling. In [27], a metamodel and an environment based on requirements are described. The tool supports graphical requirements models and automatic generation of Software Requirements Specifications (SRS). Their tool supports checking constraint violations for requirements models. However, they do not give any formal definition for their requirements relations and they do not support change impact analysis upon requirements and their relations.

Some authors [9] [23] use UML profiling mechanism for goal-oriented requirements engineering approach. Heaven et al. [9] introduce a profile that allows

the KAOS model [26] to be represented in UML. They also provide an integration of requirements models with lower level design models in UML. Supakkul et al. [23] use UML profiling mechanism to provide an integrated modeling language for functional and non-functional requirements that are mostly specified by using different notations. None of these study the formalization of relations and change impact analysis for requirements.

Ramesh et al. [20] propose models for requirements traceability. Models include basic entities like *Stakeholder*, *Object* and *Source*. Relations between different software artifacts and requirements are captured instead of relations between requirements.

In [21], an approach is proposed to define operational semantics for traceability in UML, which capture more precisely the intended meaning of various types of traceability. They claim that it will enable richer tool support for managing and monitoring traceability by making use of consistency checking technology. They define the semantic property of a traceability relationship with a triplet (event, condition, actions). Although they do not focus on a specific domain, their results are valid for change impact analysis on requirements models. Walderhaug et al. [29] propose a generic solution for traceability that offers a set of services that is meant to cover both specification and appliance of traceability. Their solution is specified as a trace metamodel with guidelines and templates. Van Gorph et al. [25] illustrate the need for developer tolerance of inconsistencies. This motivates the use of fine-grained consistency constraints and a detailed traceability metamodel. They are interested in managing inconsistencies between different model artifacts. However, they do not provide any techniques to determine the impacts within a model. Albinet et al. [1] explain how to define requirements according to a proposed requirements classification and they present tracing mechanisms based on the SysML UML 2.0 profile. They describe their methodology in order to take into consideration the expression of requirements, and their traceability along the software life-cycle.

Maletec et al. [17] describe an XML based approach to support the evolution of traceability links between models. They use a traceability graph to detect the dependency between model elements. However, they do not discuss change impact analysis. Luqi [16] uses graphs and sets to represent changes. Ajila [2] explicitly defines elements and relations between elements to be traced with intra-level and inter-level dependencies. Impact analysis based on transitive closures of call graphs is discussed in Law [13]. We have the transitive closure for requires, refines and contains relations between requirements. Lindvall et al. [15] show tracing across phases again with intra-level and inter-level dependencies. They also discuss an impact analysis based on traceability data of an object-oriented system. However, they do not support their analysis with formalism.

Change impact analysis for software architectures has been studied by Zhao et al. [28]. They use a formal architectural description language to specify and graphs to represent the architectures. They restrict their analysis to the architectural level and not for analysis level.

7 Conclusion

In this paper, we proposed a change impact analysis technique based on formalization of requirements relations within the context of Model Driven Engineering. We gave a requirements metamodel with well defined types of requirements relations. This metamodel represents the common concepts extracted from some prevalent requirements engineering approaches. The requirements relations in the metamodel were used to trace related requirements for change impact analysis. Using the formalization of these relations allowed us providing proofs of more precise rules for change impact analysis.

We applied our approach in a case study based on a requirements specification for course management system. We were able to determine candidate impacted requirements and relations with a better preciseness. Since we applied the approach to a limited number of requirements, the results may not be very convincing. However, applying it to a number of requirements like 300 requirements will make the benefit of our approach more explicit. On the other hand, we are aiming at a tool that provides semi-automatic support for change impact analysis based on the presented rules. Such a tool may use a Prolog engine to notify the requirements engineer about candidate and actual impacted elements by using these rules.

Determining the impact of requirements changes on inferred relations in [7] with tool support is another future work in evolution dimension. For the evolution of requirements, we also want to analyze the impact of requirements changes in architectural and detailed design. We need trace models in order to link requirement models to design models. These trace models will enable us to determine possible impacts of requirements changes in design models.

References

1. Albinet, A., Boulanger, J.L., Dubois, H., Peraldi-Frati, M.A., Sorel, Y., Van, Q.D.: Model-Based Methodology for Requirements Traceability in Embedded Systems. ECMDA TW 2007 Proceedings, Haifa, 2007.
2. Ajila, S.: Software maintenance: An approach to impact analysis of object change. Software - Practice and Experience, 25 (10), 1155-1181, 1995
3. Bohner, S.A.: Software Change Impacts – An Evolving Perspective. Proceedings of the International Conference on Software Maintenance (ICSM'02).
4. Cleland-Huang, J. et al.: Event-Based Traceability for Managing Evolutionary Change. IEEE Transactions on Software Engineering. Vol. 29, issue 9, 2003.
5. Dean, M., Schreiber, G., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P., Stein, L. A.: OWL Web Ontology Language Reference W3C Recommendation. (2004)
6. Gennari, J., Musen, A., Fergerson, R.W., Grosso, W.E., Crubezy, M., Eriksson, H., Noy, N.F., Tu, S.W.: The Evolution of Protégé: An Environment for Knowledge-Based Systems Development. International Journal of Human-Computer Studies, Volume 58, Issue 1, pp.89-123, January 2003
7. Goknil, A., Kurtev, I., van den Berg, K.: A Metamodeling Approach for Reasoning about Requirements. ECMDA-FA`08, LNCS, vol. 5095, pp. 311-326, 2008.

8. Guide to Software Engineering Body of Knowledge. IEEE Computer Society. http://www.swebok.org/ (last visit 06.02.2008)
9. Heaven, W., Finkelstein, A.: UML Profile to Support Requirements Engineering with KAOS. IEE Proceedings – Software, Vol. 151, No. 1, February 2004
10. Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: Semantic Web Rule Language – Combining OWL and RuleML. W3C, May, 2004
11. Jess, the Rule Engine for the Java Platform. http://herzberg.ca.sandia.gov/
12. Koch, N., Kraus, A.: Towards a Common Metamodel for the Development of Web Applications. ICWE 2003, LNCS, vol. 2722, pp. 497--506, Springer, Heidelberg (2003)
13. Law, J., Rothermel, G.: Whole program path-based dynamic impact analysis. International Conference on Software Engineering (ICSE`03), 2003
14. Lopez, O., Laguna, M.A., Garcia, F.J.: Metamodeling for Requirements Reuse. Anais do WER02 - Workshop em Engenharia de Requisitos, Valencia, Spain (2002)
15. Lindvall, M., Sandahl, K.: Traceability aspects of impact analysis in object-oriented systems. Software Maintenance: Research and Practice, 10, 37-57, 1998.
16. Luqi: A Graph Model for Software Evolution. IEEE Transactions on Software Engineering, 18 (8), 917-927.
17. Maletec, J.I., Collard, M.L., Simoes, B.: An XML based Approach to Support the Evolution of Model-to-Model Traceability Links. Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, 2005.
18. Meyer, J.J.C., Wieringa, R., Dignum, F.: The Role of Deontic Logic in the Specification of Information Systems. Logics for Databases and Information Systems, pp.71-115 (1998)
19. OMG: SysML Specification. OMG ptc/06-05-04, http://www.sysml.org/specs.htm
20. Ramesh, B., Jarke, M.: Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering, Vol. 27, No. 1 (2007)
21. Reshef, N., Paige, R. et al.: Operational Semantics for Traceability. ECMDA TW 2005 Proceedings, Nuremberg, November 8th, 2005.
22. Sommerville, I.: Software Engineering. Addison-Wesley, 7th Edition, 2004.
23. Supakkul, S., Chung, L.: A UML Profile for Goal-Oriented and Use Case-Driven Representation of NFRs and FRs. SERA'05 (2005)
24. SPARQL Query Language for RDF. W3C, January 2008. http://www.w3.org/TR/rdf-sparql-query/
25. van Gorph, P., Altheide, F., Janssens, D.: Traceability and Fine-Grained Constraints in Interactive Inconsistency Management. ECMDA TW 2006 Proceedings, Bilbao, 2006.
26. van Lamswerdee, A.: Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice. Invited Minitutorial, Proceedings RE'01 - 5th International Symposium Requirements Engineering, pp. 249-263, Toronto (2001)
27. Vicente-Chicote, C., Moros, B., Toval, A.: REMM-Studio: an Integrated Model-Driven Environment for Requirements Specification, Validation and Formatting. In Journal of Object Technology, Special Issue TOOLS Europe 2007, Vol. 6, No. 9, pp. 437-454 (2007)
28. Zhao, J., Yang, H., Xiang, L., Xu, B.: Change impact analysis to support architectural evolution. Journal of Software Maintenance and Evolution: Research and Practice, 14, 317-333, 2002
29. Walderhaug, S., Johansen, U., Stav, E.: Towards a Generic Solution for Traceability in MDD. ECMDA TW 2006 Proceedings, Bilbao, 2006.
30. Warmer, J., Kleppe, A.: Object Constraint Language: The Getting Your Models Ready for MDA. Addison-Wesley, 2003.

Appendix: Requirements for Course Management System

Req1. The system shall allow end-users to provide profile and context information for registration.
Req2. The system shall provide functionality to search for other people registered in the system.
Req3. The system shall provide functionality to allow end-users to log in the system with their password.
Req4. The system shall support three types of end-users (administrator, lecturer and student).
Req5. The system shall allow lecturers to set an alert on an event.
Req6. The system shall maintain a list of events the students can be notified about.
Req7. The system shall notify the students about the occurrence of an event as soon as the event occurs.
Req8. The system shall actively monitor all events.
Req9. The system shall notify students about the events of the lectures they are enrolled for.
Req10. The system shall allow students to enroll for lecturers.
Req11. The system shall allow lecturers to send e-mail to students enrolled for the lecture given by that lecturer.
Req12. The system shall allow assigning students to teams for each lecture.
Req13. The system shall allow lecturers to send e-mail to students in the same group.
Req14. The system shall allow lecturers to modify the content of the lectures.
Req15. The system shall give different access rights to different types of end-users.
Req16. The system shall support two types of end-users (lecturer and student) and it shall provide functionality to allow end-users to log in the system with their password.