A Metamodeling Approach for Reasoning on Multiple Requirements Models

Arda Goknil
AOSTE Research Team
UNS-I3S-INRIA
Sophia-Antipolis, France
arda.goknil@inria.fr

Ivan Kurtev
Nspyre
Dillenburgstraat 25-3,
5652 AM Eindhoven, the
Netherlands
ivan.kurtev@nspyre.nl

Jean-Vivien Millo
AOSTE Research Team
UNS-I3S-INRIA
Sophia-Antipolis, France
jean-vivien.millo@inria.fr

***Abstract*—The complex software development projects of today may require developers to use multiple requirements engineering approaches. Different teams may have to use different requirements modeling formalisms to express requirements related to their assigned parts of a given project. This situation poses difficulties in achieving interoperability and integration of requirements models for the purpose of reasoning on the overall system requirements. It is challenging to compose distributed models expressed in different notations and to reason on the composed models. In this paper we present a metamodeling approach which allows reasoning about requirements and their relations on the whole/composed models expressed in different requirements modeling approaches. In a previous work we expressed the structure of requirements documents as a requirements metamodel in which the most important elements are requirements relations and their types. The semantics of these elements is given in First Order Logic (FOL) and allows two activities: inferring new relations from the initial set of relations and checking consistency of relations. In this work we use the requirements metamodel as a core metamodel to 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 requirements modeling approaches. The specialization allows using the same semantics and reasoning mechanism of the core metamodel for multiple requirements modeling approaches. To illustrate the approach we use an example from automotive domain expressed with two modeling approaches: product-line requirements models and SysML for system requirements.**

Keywords-requirements metamodel; requirements reasoning; model-driven engineering, product-line requirements, SysML

I. INTRODUCTION

With the globalization, the development of complex software systems is often distributed over multiple third-party software development companies with multiple development teams in different locations. The complex software development projects of today may require developers to use multiple requirements engineering approaches. While one team specializes in product-line requirements of the overall system (mandatory and optional requirements with their dependencies), other teams may analyze and develop any specific part of the system. Therefore, different teams may need to use different requirements modeling approaches to express requirements related to their assigned parts of a given project: informal (interviews, surveys, textual requirements, etc.), semi-formal (Product-line, SysML, etc.) and formal (Deontic logic, B-method, etc.). This situation poses difficulties in achieving interoperability and integration of requirements models for the purpose of reasoning on the overall system requirements. It is challenging to compose distributed models expressed in different notations and to reason on the composed models.

To address these issues, researchers have developed techniques and tools that support the consolidation of multiple requirements specifications. Some techniques are based on formal approaches such as argumentation theory [13] [18], B-method [22] and conceptual ontologies [17]. Other techniques use the model-driven engineering approach to merge different requirement specifications by using a requirements metamodel [14] [16]. For instance, Brottier et al. [14] introduce a metamodel where a requirement is decomposed into fragments as *condition* and *consequence*. The requirements from different specifications are merged in the level of these fragments. One *consequence* in one specification can be a *condition* or again a *consequence* in the second specification. In all these works considerable research has been devoted to consolidating multiple requirements models and checking their consistency. Less attention has been paid to requirements relations in multiple models in different languages and reasoning about these relations.

In this paper we present a metamodeling approach which allows reasoning about requirements and their relations on the whole/composed models expressed in different requirements modeling notations. In our previous work [3] [4], to capture the structure of requirements documents explicitly we propose to encode them as models that conform to a requirements metamodel [3]. 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 the consistency of relations. Both the initially given and the inferred sets of relations can be used in several development activities such as

determining the propagation of a change from one requirement to another (change impact analysis). A *Tool for Requirements Inferencing and Consistency Checking* (TRIC) [3] [28] [29] is developed to support both activities. The requirements relations with their properties are mapped to Web Ontology Language (OWL) in the implementation of TRIC. This allows a uniform semantic domain that is formal and supported by tools. The details about the requirements metamodel, the semantics and the tool support are reported in [3]. In this work we use the requirements metamodel as a core metamodel that is specialized for multiple requirements modeling approaches such as Product-line [23] and SysML [25]. Mainly, the requirements relations in the metamodel are specialized to support relations in these approaches. The specialization allows using the same semantics and the reasoning mechanism of the core metamodel for multiple requirements modeling approaches. Since the core metamodel is general enough, the available reasoning is valid for the specialized concepts and can be extended by adding language specific rules. To illustrate the approach we use an example from the automotive domain modeled with two modeling approaches: product-line requirements models and SysML for system requirements. First, the core metamodel is specialized to model product-line requirements with their relations. New relations among product-line requirements are inferred and the consistency of the relations is checked. Second, the core metamodel is specialized to model the system requirements in SysML. The reasoning is performed for the requirements relations. Finally, we use the relation types in the core metamodel to relate the product-line and SysML requirements. This enables us to reason about the overall set of relations for both product-line and SysML requirements. A potential limitation of our approach is the expressiveness of the semantic domain (in our case it is OWL) but our example shows the feasibility for the commonly used languages product-line models and SysML.

The contributions of the paper can be summarized as follows:

- Support for reasoning: the available reasoning engine can be reused for languages whose constructs can be expressed in terms of the elements in the core metamodel. This is shown for SysML and product-line requirements models.
- Language constructs with unclear semantics can be mapped to the well-defined elements in the core metamodel. In the paper we show how the SysML relation 'deriveReqt' can be interpreted in different ways thus improving the clarity of models and eliminating inconsistencies.
- The mapping to the core metamodel provides a common semantic domain for languages. This allows reasoning on the composition of models expressed in the languages like SysML and product-line models.

This paper is organized as follows. Section II gives an overview of the approach. Section III briefly introduces the elements of the core metamodel. In Section IV we illustrate the specialization of the core metamodel for product-line requirements with the reasoning support. Section V presents the SysML specialization. Section VI explains how our solution supports reasoning on the composed models. Section VII compares our work with the existing results. In Section VIII we conclude the paper.

II. Overview of the Approach

The approach is based on the core requirements metamodel [2] [3] [4] in which the most important elements are requirements relations and their types. The semantics of relations enables reasoning: *consistency checking of relations* and *inferencing new relations based on the given relations*. We also use the relations in the core metamodel to compose requirements models in different notations (see Figure 1).

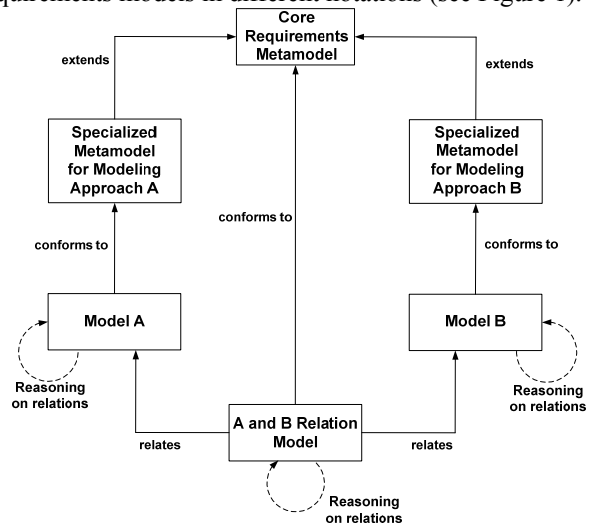

Figure 1 Overview of the Approach

In order to achieve reasoning on multiple requirements models the requirements engineer takes the following steps:

- **Reasoning in Model A**. The metamodel is specialized to model requirements and their relations in any requirements modeling approach (Modeling Approach A). The reasoning mechanism is performed on any model which is an instance of the specialized metamodel. We illustrate this step with the specialization of the core metamodel for product line requirements (see Section IV).
- **Reasoning in Model B.** Another specialization is to support requirements and relations in any second modeling approach (Modeling Approach B). The reasoning is performed on any model which is an instance of the specialized metamodel. To illustrate this step the core metamodel is specialized for SysML (see Section V).
- **Reasoning in the Composed Models (Model A & Model B).** The relation types in the core metamodel are used to compose the two models. This composition allows reasoning about the overall set of relations in Model A, Model B and between Model A & Model B (see Section VI).
- **Iterating.** The approach is iterative. The requirements engineer may model additional requirements with other approaches and compose the models with the existing composed models.

III. CORE REQUIREMENTS METAMODEL

The core metamodel contains common entities identified in the literature on requirements modeling. Mainly, the metamodel defines the requirement entity with its attributes and requirements relations (see Figure 2).

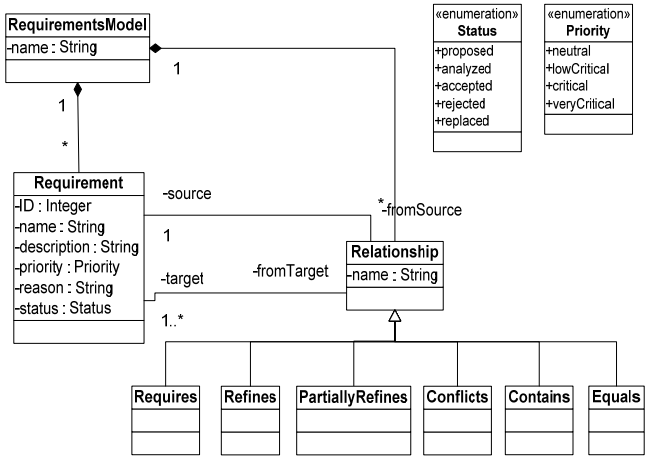

Figure 2 Core Requirements Metamodel

The requirements are captured in a *requirements model*. A requirements model contains *requirements* and their *relations*. Based on [6] we define a requirement as follows:

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

A requirement has a unique identifier (ID), name, textual description, priority, rationale, and status. A system property can be a certain functionality or any quality attribute. In this respect, our requirements relation types and their formalization are applicable to both functional and non-functional requirements.

We identified six types of relations: *requires*, *refines*, *partially refines*, *contains*, *conflicts*, and *equals*. In the literature, these relations are informally defined as follows.

- **Definition 2.** *Requires relation:* A requirement R1 requires a requirement R2 if R1 is fulfilled only when R2 is fulfilled.

The *requires* relation is *non-reflexive*, *non-symmetric* and *transitive*. We explain the *requires* relation with the following two requirements.

PR11: The automobile should have an electric engine.
PR17: The energy reservoir of the automobile should be an accumulator.

In order to run an electric engine, the automobile needs an accumulator as an energy reservoir. Therefore, we conclude that PR11 requires PR17 to be fulfilled.

- **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. The *refines* relation allows organizing requirements at various levels of abstraction to manage complexity in large documents. Similarly to the *requires* relation we have the properties *non-reflexive*, *non-symmetric*, and *transitive* for the *refines* relation. We explain the *refines* relation with the following two requirements.

PR1: The automobile should have an air conditioning.
PR2: The air conditioning in the automobile should be with climate control.

PR1 only indicates the availability of air conditioning in the automobile. PR2 specifies a concrete detail of the air conditioning: the control of the air conditioning is climate control. Therefore, we conclude that PR2 refines PR1. Note also that PR2 requires PR1 to be fulfilled. The *refines* relation implies the *requires* relation.

- **Definition 4.** *Contains relation*: A requirement R_1 *contains* requirements $R_2 \dots R_n$ if $R_2 \dots R_n$ 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*. The *contains* relation is *non-reflexive*, *non-symmetric*, and *transitive*. For this relation, all parts are required in order to fulfill the composing requirement.

We explain the *contains* relation with the following two requirements.

PR31: The automobile should have roof control and rear wiper.
PR26: The automobile should have a rear wiper.

PR31 states that the automobile should have two different facilities as *roof control* and *rear wiper*. The requirement can be interpreted as two different properties for the system. PR26 states only one of these properties, which is *having a rear wiper*. Therefore, we conclude that PR26 is one of the decomposed requirements of PR31 (PR31 contains PR26). It is also noted that PR31 requires PR26 to be fulfilled.

- **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.

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 special combination of decomposition and refinement. It is mainly drawn from the decomposition of goals in goal-oriented requirements engineering [7]. The *partially refines* relation is *non-reflexive*, *non-symmetric*, and *transitive*.

We consider the following requirements for the explanation of the *partially refines* relation.

PR31: The automobile should have roof control and rear wiper.
PR24: The roof of the automobile should be controlled by rain sensors.

In PR31, it is stated that the automobile should provide two system properties as controlling roof and wiping the rear windows. PR24 specifies a concrete detail of only one of these properties and excludes the second property: the roof

control should be done with rain sensors. Therefore, we conclude that PR24 partially refines PR31.

- **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.

The *conflicts* relation addresses a contradiction between requirements. 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 binary *conflicts* relation is *symmetric* and *non-reflexive*. It is not *transitive*.

We explain the *conflicts* relation with the following two requirements.

PR9: The automobile engine should be gasoline.
PR10: The automobile engine should be diesel.

The combustion in the automobile engine can be either with gasoline or diesel. The fulfillment of PR9 excludes the fulfillment of PR10 and vice versa. Therefore, we conclude that PR9 conflicts with PR10.

- **Definition 7.** *Equals relation*: A requirement R_1 *equals to* a requirement R_2 if R_1 states exactly the same properties with their constraints with R_2 and vice versa.

This relationship enables a requirement to be copied in a second requirement. The *equals* relation is *symmetric*, *non-reflexive* and *transitive*.

The definitions given above are informal. Our tool uses formal semantics of requirements relations that allows performing reasoning tasks. The semantics of the relations is given in FOL. Requirements are interpreted as formulae in a fragment of FOL where all the formulae are in a conjunctive normal form (CNF). Requirements relations are mapped to relations between the formulae. Since the rules of formula relations can be directly mapped to OWL, we use an OWL reasoner in the implementation of TRIC. For the detailed description of the semantics, the reader is referred to our previous work [2] [3] [4]. The example in Figure 3 gives an inferencing example for the following requirements:

PR11: The engine should be electric.
PR16: The automobile must have an energy reservoir.
PR17: The energy reservoir should be accumulator.

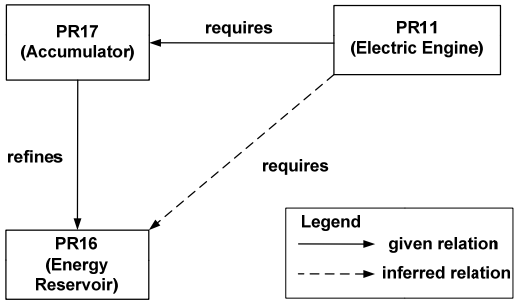

Figure 3 Example Inferred Relation

The following relations are given: (PR11 requires PR17) and (PR17 refines PR16).When we run our tool over the requirements model, the relation (PR11 requires PR16) is inferred automatically. Since the *refines* relation implies the *requires* relation, PR17 also requires PR16. The *requires* relation is transitive. Therefore, (PR11 requires PR17) and (PR17 requires PR16) implies (PR11 requires PR16).

TRIC is also able to detect relations that are contradicting. Figure 4 gives example inconsistent relations.

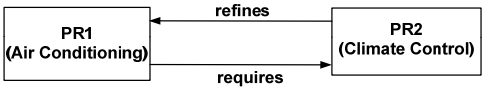

Figure 4 Example Inconsistent Relations

The *refines* and *requires* relations between two requirements in the opposite direction cause a contradiction. Since the *refines* relation implies the *requires* relation, PR2 also requires PR1. The *requires* relation is non-symmetric. Therefore, two requirements cannot require each other. One of the relations in the example is invalid. When we analyze the requirements, we conclude that PR2 refines PR1 but PR1 does not require PR2 to be fulfilled because air conditioning might be manual (see Definition 2 and Definition 3).

IV. Reasoning on Product-Line Models

In this section we show how we express product-line requirements and their relations in terms of the core metamodel with some extensions (see Figure 5).

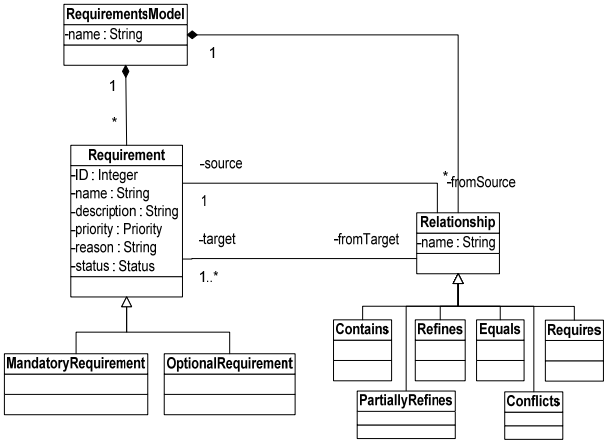

Figure 5 Specialized Metamodel for Product-line Requirements

The *requirement* entity is specialized as *Mandatory Requirement* and *Optional Requirement* which support the variant and variations. We use a product-line model from the automotive domain as an example (see Figure 6). The requirements of the example can be found in [30].

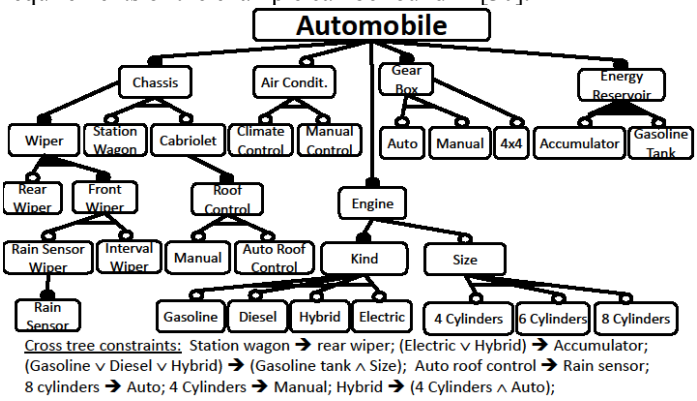

Figure 6 Product-line Model for the Automotive Example

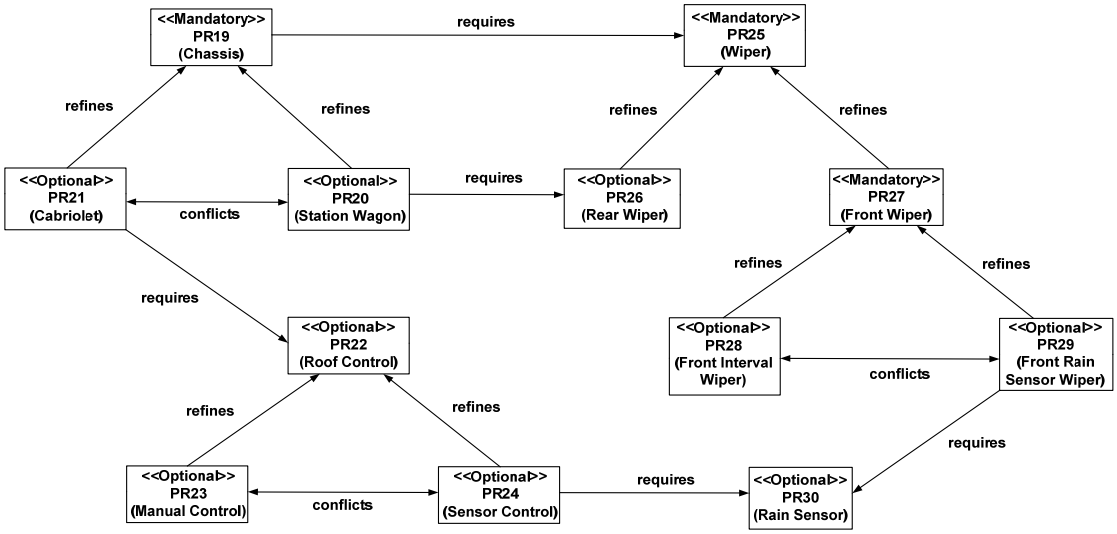

Figure 7 Part of the Requirements Model for the Automotive Product-line Requirements

The feature-subfeature relations can be interpreted in multiple ways. Based on the semantics of the specialized metamodel and product-line models [9] we provide some rules for expressing product-line requirements in TRIC. Figure 7 gives a part of the example product-line model in TRIC.

- ***XOR-group.*** This is mapped to *refines* and pairwise *conflicts*. The semantics of XOR-group is that: (1) every feature in the group implies the parent; (2) the parent implies exactly one of the elements in the group. (1) holds because the *refines* relation implies the *requires* relation and *requires* is equivalent to the propositional implication. (2) holds because at least one of the refining requirements must hold and conflicts relations guarantee that it is exactly one. In Figure 7 the chassis (PR19) cannot be both cabriolet and station wagon (PR21 conflicts with PR20).
- ***OR-Group.*** All subfeatures in the group refine the parent feature. In Figure 7 an automobile may have both rear wiper (PR26) and front wiper (PR27) where the front wiper is mandatory.
- ***Mandatory feature.*** For every concrete mapping of the mandatory feature, the relation between the requirements has to guarantee that if the parent requirement is satisfied, the child requirement should be satisfied as well. If the child requirement is satisfied then the parent requirement is also satisfied. A mandatory feature is translated as a mandatory requirement in TRIC only if the mandatory feature holds in every product configuration derived from the product-line model. For instance, in Figure 7, the roof control is an optional requirement since it is needed only if the automobile has a cabriolet chassis. The feature-subfeature relation between the cabriolet and the roof control features is mapped to the *requires* relation (PR21 requires PR22). Therefore, the child requirement (the roof control) has to be chosen when the parent requirement (the cabriolet) is chosen for any product configuration. On the other hand, the front wiper is a mandatory subfeature and it is translated as a mandatory requirement because its parent features 'chassis' and 'wiper' hold for every product configuration. The feature-subfeature relation between the wiper and the front wiper features is mapped to the *refines* relation (PR27 refines PR25).
- ***Optional feature.*** The semantics of optional features states that if the child requirement is satisfied the parent is also satisfied. The *requires* and *refines* relations can be used to encode relations between a feature and an optional feature.
- ***Cross-tree constraints.*** They are mapped to the *requires* and *conflicts* relations. For instance, (station wagon → rear wiper) is mapped to (PR20 requires PR26).

PR19: The automobile must have a chassis (mandatory).
PR20: The chassis should be a station wagon type (optional).
PR21: The chassis should be a cabriolet type (optional).
PR22: The automobile should have a roof control (optional).
PR23: The roof control should be manual control (optional).
PR24: The roof control should be sensor control (optional).
PR25: The automobile must have wiper (mandatory).
PR26: The automobile should have rear wiper (optional).
PR27: The automobile must have front wiper (mandatory).
PR28: The automobile should have front interval wipe (optional).
PR29: The automobile should have front rain sensor wipe (optional).
PR30: The automobile should have rain sensors (optional).

The example in Figure 8 illustrates the inferencing for the following requirements:

PR5: The gearbox should be with manual gear (optional).
PR6: The gearbox should be with automatic gear (optional).
PR15: The engine should be 8-cylinder (optional).

The following relations are given: (PR6 conflicts PR5) and (PR15 requires PR6).The relation (PR5 conflicts PR15) is inferred (dashed line in Figure 8). An 8-cylinder engine requires an automatic gear in an automobile where we cannot have a manual gear. Therefore, we confirm that the inferred relation (PR5 conflicts PR15) is a valid relation in the model.

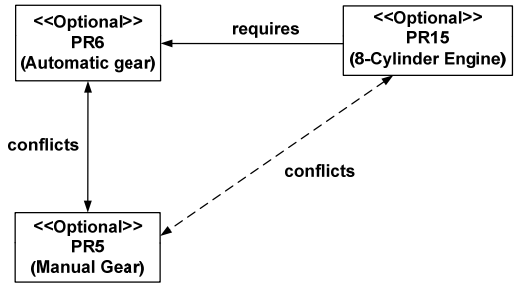

Figure 8 Example Inferred Conflicts Relation

The interpretation of requirements depends on the requirements engineer. In the example, we discovered some invalid given relations. TRIC helps to identify invalid given relations in the example (see Figure 9).

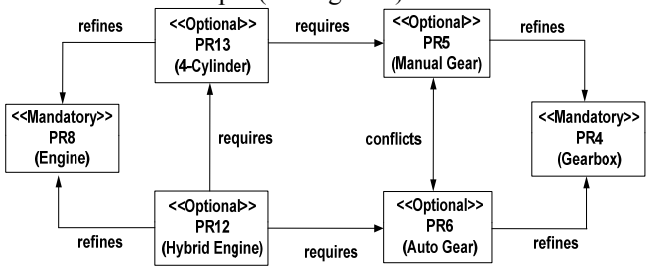

Figure 9 Inconsistent Relations in the Example Product-line Model

The consistency checking engine of TRIC reports that the relations (PR12 requires PR13), (PR12 requires PR6), (PR13 requires PR5) and (PR6 conflicts PR5) cause a contradiction. The relation (PR12 requires PR5) is inferred via (PR12 requires PR13) and (PR13 requires PR5). Therefore, a hybrid engine requires both auto gear and manual gear which are conflicting with each other. According to the semantics of the relations, it is not possible to require different properties that are conflicting with each other. We need to reconsider the relations causing the contradiction in order to have a consistent model. One possible fix is that a 4-Cylinder engine just requires a gear. We add the relation (PR13 requires PR4) and remove the relation (PR13 requires PR5).

In product-line reasoning a usual check is if the feature diagram is consistent, that is it has at least one valid configuration. This is possible with a different mapping to OWL. This also allows detecting dead features. However, we have not shown it in the paper because we use the current reasoning facilities in TRIC.

V. Reasoning on SysML Models

The System Modeling Language (SysML) [25] 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. In the SysML specification [31] the semantics of the SysML constructs (requirement and relation types) is given textually. As a result some constructs may be interpreted in multiple ways and generally have unclear semantics. In order to provide formal semantics we map the SysML constructs to the core metamodel elements (see Figure 10).

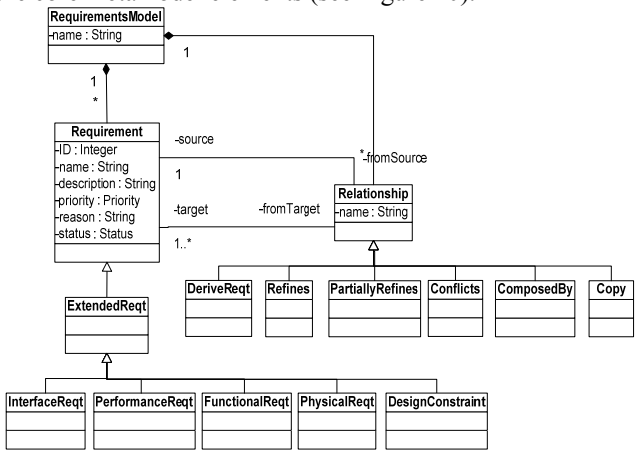

Figure 10 Specialized Requirements Metamodel for SysML Requirements

There are three types of requirements given as an extension of the *ExtendedReqt* entity in the SysML metamodel: *InterfaceReqt*, *PerformanceReqt* and *DesignConstraint*. These entities are mapped to an extension of the *Requirement* entity in the core metamodel (see Figure 10). Due to the ambiguous definitions of the SysML requirements relations (*DeriveReqt*, *Refines*, *ComposedBy*, and *Copy*), there are multiple interpretations of how to map the SysML relations to formally well-defined relations in the core metamodel (see Table I for our mapping).

TABLE I. Relation Types in the Core and SysML Metamodels

Relation Types in the Core Requirements Metamodel	Relation Types in the SysML Metamodel
Requires	DeriveReqt
Refines	Refines
Partially Refines	-
Contains	ComposedBy
Conflicts	-
Equals	Copy

The mapping is based on our own interpretation of the textual definitions of the SysML relations. *DeriveReqt*, *Refines*, *ComposedBy* and *Copy* are directly mapped to *Requires*, *Refines*, *Contains* and *Equals* in the core metamodel respectively. For *partially refines* and *conflicts* in the core metamodel there is no corresponding relation in the SysML metamodel.

In the SysML specification [31] *deriveReqt* has an ambiguous definition which is given as a dependency between two requirements in which a client requirement can be derived from the supplier requirement. Therefore, the mapping from *deriveReqt* to *requires* seems the most general one and we take it as a default mapping. For *deriveRqt* and

composedBy we consider that if 'A derivedRqt (or composedBy) B', then in TRIC it is encoded as 'B requires (contains) A'.

As an example we use the requirements of the Rain Sensing Wiper (RSW) system which is already modeled with SysML by Balmelli [21]. The goal of the RSW is to wipe the surface of the windshield automatically (i.e. without user intervention) whenever droplets of liquid are detected. The example model can be found in [30].

We had three iterations over the RSW requirements model. In the first iteration, we input the SysML model as given by Balmelli (see Figure 11) to TRIC based on Table I. For instance, the relations (SR16 deriveReqt SR13), (SR10 composedBy SR9) and (SR6 refines SR8) are mapped to the relations (SR13 requires SR16), (SR9 contains SR10) and (SR6 refines SR8) in TRIC respectively.

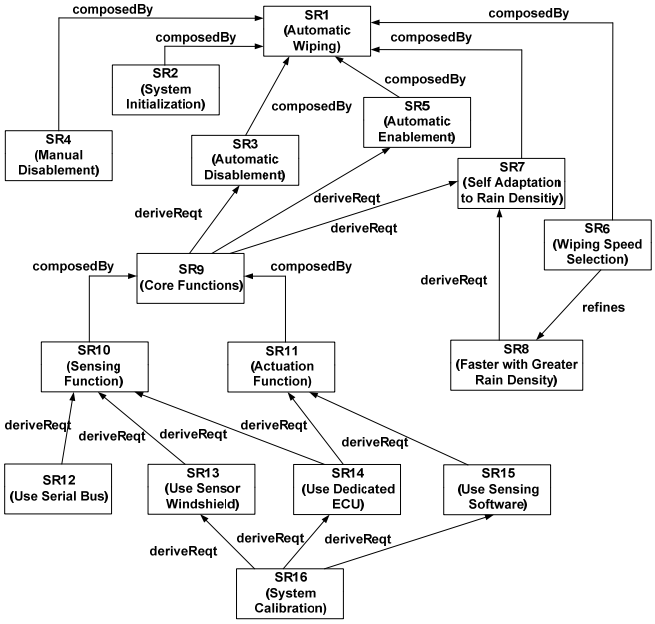

Figure 11 SysML Model for the Rain Sensing Wiper System [21]

The reasoner in TRIC inferred 46 relations from 21 given relations and no inconsistent relation was found. Since *deriveReqt* can be interpreted in different ways, we looked for a more specific interpretation in this concrete example. For instance, in some cases *deriveReqt* is mapped to *requires*; in others it is *refines*. In the second iteration we altered the model by updating some *deriveReqt* relations. Figure 12 shows the changed relations in the SysML model.

Two *deriveReqt* relations between SR8&SR7 and between SR13&SR10 are replaced by the *refines* relations since SR8 and SR13 are adding more details to the system properties given in SR10 and SR7. Other *deriveReqt* relations for SR10, SR13, SR14, SR15, SR16 are replaced by the *requires* relations in the same direction although the default mapping changes the direction of the relation when *deriveReqt* is replaced by *requires*. While the relation (SR10 requires SR15) is added to the model, the relation (SR14 deriveReqt SR10) is removed from the model. For other *deriveReqt* relations in Figure 11 we keep the default mapping.

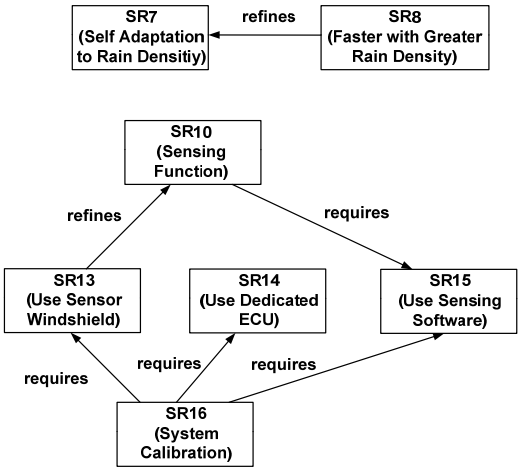

Figure 12 Changed Relations in the SysML Requirements Model

In the second iteration the TRIC reasoner inferred 55 relations from 22 given relations in the updated model and one inconsistency is detected. Figure 13 gives the inconsistent part in the SysML model.

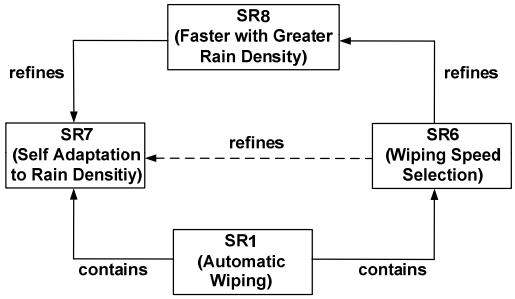

Figure 13 Inconsistent Part in the SysML Model

The TRIC reasoner reports that the *refines* relation between SR7&SR6 and the *contains* relations for SR1, SR6 and SR7 cause a contradiction. Please note that the *contains* relations are derived from the *composedBy* relations for SR1, SR6 and SR7 in Figure 11. According to the formal semantics the contained pairs cannot refine each other. As a result of our re-analysis of the relations, we concluded that the *contains* (composedBy) relations are valid. The relation (SR6 refines SR7) is a relation inferred via the given relations (SR6 refines SR8) and (SR8 refines SR7). Our conclusion for the given *refines* relations is that the selection of the wiping speed has no relation with the wiper adjustment based on the rain density. Although (SR6 refines SR8) is a relation given in [21], it is invalid. Therefore, it is removed from the model. In the third iteration 45 relations are inferred from 45 given relations and no inconsistency is detected.

VI. Reasoning on SysML and Product-Line Models

The mapping to the core metamodel provides a common semantic domain for multiple languages. This allows composing models expressed in different languages via the relations in the core metamodel and reasoning on the composed models. The requirements engineer should investigate two models and assign relations between models based on the relation types in the core metamodel. Since the

models might be developed by different development teams, there may be a need to have a negotiation between the teams about relations between models. After composing the models by assigning relations, the TRIC reasoner can be run over the composed models to infer new relations and check the consistency of given and inferred relations. Figure 14 gives the part of the composed models for the automotive example with only the given relations.

The mandatory and optional wiping requirements are related to the wiping system requirements given in the SysML model. The relations are not only from the product-line requirements to the SysML requirements but also from the SysML requirements to the product-line requirements. In Figure 14, three optional requirements PR26, PR28 and PR29 require some of the wiping system requirements. The selection of the optional requirements includes some SysML requirements in the product configuration. For instance, if we select only 'Front Interval Wiper' (PR28) by excluding 'Rear Wiper' (PR26) and 'Front Rain Sensor Wiper' (PR29), only 'Manual Disablement' (SR4) will be included in the product configuration.

In the composed models it is possible to infer some relations that are not inferred previously. Figure 15 shows an inferred relation between two product-line requirements in the composed models. The relations (PR26 requires SR5) and (SR5 requires PR30) are given where the relation (PR26 requires PR30) is inferred in the composed models. The inferred relation is not previously inferred in the product-line model (see Section IV) because the given relations used in the inferencing are between product-line and SysML requirements.

Table II gives the number of given and inferred relations, in the product-line (Section IV), SysML (Section V) and composed models. In all models TRIC mainly inferred some *requires* relations. From the formalization of relations, we know that the *contains* and *refines* relations imply the *requires* relation. The number of inferred relations in the composed models is more than the total number of inferred relations in the product-line model and the SysML model (the third iteration in Section V) because of the inferred relations like the one in Figure 15.

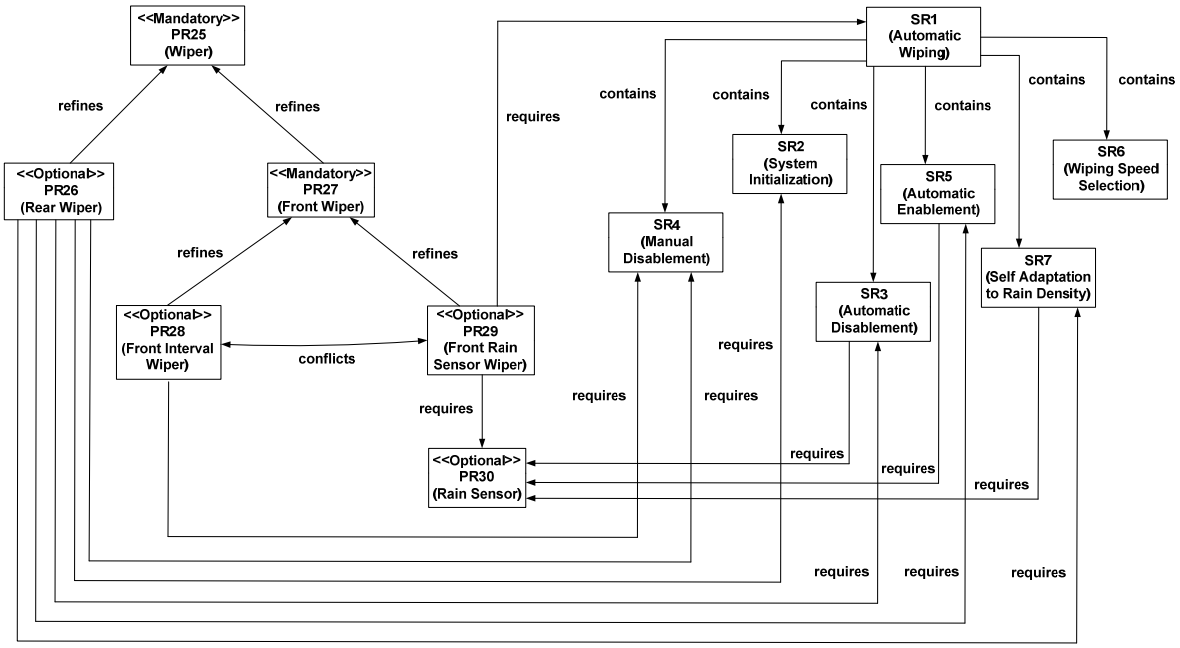

Figure 14 Part of the Composed Models

TABLE II. NUMBER OF GIVEN AND INFERRED RELATIONS IN THE AUTOMOTIVE EXAMPLE

		Number of Relations per Relation Type					
		Refines	***Partially Refines***	***Requires***	***Contains***	***Conflicts***	***Total***
Reasoning on the Product Line Model	**Given**	22	1	14	2	16	55
	Inferred	2	0	37	0	7	46
Reasoning on the SysML Model (First Iteration)	**Given**	1	0	12	8	0	21
	Inferred	0	0	46	0	0	46
Reasoning on the SysML Model (Second Iteration)	**Given**	3	0	11	8	0	22
	Inferred	1	0	54	0	0	55
Reasoning on the SysML Model (Third Iteration)	**Given**	2	0	11	8	0	21
	Inferred	0	0	45	0	0	45
Reasoning on the Composed Model	**Given**	24	1	35	10	16	86
	Inferred	2	0	132	0	1	135

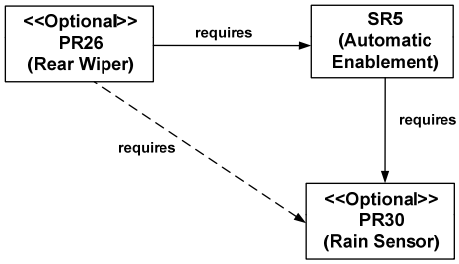

Figure 15 Inferred relation in the Composed Models

We do not address the consistency of the composed product-line and SysML models in the sense of checking the consistency of product-lines as described by Lauenroth and Pohl [19] [20].

VII. Related Work

Composing Requirements Specifications in Natural Text: A number of approaches address composing text based requirements specifications. Brottier et al. [14] introduce a requirements metamodel and present how they use it on top of a constrained natural language for requirements definitions. The authors proposed a model-driven mechanism to merge different requirement specifications and reveal inconsistencies between them by using their metamodel in [16]. The requirements metamodel is mainly used to produce a requirements model from a given requirements document. The requirements relations in the metamodel are not typed and they lack semantics. Reasoning about relations is not supported.

Mapping Requirements Models to Formal Models: Laleau et al. [22] present an extension of SysML with concepts from the KAOS method. Some rules are provided to derive a formal B specification from this extension. This gives a precise semantics to some SysML elements. However, it is not discussed what kind of reasoning can be achieved with the formal semantics based on a formal B specification. Another mapping to formal models is an argumentative approach [13] towards handling inconsistent requirement specifications. Requirements specifications from different sources are represented in terms of their interactions and mutual implications rather than internal semantics. A distinction between correctness and desirability is provided to handle and resolve inconsistencies among multiple requirements specifications. The notion of inconsistency is used for inconsistent requirements, not for inconsistent relations. The argumentation framework shows that two requirements are inconsistent without going into the details of their inconsistency. Mirbel et al. [18] uses the meta-argumentation theory to detect consistent sets of goal-based requirements and maintain their consistency over time. While Bagheri and Ensan [13] concentrate only on the *conflicts* relation, in [18] all the relations required to organize goals (AND/OR-decomposition, conflict, require and equivalence dependencies) are taken into consideration. These two approaches using argumentation framework do not support reasoning about requirements relations by combining multiple requirements models in different modeling languages. Giorgini et al. [27] propose a formal framework for reasoning with goal models. A precise semantics is given for all goal relationships in a qualitative and numerical form. The presented reasoning framework is very specific to goal models. Neither a reasoning facility nor a tool support is introduced. Zowghi et al. [11] [12] propose a logical framework for modeling and reasoning about the evolution of requirements. They characterize the properties correctness, completeness, and consistency of requirements in an evolutionary framework. The interaction of consistency and completeness with correctness during requirements evolution is discussed. Duffy et al. [10] propose a logic-based framework for reasoning about requirements specifications based on goal-tree structures. The framework is based on goal decomposition supported by an automated reasoning.

Specializing Requirements Metamodels: Navarro et al. [15] propose a metamodel customization approach for their requirements metamodel. They propose a core requirements metamodel which is generic and considers only *Artifact* and *Dependency* as core entities. Their metamodel does not support any of the requirements relations in our approach. Boukhari et al. [17] introduce a pivot model to provide interoperability between different requirements models such as UML Use Case and Goal-Oriented models. The existence of a shared global requirements metamodel is assumed for the interoperability. The requirements engineers are supposed to reference the global metamodel to have his/her local metamodel for the requirements modeling notation that he/she is using. The local metamodels can be considered as a specialization of the global metamodel. The main idea of the approach is very similar to our approach but no reasoning is supported for requirements relations either for local models or for the global model in [17].

Lopez et al. [26] 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. The integration aims at providing a structure for representing requirements and requirements relations but only informal definitions of the relations are provided.

Handling Multiple Views in Requirements Specifications: Finkelstein et al. [1] [8] describe a technique for inconsistency handling in requirements documents developed using multiple methods and notations for the same system. They combine the ViewPoints framework and a logic-based approach. Partial specification knowledge in each ViewPoint is translated into first-order logic. Logical inconsistencies are identified. Then, some temporal logic rules are combined with the identified inconsistencies to specify inconsistency handling actions. Hunter et al. [5] present an adaptation of classical logic, which they term quasi-classical (QC) logic that allows reasoning in the presence of inconsistency. This facilitates an analysis of inconsistent information. In our approach, inconsistencies are explained based on the derivation of relations. Sabetzadeh and Easterbrook [24] present a framework for merging

multiple views that tolerates inconsistency between the views. They demonstrate the application of the framework to the goal models and to entity-relationship models. Mainly all the approaches for handling multiple views do not target multiple models in different languages.

VIII. CONCLUSION

In this paper we presented a metamodeling approach which allows reasoning about requirements and their relations on the whole/composed models expressed in different requirements modeling notations. We used a core requirements metamodel that can be specialized for requirements modeling approaches. Mainly, the requirements relations in the metamodel were specialized to support the relations in multiple modeling approaches. Our example showed the feasibility of our approach for commonly used languages such as product-line models and SysML. The specialization allowed using the same semantics and the reasoning mechanism of the core metamodel for multiple languages whose constructs can be expressed in terms of the core metamodel. The language constructs with unclear semantics were mapped to the well-defined elements in the core metamodel. With the formal semantics and reasoning support we managed to detect some false positive *deriveReqt* and *refines* relations in the industrial SysML example.

As a future work, we plan to use our approach for other requirements engineering approaches like goal-oriented requirements models. For our example we encoded the product-line and SysML models in TRIC manually. We also plan to automate the encoding of product-line and SysML requirements and their relations in TRIC via model transformation techniques as a future work.

REFERENCES

[1] A. C. W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, B. Nuseibeh, "Inconsistency Handling in Multiperspective Specifications," IEEE Transactions on Software Engineering, 20(8), pp. 569-578, 1994.

[2] A. Goknil, "Traceability of Requirements and Software Architecture for Change Management," PhD Thesis, University of Twente, Enschede, 2011.

[3] A. Goknil, I. Kurtev, K. van den Berg, & J. W. Veldhuis, "Semantics of Trace Relations in Requirements Models for Consistency Checking and Inferencing," Software and System Modeling, *10*(1), 31-54, 2011.

[4] A. Goknil, I. Kurtev, K. van den Berg, "A Metamodeling Approach for Reasoning about Requirements," ECMDA-FA, LNCS, vol 5095, pp. 310-325, 2008.

[5] A. Hunter, B. Nuseibeh, "Managing Inconsistent Specifications: Reasoning, Analysis, and Action," ACM Transactions on Software Engineering and Methodology (TOSEM), 7(4), pp. 335-367, 1998.

[6] SWEBOOK. Guide to Software Engineering Body of Knowledge. IEEE Computer Society.

[7] A. van Lamswerdee, "Goal-oriented Requirements Engineering: a Roundtrip from Research to Practice," Invited Minitutorial, Proceedings RE'01, 249-263.

[8] B. Nuseibeh, J. Kramer, A. Finkelstein, "A Framework for Expressing the Relationships between Multiple Views in Requirements Specification," IEEE TSE, 20(10), 760-773, 1994.

[9] D. Batory, "Feature Models, Grammars, and Propositional Formulas", SPLC, S. Obbink, J. H. & Pohl, K. (Eds.), Springer, 2005, 3714, 7-20

[10] D. Duffy, C. MacNish, J. McDermid, P. Morris, "A Framework for Requirements Analysis using Automated Reasoning," CAiSE'95, Lecture Notes in Computer Science, 932, pp. 68-81, 1995.

[11] D. Zowghi, V. Gervasi, "On the Interplay between Consistency, Completeness and Correctness in Requirements Evolution," Information and Software Technology, 45, 993-1009, 2003.

[12] D. Zowghi, R. Offen, "A Logical Framework for Modeling and Reasoning about the Evolution of Requirements," RE'97, 247-257, 1997.

[13] E. Bagheri, F. Ensan, "Consolidating Multiple Requirement Specifications through Argumentation," Proceedings of the 2011 ACM Symposium on Applied Computing, pp. 659-666, 2011.

[14] E. Brottier, B. Baudry, Y. Le Traon, D. Touzet, B. Nicolas, "Producing a Global Requirements Model from Multiple Requirement Specifications," EDOC 07, pp. 390-404, 2007.

[15] E. Navarro, J. A. Mocholi, P. Letelier, I. Ramos, "A Metamodeling Approach for Requirements Specification," The Journal of Computer Information Systems, 46(5), 67-77, 2006.

[16] G. Perrouin, E. Brottier, B. Baudry, Y. Le Traon, "Composing Models for Detecting Inconsistencies : A Requirements Engineering Perspective," REFSQ 09, pp. 89-103, 2009.

[17] I. Boukhari, L. Bellatreche, S. Jean, "An Ontological Pivot Model to Interoperate Heterogeneous User Requirements," ISoLA 2012, LNCS 7610, pp. 344-358, 2012.

[18] I. Mirbel, S. Villata, "Enhancing Goal-Based Requirements Consistency: An Argumentation-Based Approach," 13th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA 2012), LNCS Springer, Montpellier, France, August, 2012.

[19] K. Lauenroth, K. Pohl, "Towards Automated Consistency Checks of Product Line Requirements Specifications," ASE'07, Atlanta, 2007.

[20] K. Lauenroth, K. Pohl, "Dynamic Consistency Checking of Domain Requirements in Product Line Engineering," RE'08, 2008.

[21] L. Balmelli, "An Overview of the Systems Modeling Language for Products and Systems Development," Journal of Object Technology, 6(6), 149-177, 2007.

[22] R. Laleau, F. Semmak, A. Matoussi, D. Petit, A. Hammad, B. Tatibouet, "A First Attempt to Combine SysML Requirements Diagrams and B," Innovations Syst Softw Eng, 6:47-54, 2010.

[23] 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), 551-569, 2005.

[24] M. Sabetzadeh, S. Easterbrook, "View Merging in the Presence of Incompleteness and Inconsistency," Requir. Eng. 11(3): 174-193, 2006

[25] M.S. Soares, J. Vrancken, "Model-driven User Requirements Specification using SysML," Journal of Software, 3 (6), 57-68, 2008.

[26] O. Lopez, M. A. Laguna, F. J. Garcia, "Metamodeling for Requirements Reuse," WER02-Workshop, pp. 76-90, 2002.

[27] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, R. Sebastini, "Formal Reasoning Techniques for Goal Models," Journal on Data Semantics, LNCS, 2800, pp. 1-20, 2003.

[28] J. W. Veldhuis, "Tool support for a metamodeling approach for reasoning about requirements," MSc Thesis, University of Twente, Enschede, 2009.

[29] Tool for Requirements Inferencing and Consistency Checking (TRIC) from http://trese.cs.utwente.nl/tric/

[30] http://www-sop.inria.fr/members/Arda.Goknil/automotive_example/

[31] OMG SysML Specification from http://www.sysml.org/docs/specs