

A Metamodeling Approach for Reasoning about Requirements

Arda Goknil, Ivan Kurtev, and 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. In requirements engineering, there are several approaches for requirements modeling such as goal-oriented, aspect-driven, and system requirements modeling. In practice, companies often customize a given approach to their specific needs. Thus, we seek a solution that allows customization in a systematic way. In this paper, we propose a metamodel for requirements models (called *core metamodel*) and an approach for customizing this metamodel in order to support various requirements modeling approaches. The core metamodel represents the common concepts extracted from some prevalent approaches. We define the semantics of the concepts and the relations in the core metamodel. Based on this formalization, we can perform reasoning on requirements that may detect implicit relations and inconsistencies. 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 illustrate the customization of our core metamodel with SysML concepts. As a case study, we apply the reasoning on requirements of an industrial mobile service application based on this customized core requirements metamodel.

Keywords: requirements metamodels, reasoning, model customization.

1 Introduction

Model Driven Engineering (MDE) considers models as primary engineering artifacts throughout the software development [11]. A software system is specified as a set of models that are repetitively refined until a model with enough details to implement the system is obtained.

Software development has different phases (requirement analysis, architectural design, detailed design, implementation and testing) which result in different artifacts. Currently, there exist standard modeling languages for expressing architecture, detailed design, and implementation of systems. Requirements descriptions, however, are considered mostly as textual artifacts with structure often not explicitly specified. Requirements descriptions are one of the earliest models of a system. In order to keep the continuum of models in MDE by treating every artifact as a model we need to represent requirements descriptions as models as well. To achieve this, developers need to employ a metamodel for requirements. However, it is difficult to propose a single and eventually standardized metamodel for requirements. There are several

commonly used approaches to represent requirements: goal-oriented [27] [16], aspect-driven [20], variability management [16], use case [3], domain-specific [12], and reuse-driven techniques [13]. Goal-oriented approach [27] defines a model for decomposing system goal into requirements with goal trees and offers some decision methods based on this decomposition. Aspect-oriented approach [20] gives a requirements model for separation of crosscutting functional and non-functional properties in requirements analysis phase.

A possible approach is to extract the common concepts from the existing techniques into a single metamodel. The current state of the requirements engineering practice shows that companies often adapt and customize a given approach to the company's specific needs. Thus, we need a solution that will allow us to achieve generality by using a set of common concepts and to allow customization in a systematic way.

In this paper, we propose a metamodel for requirements models (called *core* metamodel) and suggest an approach for customizing this metamodel in order to support different requirements specification techniques. We define the semantics of the concepts and the relations in the core metamodel. On the basis of the semantics we can perform reasoning on requirements that may detect implicit relations and inconsistencies. 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.

The core metamodel represents the common concepts extracted from some existing requirements modeling approaches [27] [15] [16] [18] [20] [28]. The customization of the core metamodel is based on set-theoretic operations. This ensures the validity of the results inferred from the customized requirements models by using the reasoning rules defined for the core metamodel. In the core metamodel we give the building blocks of a requirements specification. We are not interested in giving the details of requirements such as dynamic properties of target systems. Requirements engineer can always come up with his/her domain specific language for different types of requirements such as real-time specifications of embedded systems.

We illustrate our approach by customizing the core metamodel with SysML constructs. As a case study we model the requirements of an industrial mobile service application based on the customized metamodel.

The paper is structured as follows. In Section 2, we describe the customization approach for the requirements metamodels. Section 3 gives the details of the core requirements metamodel with the inference rules. Section 4 gives the details of SysML requirements metamodel. In Section 5 we describe the mappings between the two requirements metamodels. We also give the customized core requirements metamodel for SysML. In Section 6, we give a case study to illustrate the customization. Section 7 presents the related work. Section 8 summarizes the paper and describes future work.

2 Overview of the Customization Approach

In our approach the requirements engineer starts with the core requirements metamodel (See Fig. 1) and identifies the concepts that need specialization and concepts that has to be added. The result of the customization is a new requirements

metamodel. In Fig. 1, we use SysML as an example metamodel that specializes the core metamodel. The plus operator denotes the specification of the relations between the elements in the metamodels. These relations are based on set operations. An example is given in Section 5. Other metamodels for different approaches such as goal-oriented and aspectual requirements can be composed with the core requirements metamodel.

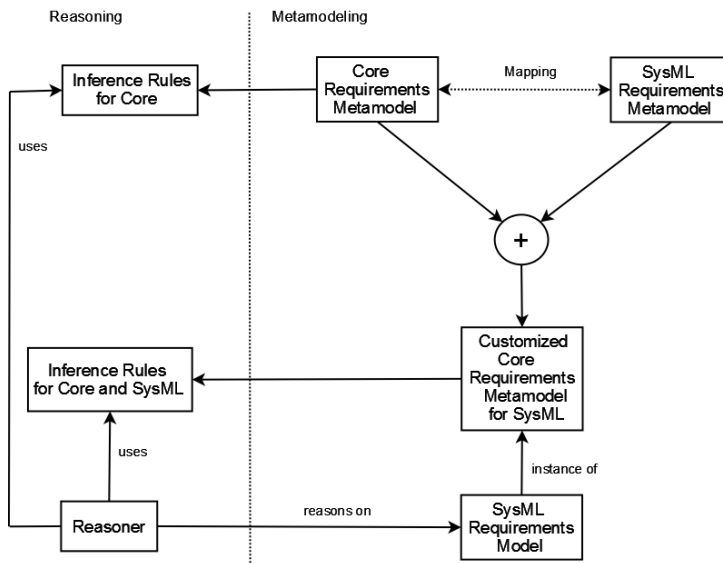


Fig. 1. Customization of Requirements Metamodels

In this paper we express the metamodels as OWL [4] ontologies. The composition operator is also expressed in OWL since this language allows direct mapping from set operations to the language constructs. By using OWL we can use the reasoning capabilities of the ontology tools. The aim of the approach is to specify generic inference rules for the core metamodel and to apply them for the customized metamodels (see left part of Fig. 1). Additional inference rules, specific for a given metamodel, may be added if needed.

3 Core Requirements Metamodel

The core requirements metamodel contains common concepts identified in existing requirements modeling approaches [27] [15] [16] [18] [20] [28]. The core metamodel in Fig. 2 includes entities such as *Requirement*, *Stakeholder* and *Relationship* in order to model general characteristics of requirements artifacts. They serve as extension points for possible customizations of the core 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

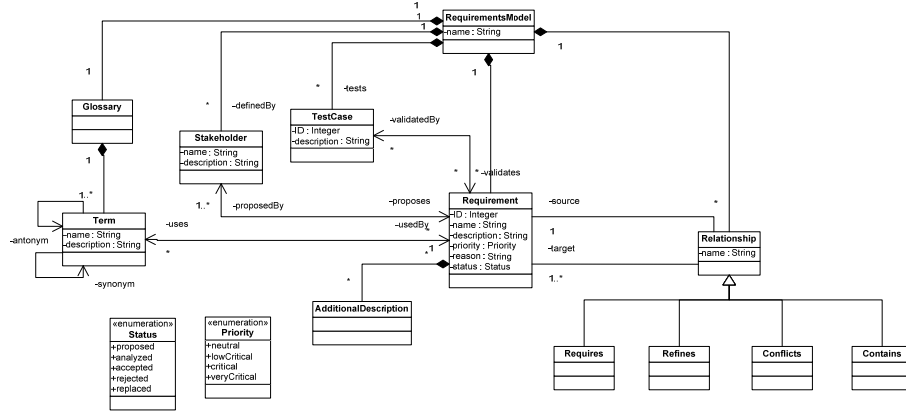


Fig. 2. Core Requirements Metamodel

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. Since there might be different classifications of requirements for different approaches, we decided not to give any further specialization of the *Requirement* concept in the core metamodel: this can be added in the customization. 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 [18] [28] consider test cases as a part of the requirements specification.

In order to specify relations between core and other requirements metamodels we give a set-theoretic interpretation of the core entities.

Let Core Requirements Metamodel (CRM) = {R, RS, RF, RQ, CF, CT, SH, TC, GS, T, AD} where the following abbreviations for the entities are used:

AD: AdditionalDescription	R: Requirement	SH: Stakeholder
CF: Conflicts	RF: Refines	T: Term
CT: Contains	RQ: Requires	TC: TestCase
GS: Glossary	RS: Relationship	

We assume that (a): all relations between requirements are the subset of relationship and (b): the intersection of these four relations is an empty set and the *Refines* relation is a subset of the *Requires* relation.

$$a: (RF \subseteq RS) \wedge (RQ \subseteq RS) \wedge (CF \subseteq RS) \wedge (CT \subseteq RS)$$

$$b: RF \cap RQ \cap CF \cap CT \equiv \phi \quad \wedge \quad RF \subseteq RQ$$

The relations in the core 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 [28].
- *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 [27].
- *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 [18].
- *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 [26].

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 ensure sound inference rules.

We assume the general notion of requirement being “a property which must be exhibited by a system” [7]. We define a requirement R as a tuple $\langle P, S \rangle$ 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 $R_1 = \langle P_1, S_1 \rangle$ and $R_2 = \langle P_2, S_2 \rangle$. 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 $R_1 = \langle P_1, S_1 \rangle$ and $R_2 = \langle P_2, S_2 \rangle$. 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 [14]) and P_2 can be represented in a conjunctive normal form in the following way:

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

Let $q^1_1, q^1_2, \dots, q^1_{m-1}, q^1_m$ 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 $i \in 1..m$ such that the following two statements hold:

- (a) $P_1 = p_1 \wedge p_2 \wedge \dots \wedge p_{n-1} \wedge p_n \wedge q^1_1 \wedge q^1_2 \wedge \dots \wedge q^1_{m-1} \wedge q^1_m$
- (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 $R_1 = \langle P_1, S_1 \rangle$, $R_2 = \langle P_2, S_2 \rangle$, and $R_3 = \langle P_3, S_3 \rangle$. 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:

$$P_2 = p_1 \wedge p_2 \wedge \dots \wedge p_{m-1} \wedge p_m$$

$$P_3 = p_{m+1} \wedge p_{m+2} \wedge \dots \wedge p_{n-1} \wedge p_n$$

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

$$P_1 = P_2 \wedge P_3 \wedge P' \text{ where } P' \text{ denotes properties that are not captured in } P_2 \text{ and } P_3 \text{ (i.e. we do not assume completeness of the decomposition [27])}$$

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 $R_1 = \langle P_1, S_1 \rangle$ and $R_2 = \langle P_2, S_2 \rangle$. Then, R_1 *conflicts with* R_2 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.

From the given definitions we may infer several rules that show how these three relations can be combined. We explore all combinations of requirements relations in the core metamodel in order to derive inference rules for requirements. Due to space limitation we do not give all combinations and inference rules for the relations. The rules are expressed in Semantic Web Rule Language (SWRL) [9] since OWL is not expressive enough in this case. The following example illustrates some of the rules on the basis of a concrete requirements specification document given in the WASP framework [21]. The example requirements (see Case Study in Section 6) are:

- *REQ_BDS_007*: When changes are discovered in the status and/or location of a user's body, the WASP platform must send out notifications according to the alerts set by the user.
- *REQ_NOT_006*: The WASP platform must notify the end-user about the occurrence of an event for which an alert was set, as soon as the event occurs.
- *REQ_NOT_009*: The WASP platform must actively monitor all events.

In the requirements document, the following relations are given: *refines* (*REQ_BDS_007*, *REQ_NOT_006*) and *requires* (*REQ_NOT_006*, *REQ_NOT_009*). When we apply the inference rules to the given requirements, we have inferred that *REQ_BDS_007* also requires *REQ_NOT_009* (dashed line in Fig. 3).

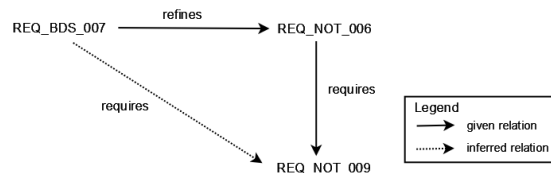


Fig. 3. Example with Given and Inferred Relations

We can formalize and proof these rules as follows:

Rule 1: $\text{refines}(R_1, R_2) \wedge \text{requires}(R_2, R_3) \rightarrow \text{requires}(R_1, R_3)$

Proof: Let $R_2 = \langle P_2, S_2 \rangle$ where $P_2 = p_1 \wedge p_2 \wedge \dots \wedge p_{n-1} \wedge p_n$ and $R_1 = \langle P_1, S_1 \rangle$. Since R_1 *refines* R_2 , from the definition we have that $P_1 = p_1 \wedge p_2 \wedge \dots \wedge p_{n-1} \wedge p_n \wedge q_1^1 \wedge q_1^2 \wedge \dots \wedge q_{m-1}^1 \wedge q_m^1$ and $q_i^1 \rightarrow q_i$ $i \in 1..m$. Again from the definition we have that if P_1 holds then P_2 also holds. From the *requires* relation between R_2 and R_3 we have that $S_2 \subset S_3$. Therefore if P_2 holds then P_3 also holds. Now we may conclude that if P_1 holds then P_3 also holds. This gives the subset relation $S_1 \subset S_3$ which proves that R_1 *requires* R_3 .

Rule 2: $\text{contains}(R_1, R_2) \wedge \text{requires}(R_2, R_3) \rightarrow \text{requires}(R_1, R_3)$

Proof: Let $R_1 = \langle P_1, S_1 \rangle$, $R_2 = \langle P_2, S_2 \rangle$, and $R_3 = \langle P_3, S_3 \rangle$

Since R_1 *contains* R_2 we have $S_1 \subset S_2$. From R_2 *requires* R_3 it follows that $S_2 \subset S_3$. Consequently $S_1 \subset S_3$. Similarly to the previous proof, we conclude that R_1 *requires* R_3 .

We can have implications for more combinations (e.g. three relations for four requirements and two conjunction operators) by using these inference rules.

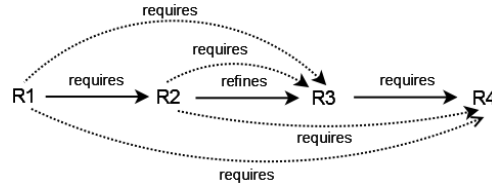


Fig. 4. Example with Inferred Relations by Combining Inference Rules

The relations shown with dash lines in Fig. 4 are inferred by using Rule 1, the transitivity of the relations, and the fact that *refines* implies *requires*. By combining these rules we have the following indirect relations:

$$\text{requires}(R_1, R_2) \wedge \text{refines}(R_2, R_3) \wedge \text{requires}(R_3, R_4) \rightarrow \text{requires}(R_2, R_3) \wedge \text{requires}(R_2, R_4) \wedge \text{requires}(R_1, R_3) \wedge \text{requires}(R_1, R_4)$$

Several rules for consistency checking are derived from the basic combinations where there is only one relation between two requirements. These inconsistencies are different from *conflicts* relation between requirements. Inconsistencies here indicate that relations between requirements are violating their constraints. Some of the consistency rules are given below:

- $\text{refines}(x_1, x_2) \rightarrow \neg \text{refines}(x_2, x_1)$
- $\text{refines}(x_1, x_2) \rightarrow \neg \text{requires}(x_2, x_1)$
- $\text{refines}(x_1, x_2) \rightarrow \neg \text{contains}(x_2, x_1)$

We specified OWL [4] ontologies for each metamodel with Protégé [6] environment. Inference rules were expressed in SWRL [9]. The rules to check the consistency of relations were implemented as SPARQL [24] queries. The inference rules are executed by Jess rule engine [10] 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 SysML Requirements Metamodel

The System Modeling Language (SysML) [18] is a domain specific modeling language for system engineering. It is defined as an extension of a subset of UML using UML’s profiling mechanisms. SysML provides modeling constructs to represent text-based requirements and relate them to other modeling elements with stereotypes. We apply the customization mechanism (see Fig. 1) on a metamodel for requirements used in SysML. (see Fig. 5). The requirements are represented as a requirements diagram, and have a name, a unique identifier (ID property), and a textual description. Requirements may be additionally described by use cases. There are also use case relations *Uses*, *Specializes* and *Extends*.

There are different types of requirements specified as an extension of *ExtendedReq* entity. They are *InterfaceReq*, *PerformanceReq* and *DesignConstraint*. Requirements may be related with each other with relations *Derives*, *Copies*, and *Contains*. The relations extend the concept *Trace*. Similarly to the core metamodel we interpret the metamodel elements as sets.

Let SysML Requirements Metamodel (SRM) = {R, US, AD, EX, SC, CP, FR, T, CT, IR, TC, DC, PR, UC, DV, PSR, UCR, ER} using the following abbreviations for the entities:

AD: AdditionalDescription	EX: Extends	SC: Specializes
CP: Copy	FR: FunctionalReq	T: Trace
CT: Contains	IR: InterfaceReq	TC: TestCase
DC: DesignConstraint	PR: PerformanceReq	UC: UseCase
DV: Derives	PSR: PhysicalReq	UCR: UseCaseRelation
ER: ExtendedReq	R: Requirement	US: Uses

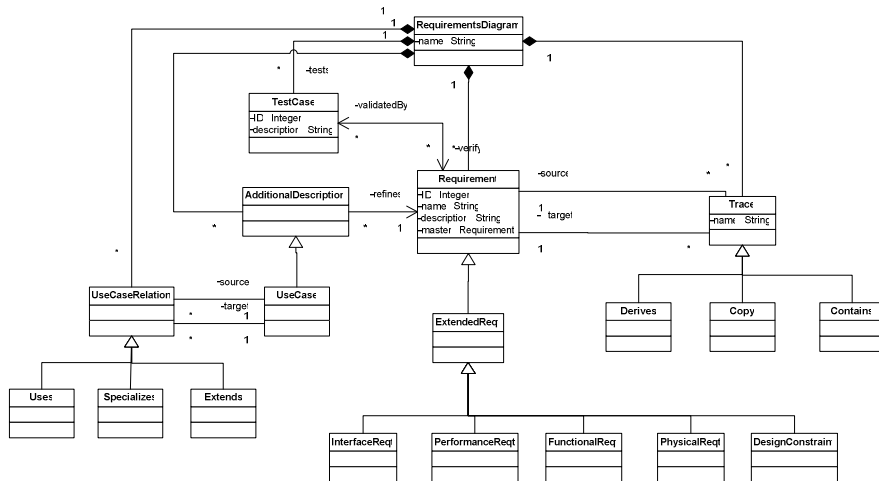


Fig. 5. SysML Requirements Metamodel

We assume that (a): Requirements types in SysML are subsets of *ExtendedReq* which is a subset of *Requirement*, (b): The intersection of all these requirements types is an empty set (they are disjoint), (c): Relations between requirements are the subset of relationship *Trace*, (d): the intersection of these relations is an empty set, (e): *UseCase* is a subset of *AdditionalDescription*, (f): Relations between use cases are the subset of relationship *UseCaseRelation*, and (g): the intersection of the relations between use cases is an empty set.

$$a : (IR \subseteq ER) \wedge (PR \subseteq ER) \wedge (FR \subseteq ER) \wedge (PSR \subseteq ER) \wedge (DC \subseteq ER) \wedge (ER \subseteq R)$$

$$b : IR \cap PR \cap FR \cap PSR \cap DC \equiv \emptyset$$

$$c : (DV \subseteq T) \wedge (CP \subseteq T) \wedge (CT \subseteq T)$$

$$d : DV \cap CP \cap CT \equiv \emptyset$$

$$e : UC \subseteq AD$$

$$f : (US \subseteq UCR) \wedge (SC \subseteq UCR) \wedge (EX \subseteq UCR)$$

$$g : US \cap SC \cap EX \equiv \emptyset$$

We introduce the following inference rules specific for SysML and not defined for the core metamodel. The relations *uses*, *extends* and *specializes* are transitive. Transitivity is captured in Rule 1:

$$\text{Rule 1. } \text{uses}(uc_1, uc_2) \wedge \text{uses}(uc_2, uc_3) \rightarrow \text{uses}(uc_1, uc_3)$$

If two use-cases are related, derived requirements from these use-cases are also related. Rule 2 specifies this:

$$\text{Rule 2. } \text{uses}(uc_1, uc_2) \wedge \text{hasAdditionalDescription}(\text{req}_1, uc_1) \wedge \text{hasAdditionalDescription}(\text{req}_2, uc_2) \rightarrow \text{requires}(\text{req}_1, \text{req}_2)$$

Since the concepts of use case and *uses* relation are not precisely defined in SysML, the inference rules 1 and 2 express the intuitive meaning we assigned to them. The formalization of SysML concepts needs further investigation.

5 Mappings between the Core and SysML Metamodels

In order to customize the core metamodel with SysML constructs we establish mappings between the elements in these metamodels. Mappings are specified as relations on sets. Some elements like *Requirement* in the core and *Requirement* in SysML are mapped directly. However, some elements e.g. *Derive* from SysML has no corresponding element in the core metamodel. Table 1 shows the mappings between core and SysML metamodels.

The semantically equivalent entities are related with set equality (e.g. rows 1, 5, 6). All specialized requirements in SysML are specializations of *Requirement* in the core metamodel (row 2). *Requires* (RQ) and *Conflicts* (CF) have no corresponding relation in SysML metamodel. All relations that have no corresponding relations in SysML metamodel are specializations of *Trace* (T) relation (rows 9 and 10). The relation

WASP (Web Architectures for Services Platforms), a framework for context-aware mobile services [21]. The requirements are identified using a three-step process of defining scenarios, use cases and requirements (see [21] for concrete details). There are 2 scenarios, 32 use cases and 81 requirements (70 functional and 2 non-functional; three of these requirements are decomposed into 9 sub-requirements).

We compared the reasoning facilities available in our approach with the similar support provided by IBM Rational RequisitePro. RequisitePro provides only two relations between requirements: *traceFrom* and *traceTo*. The relations in the customized metamodel (e.g., the uses relation) must all be mapped to one of those two relations. For example, links from requirements to use cases are mapped to *traceTo* links in RequisitePro and to *hasAdditional-Description* in our framework.

There is no explicit indication in the WASP requirements document for requirements relations. However, there are some keywords in the document to reference to other requirements. These keywords are “see also”, “implies”, “implied by” and “extension of”. We mapped them to the available relations in RequisitePro and our framework (see Table 2). In the 4th column, we indicate our choice for the directionality, e.g. for “implies” and “implied-by”.

Table 2. Mapping of Requirements Relations in Case Study

Document	RequisitePro	Our Framework	Directionality w.r.t. document
R_1 see (also) R_2	R_1 traceTo R_2	R_1 requires R_2	both the same
R_1 implies R_2	R_1 traceTo R_2	R_1 requires R_2	both the same
R_1 implied by R_2	R_2 traceTo R_1	R_2 requires R_1	both reversed
R_1 extension of R_2	R_1 traceTo R_2	R_1 refines R_2	both the same
R_1 example in R_2	R_1 traceTo R_2	R_2 refines R_1	ours reversed

Individual requirements in the document were represented as individuals in the OWL ontology in Protégé. The execution of the inference rules with the Jess rule engine inferred the implicit relations between requirements in the document. We also executed consistency rules to check the requirements relations (both given and inferred). The Jess rule engine was executed in two steps: a) with inference rules written for only the core requirements metamodel, b) with inference rules written for the customized metamodel for SysML. Table 3 shows given and inferred facts for requirements document of the WASP application.

Table 3. Given and Inferred Facts for the WASP Application Requirements

Facts	# R	# UC	# Relations R x R	# Relations R x UC	# Relations UC x UC
Given	81	32	20	103	24
Inferred in Step a	0	0	5	0	0
Inferred in Step b	0	0	735	0	4

Reasoning on the core metamodel (step a) resulted in 5 inferred relations between requirements. Since we do not have any inference rule for use cases in the core metamodel, we do not have any inferred relations between use cases and use cases & requirements. We executed the rules to check the consistency of the given and inferred requirements relations. We did not detect any inconsistency for these relations. The result reflects the accuracy of relations given in the document regarding the relation definitions we use for the core metamodel. We also checked the inferred relations manually if they correspond to a relation that can be identified by analyzing the textual requirements document. We found one inferred relation that is not true. When we traced from the inferred relation back to the given relations, we found that one given relation in the ontology has not a correct mapping to the requirement relations in the document. This is due to the assumption that links “see (also)” represent “requires” relations. However, we found that one of these links actually corresponds to “refines” relation. Our conclusion is that often the requirements engineers use links with ambiguous meaning or the links are not applied systematically.

The execution of the inference rules added by the SysML requirements meta-models (step b) resulted in 735 inferred relations between requirements and 4 inferred relations between use cases. The consistency check detected 16 inconsistent relations. The analysis of these inconsistencies revealed that they are caused by Rule 2 in Section 4. Rule 2 implies that if two requirements are related to two different use cases and one of these use cases uses another one, then there should be a “requires” relation between these requirements. When we checked the given relations in the requirements document, we realized that the interpretation of the requirements engineer for “uses” relation is different. There are given “requires” relations between requirements whose use cases are not related each other with “uses cases”. Therefore, Rule 2 does not capture the document structure properly and does not reflect the understanding of the requirements engineer. In order to apply the rules in practice, we should give the precise definition for each relation to requirements engineer and offer a guideline about how to specify these relations for more accurate reasoning results.

We compared the results in our framework with the results in RequisitePro. Table 4 gives the given and inferred relations in RequisitePro and our framework.

We observe more inferred relations between requirements and use cases in RequisitePro than in our framework. RequisitePro infers links on the base of the

Table 4. Given and Inferred Relations in RequisitePro and Our Framework

relations	# Given	# Inferred	# Inferred
# UC = 32; # R = 81	Document	RequisitePro	Our Framework
UC x UC	24	3	3
UC x R	103	98	0
Step a: R x R	9	1	5
# inconsistencies	-	-	0
Step b: R x R	9	-	735
# inconsistencies	-	-	16

transitivity of *trace* relations without considering the linked artifacts. For example, it assumes transitivity between R_1 and UC in case of $R_1 \text{ traceTo } R_2 \text{ traceTo UC}$, which is debatable. RequisitePro does not define any specific types of relations. This prohibits sophisticated reasoning based on various relation types and leads to some wrong inferred relations as seen in $UC \times R$. In our framework, the relation types and the inference rules allow us to have more precise inferred relations. Having types for relations also avoids finding non-meaningful relations inferred by RequisitePro.

7 Related Work

Several authors address requirements modeling in the context of MDE. In [28] a metamodel and an environment based on it are described. The tool supports graphical requirements models and automatic generation of Software Requirements Specifications (SRS). Baudry et al. [1] introduce a metamodel for requirements and present how they use it on top of a constrained natural language for requirements definition. In [2] they propose a model-driven engineering mechanism to merge different requirement specifications and reveal inconsistencies between them by using their core requirement metamodel. However, their core metamodel is mainly used to produce a global requirements model from a given set of texts. It does not specify entities and core relations and does not support customization.

Some authors [8] [25] use UML profiling mechanism in goal-oriented requirements engineering approach. Heaven et al. [8] introduce a profile that allows the KAOS model [27] to be represented in UML. They also provide an integration of requirements models with lower level design models in UML. Supakkul et al. [25] use UML profiling mechanism to provide an integrated modeling language for functional and non-functional requirements that are mostly specified by using different notations. SysML [18] also uses UML profiling mechanism to provide modeling constructs that represent text-based requirements and relate them to other modeling elements.

Koch et al. [12] propose a requirements metamodel that is specific to web systems. They do not consider general concepts for requirements analysis. They identify the general structure of web systems in order to define the requirements metamodel. Rashid et al. [20] give an activity model in requirements engineering for identifying and separating crosscutting functional and non-functional properties. Moon et al. [15] propose a methodology of producing requirements that can be considered as a core asset in the product line. Lopez et al. [13] propose a 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. Navarro et al. [17] propose a customization approach for requirements metamodels similar to ours. Their core metamodel is too generic and considers only artifact and dependency as core entities. It does not contain any entity specific to requirements. This prevents applying inference rules written for the core entities to customized entities. Requirements Interchange Format (RIF) [22] is a format which structures requirements and their attributes, types, access permissions and relationships. It is tool independent and defined as an XML schema. However, its data model has too generic entities and relations like *Information Type*, *Association*, and *Generalization* instead of entities that can be formalized to reason about requirements and their relations. Ramesh et. al [19] 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 core relations between requirements.

A number of approaches suggest reasoning about requirements. Zowghi et al. [29] propose a logical framework for modeling and reasoning about the evolution of requirements. Duff et al. [5] propose a logic-based framework for reasoning about requirements specifications based on goal-tree structures. Rodrigues et al. [22] propose a framework for the analysis of evolving specifications that can tolerate inconsistency by allowing reasoning in the presence of inconsistency.

8 Conclusion

There are several approaches for modeling requirements. These approaches are usually customized to serve specific needs and standards in industrial projects. In this paper, we proposed a metamodel for requirements and a customization approach in the context of Model Driven Engineering. Using metamodels for this customization allows us providing an environment for reuse of tools such as reasoners. The main concepts in our approach are the core requirements metamodel and the customization mechanism. We surveyed existing requirements modeling approaches to extract the core metamodel. We presented definitions and a formalization of requirements relations for the core metamodel. The customization mechanism is implemented on the basis of OWL properties in ontology.

We applied our approach in a case study based on a requirements specification document from a real project. We were able to infer several new relations that were not explicit in the document. We compared the capability of our approach to infer relations with the similar functionality provided by IBM RequisitePro, a commercial tool for requirements management. The relations in RequisitePro lack formal semantics. As a consequence, the inferred relations may not correspond to a “real” relation that may be discovered by inspecting the requirements document.

Since a wide range of inconsistencies can arise during requirements engineering, we did not elaborate on the *conflicts* relation in this paper. Some authors [26] review the main types of inconsistency and formalize them for specific cases. We plan to study definition and formalization of *conflicts* relation as future work. The impact of changes in requirements on inferred relations and checking the consistency of requirements against these changes are another future work in evolution dimension. For the evolution of requirements, we also want to analyze the impact of changes in requirements on architectural and detailed design. We need trace models to link requirement models to design models. These trace models will enable us to determine possible impacts of changes of requirements models on design models.

References

1. Baudry, B., Nebut, C., Le Traon, Y.: Model-Driven Engineering for Requirements Analysis. In: EDOC 2007, pp. 459–466. IEEE, Annapolis (2007)
2. Brottier, E., Baudry, B., Le Traon, Y., Touzet, D., Nicolas, B.: Producing a Global Requirement Model from Multiple Requirement Specifications. In: EDOC 2007, pp. 390–404. IEEE Computer Society Press, Annapolis (2007)

3. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley, Reading (2000)
4. 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)
5. Duffy, D., MacHish, C., McDermid, J., Morris, P.: A Framework for Requirements Analysis Using Automated Reasoning. In: Iivari, J., Rossi, M., Lyytinen, K. (eds.) CAiSE 1995. LNCS, vol. 932. Springer, Heidelberg (1995)
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* 58(1), 89–123 (2003)
7. Guide to Software Engineering Body of Knowledge. IEEE Computer Society, Los Alamitos (last visit 06.02.2008), <http://www.swebok.org/>
8. Heaven, W., Finkelstein, A.: UML Profile to Support Requirements Engineering with KAOS. *IEE Proceedings – Software* 151(1) (February 2004)
9. Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosz, B., Dean, M.: SWRL: Semantic Web Rule Language – Combining OWL and RuleML. W3C (May 2004)
10. Jess, the Rule Engine for the Java Platform, <http://herzberg.ca.sandia.gov/>
11. Kent, S.: Model Driven Engineering. In: Proceedings of the 3rd International Conference on Integrated Formal Methods, London, UK, pp. 286–298 (2002)
12. Koch, N., Kraus, A.: Towards a Common Metamodel for the Development of Web Applications. In: Cueva Lovelle, J.M., Rodríguez, B.M.G., Gayo, J.E.L., Ruiz, M.d.P.P., Aguilar, L.J. (eds.) ICWE 2003. LNCS, vol. 2722, pp. 497–506. Springer, Heidelberg (2003)
13. Lopez, O., Laguna, M.A., Garcia, F.J.: Metamodeling for Requirements Reuse. In: Anais do WER 2002 - Workshop em Engenharia de Requisitos, Valencia, Spain (2002)
14. 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*, 71–115 (1998)
15. Moon, M., Yeom, K., Chae, H.S.: An Approach to Developing Domain Requirements as a Core Asset Based on Commonality and Variability Analysis in a Product Line. *IEEE Transactions on Software Engineering* 31(7) (2005)
16. Mylopoulos, J., Chung, L., Yu, E.: From Object-Oriented to Goal Oriented Requirements Analysis. *Communications of the ACM* 42(1) (1999)
17. Navarro, E., Mocholi, J.A., Letelier, P., Ramos, I.: A Metamodeling Approach for Requirements Specification. *The Journal of Computer Information Systems* 46(5), 67–77 (2006)
18. OMG: SysML Specification OMG ptc/06-05-04, <http://www.sysml.org/specs.htm>
19. Ramesh, B., Jarke, M.: Toward Reference Models for Requirements Traceability. *IEEE Transactions on Software Engineering* 27(1) (2007)
20. Rashid, A., Moreira, A., Araujo, J.: Modularization and Composition of Aspectual Requirements. In: AOSD 2003, Boston, United States, pp. 11–20 (2003)
21. Requirements for the WASP application Framework, https://doc.telin.nl/dsweb/Get/Document-27861/WASP_D2.1_version_1.0_Final.pdf
22. Requirements Interchange Format (RIF), <http://www.automotive-his.de/rif/doku.php>
23. Rodrigues, O., Garcez, A., Russo, A.: Reasoning about Requirements Evolution using Clustered Belief Revision. In: Bazzan, A.L.C., Labidi, S. (eds.) SBIA 2004. LNCS (LNAI), vol. 3171. Springer, Heidelberg (2004)
24. SPARQL Query Language for RDF. W3C (January 2008), <http://www.w3.org/TR/rdf-sparql-query/>

25. Supakkul, S., Chung, L.: A UML Profile for Goal-Oriented and Use Case-Driven Representation of NFRs and FRs. In: SERA 2005 (2005)
26. van Lamswerde, A., Darimont, R., Letier, E.: Managing Conflicts in Goal-Driven Requirements Engineering. *IEEE Transactions on Software Engineering* 24(11) (November 1998)
27. van Lamswerde, A.: Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice. In: Invited Minitutorial, Proceedings RE 2001 - 5th International Symposium Requirements Engineering, Toronto, pp. 249–263 (2001)
28. Vicente-Chicote, C., Moros, B., Toval, A.: REMM-Studio: an Integrated Model-Driven Environment for Requirements Specification, Validation and Formatting. *Journal of Object Technology, Special Issue TOOLS Europe 2007* 6(9), 437–454 (2007)
29. Zowghi, D., Offen, R.: A Logical Framework for Modeling and Reasoning about the Evolution of Requirements. In: RE 1997, Annapolis, USA (January 1997)