An Ontology Based Approach for Requirement Inconsistency Detection

Farheen Siddiqui*
M Afshar Alam*

Abstract

This paper describes the approach to use ontologies developed for use in Semantic Web technology in checking the consistency of requirements specifications. Our focus area is use of reasoning which the distinguishing feature of OWL that separates it from other knowledge representation class of languages a part of ontology. In the first step, we transform the classes and properties (static) of UML model and its constraints into problem ontology and point out contradictions using ontology reasoning. The contradictions that have been found indicate inconsistencies. In the second step, we try to find mismatch between the problem ontology coming from the UML model that represents the requirements, and the domain ontology, that represents the domain knowledge available on web. However, a limitation of this approach is that only the static part of model is checked and the dynamic part cannot be represented in ontology.

Keywords: Consistency, Checking, Ontology, Reasoning

1. Introduction

Requirements specification is a is a complicated process and usually takes time longer than expected. The goal is to describe exactly what the user wants and needs. Any failure and mistake in requirements specification is very expensive because it results in the development of software parts that do not fit to the real needs of the user and must be reworked later causing additional costs. When the analysis phase of a project starts, analysts have to discuss the problem to be solved with the customer (users, domain experts). Then, they will note the requirements they have found in form of a textual description. This is a form the customer can understand. However, any textual description of requirements can be (and usually is) incorrect, incomplete, ambiguous, and inconsistent. Later, the analyst specifies a UML model (Unified Modeling Language) [21] based on the requirements description he has written himself. However, users and domain experts cannot validate the UML model as most of them do not understand (semi-)formal languages such as UML. Misunderstandings between analysts and users are very common and bring projects over budget.

Requirements are based on the knowledge of domain experts and the users' needs and wishes. One possible way to represent and classify this knowledge and then fashion it into a tool is

*Department of Computer Science, Hamdard University, New Delhi, India.

through ontology engineering. Simplified, ontologies are structured vocabularies (basic concepts in domain and the relationships among them) and the possibility of reasoning. Domain models play a central role throughout the software development cycle, from requirements analysis to design, through implementation and beyond. As such, great progress has been made in the consistent use of models throughout this process. Modern software development tools with support for the UML and code generation as well as Model-Driven Architectures allow for developers to synchronize and verify technical implementation with user requirements using models. Our paper focuses on the possibility of improvement requirements specifications by incorporating domain ontologies. We will introduce our concept that uses ontology reasoning; present the implementation and discuss the advantages in the conclusion.

Ontology technologies are frequently applied to many problem domains nowadays. Because concepts, relationships and their categorizations in a real world can be represented with ontologies, they can be used as resources of domain knowledge. The rest of the paper is organized as follows. In Section 2, we discuss related work. In Section 3, we briefly explain why ontology is the proper mechanism to be used in requirements specification. Section 4 describes the architecture and functionality of our consistency checking model. In Section 5, the implementation of the component that represents our approaches is given. In Section 6, we present evaluation of our approach, Section 7 discusses limitations and the Section 8 contains conclusions.

2. Related Work

Using ontologies in the requirements engineering process is clearly not a new idea. An ontology-based approach to knowledge acquisition is discussed in [16], in [18], and in [3]. In [22], they have constructed the Enterprise Ontology to assist developers in having an enterprise-wide view of an organisation. Such ontology could be used as our predefined domain ontology. The approach in [16] is intended to automate both, interactions with users and the development of application models. The ontologies used by [29] in their Oz system are domain models which prescribe detailed hierarchies of domain objects and the relationships between them. Formal models for ontology in requirements are described in different papers, e.g., in [15]. In [24], the UML model is enhanced with a formal semantics and used for working with a use-case model and class diagrams. In [25], an ontology model for requirements elicitation for domain knowledge and domain rules checking is presented. In [23], the inconsistency measurement is discussed. The ontology used by the QARCC system [1] is decomposition taxonomy of software system quality attributes. In [25], a formal model of requirements elicitation that contains domain ontology checking is discussed.

The concept used for consistency checking in the 1990s based on the relationships between viewpoints and their problematic interferences [4]. An overview about inconsistencies is given in [5], in [26], and lately in [30]. However, there is not an approach applying ontology in the sense of our way. Another approach is a concept based on the semantics of dependency as described by the UML specification. It will be used to describe the behavioral part of the UML model [19]. Other research checking behavioral consistency is [20] or [9]. They are working with the dynamic model.

3. Need for using Ontologies in Requirement Consistency Checking

According to [23], the most popular definitions of "ontology" are "formal explicit specification of shared conceptualization" and "classifications of the existing concepts". In this paper, domain ontology provides semantic basis on requirements to be elicited. More concretely, each statement representing a requirement (we call it requirements statement) can be interpreted with ontological elements and such ontology should include atomic concepts that any stakeholders can commonly have in a problem domain. OWL language element used to express such definitions is called restriction. A restriction describes the class of all instances that fulfill a specific condition on a property. There are various types of restrictions in OWL. For example, hasValue restriction which links a property to a specific individual. Other restrictions limit the cardinality of a property, for example to define the class of all things that have at least two values for a specific property.

The key power of OWL is that classes can be defined by combining multiple restrictions and other classes. For that purpose, OWL provides logical operands to build intersections (and), unions (or) and complements (not) of other classes. In object-oriented systems, such statements would typically have been hidden somewhere inside the code base itself. In Semantic Web ontologies, the logical relationships are made explicit through OWL class definitions and other formal statements. Semantic Web applications usually exploit other tools to handle and analyze OWL models. One family of such tools is called reasoners. A reasoner is a service that takes the statements encoded (asserted) in an ontology as input and derives (infers) new statements from them. In particular, OWL reasoners can be used to:

- Reveal subclass/superclass relationships among classes
- Determine the most specific types of individuals
- Detect inconsistent class definitions

Reasoning in ontology adds the inferential capabilities that are not present in taxonomies which have been used for modeling so far. It makes possible to search for contradictions that indicate inconsistencies. The UML model includes a constraint part but it has no mechanism for indicating contradictions in constraints. When many UML models are considered, the resulting UML model has no mechanism for indicating a discrepancy in the class hierarchy. When we transform the requirements specifications into a UML model (using TESSI in our case), we either have to believe that the analyst understands the user's needs perfectly and is perfect in UML model building or we do not believe it and to try to verify the UML model. Ontology engineering offers a mechanism to discover contradictions in constraints and in class hierarchies. To use it, it is necessary to transform the UML model into an ontology which we call problem ontology.

In a specialized software house, developing for example information systems for Education, domain ontology can be constructed that represents the domain knowledge. Such

domain ontology can be merged with the problem ontology of the current project and the resulting ontology can be tested for contradictions as described above. The contradictions that have been found indicate inconsistencies in the requirements specifications. The fact that it is not possible to test the dynamic part of the UML model (the behavior part) is a disadvantage of this approach. However, the methods of UML behavioral consistency checking (in [20]) depend on formal semantics methods and they are very complex.

Requirements are based on knowledge of domain experts and users' needs and wishes. One possible way to classify this knowledge and then fashion it into a tool is through ontology engineering. This way will be discussed in this report. Ontologies can be seen as explicit formal specifications of the terms in the domain and relationships among them. They care for a shared understanding of some domain of interest. Such an understanding can serve as the basis for communication in requirements development.

Ontologies are a guarantee of consistency and enable reasoning. Ontology-based requirements specification tool may help to reduce misunderstanding, missed information, and help to overcome some of the barriers that make successful acquisition of requirements so difficult.

Simplified, ontologies are structured vocabularies having the possibility of reasoning. It includes definitions of basic concepts in the domain and relations among them. It is important that the definitions are machine-interpretable and can be processed by algorithms.

Why would someone want to develop an ontology?
Some of the reasons are:

- To share common understanding of the structure of information among people or software agents.
- To enable reuse of domain knowledge.
- To make domain assumptions explicit.
- To separate domain knowledge from the operational knowledge.

Ontology research has primarily focused on the act of engineering ontologies or it has been explored for use in domains other than requirements elicitation, specification, checking, and validation.

For an ontology being successfully used in requirements checking, it has to have the following properties: completeness, correctness, consistency, and unambiguity.

The intuitive meaning is:

- correctness means that the knowledge in the ontology does not violate the domain rules that correctly represent the reality,
- consistency means that there are no contradictory definitions in ontology,
- completeness means that the knowledge in ontology describes all aspects of the domain,
- unambiguity means that the ontology has defined a unique or unambiguous terminology.

There are not obscure definitions of ontology concepts, i.e. each entity is denoted by only one, unique name, all names are clearly defined and have the same meaning for the analyst and all stakeholders. Correctness and consistency are logical properties that can be checked by some reasoning mechanism provided that this mechanism works correctly.

This is what we can use for improving the quality of requirements. After we have checked the correctness and consistency of the corresponding domain ontology (domain knowlegde), we can check whether the requirements (transformed into an ontology) are correct and consistent mutually, and correct and consistent to the given domain ontology. Completeness is a complex concept. We know about other kinds of completeness that are related to computing. For example, logical completeness is a property associated with combining a procedure for constructing well-formed formulas, a definition of truth that relates to interpretations and models of logical systems, and a proof procedure that allows new well-formed formulas to be derived from old well-formed formulas. A logical system is logically complete if every true well-formed formula can be derived. Consistency is the other side to logical completeness. In inconsistent systems falsity can be derived because of an contradiction and any well-formed formula can be derived as true. This is sure not what we want. A second kind of completeness is computational completeness, which is a property of a programming language. In our paper we are focusing on the consistency property of requirement.

4. Approch for Requirment Checking

In this section, we will discuss our approach for consistency checking and propose consistency checking model. There is a gap between the requirements definition in a natural language and the requirements specification in some semi-formal, graphical representation. The analyst's and the user's understanding of the problem are usually more or less different when the project starts. The first possible point of time when the user can validate the analyst's understanding of the problem is when a prototype is used and tested. We developed a method that check for the possible instances of domain rule and restriction violation and then produce a refined consistent set of requirement.

Working with it, the analyst uses the grammatical inspection to specify the roles of words in the text in the sense of object-oriented analysis. During this process—shown in Fig. 1—a UML model will be built driven by the analyst's decisions.

To make the UML model obtained from analyst more accurate, we have incorporated the semantic web technology in consistency checking process. A model of our concept is shown in Fig. 2. Using the experiences given in [25], we describe the domain ontology in Prot´eg´e [28] and apply ontology reasoning (e.g., the inference engine in HermiT [27] – for checking classes) first for domain ontology checking, then for requirements problem ontology checking, and at last for checking whether the requirements problem ontology subsumes the domain ontology.

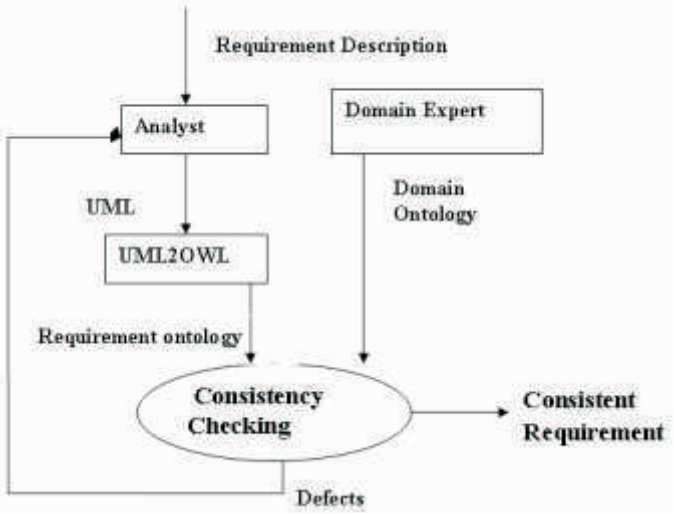

Figure 1: Flow diagram for consistency checking

The steps of the requirements processing using ontologies are the following :

- Build domain ontology using Prot´eg´e,
- check the domain ontology for consistency using Pellet and Jess
- The analyst writes a text description of requirements based on interviews with users,
- the analyst builds the UML model from a textual description of requirements
- Convert requirements described as a UML model to a problem ontology using converter UML2OWL called as problem ontology,
- Check the problem ontology for its consistency, merging the problem ontology with the domain ontology and checking them for consistency identifying inconsistency problems,
- Find the corresponding parts in the former textual description of requirements and correcting them,
- building a new UML model based on corrected textual description of requirements
- Follow the above step till no inconsistency is found
- When no defects are found, use the refined requirement ontology for further step of software development.

5. Evaluation

For experiments, we used the requirements specification of a Education Provider that describes the functional requirements for a student management system.

A. Checking with rules

Let us consider this example for checking rules: There is a relation "SetPaper" between classes Person and Student. But if we model the class Student (additionally feature of enrollment in some Course) as a subset of the class Person, the student (as an instance) could set paper for himself. This is usually not what we want. Usually, we do not allow that a clerk in a bank can give a loan to himself, we do not want a manager decide about his salary, etc. The solution is that we do not allow some relations to be reflexive in the domain ontology, e.g., the relation "SetPaper". Any problem ontology that does not contain the condition that a Student must not set a paper to himself will be found to be inconsistent to the domain ontology.

This example can be checked in TESSI by modeling the two classes Person and Student. We decide that a Student is a specialization of a Person with additional possibilities to Enroll in a course. Then we define an association between these two and name the direction from Student to Person "SetPaper". After that, we can use SWRL-Rules to describe the desired behavior. The first rule we need will set the relation between every possible Student and Person pair:

STUDENT(?X) ^ PERSON(?Y) →SETPAPER(?X, ?Y)

The second rule will be used to check if any of the students is able to set paper for himself:

SETPAPER(?X, ?Y)) ^ SAMEAS(?X, ?Y) ERROR(?X, "SELF")

Now, we can create a UML model based on our example and then generate ontology with this content and load it into Prot´eg´e. It will set up the "SetPaper" relationship for all Person and Students and then test for Students that set paper for themselves. The Inferred Axioms window in Prot´eg´e will then list all possible errors so we can use this information to make corrections to the UML model. In this case, we can remove the subclass from User. During a further test, Jess will get no errors.

B. Checking with restrictions

The next example will show the possibility to check restrictions. In our education system a student can appear in exam only if he does not having more than five backlog paper that are pending. So, the "AppearIn" relationship between student and exam should be restricted. In our modeling tool we set Student and Exam as association ends. The direction from student to Exam will be labeled with ApperIn and has cardinality 0 to n, in this example n is set to 5. Both classes, Student and Exam must have set some equivalents in the domain ontology to access the corresponding individuals later. To provide some test data we need to add a constraint that fills the AppearIn relation:

Student(?x) ^ Exam(?y) AppearIn(?x, ?y)

After converting the model to an ontology we use the SWRLTab to infer the new axioms and then use Jess to include the new knowledge into our ontology. Afterwards we can check the results on the individuals tab in Prot´eg´e. It will indicate that the defined restrictions are not fulfilled. Based on these observations either the restrictions must be corrected or the test data is wrong and the constraint for filling it must be adopted.

6. Limitations

One of the problems that occur is that the constraints of requirements are described in OCL (Object Constraint Language) which is stronger than SWRL (Semantic Web Rule Language). The reason is that the computational complexity of the reasoning, i.e., of the decision whether the system is correct and consistent, may explode and we never obtain the result guaranteed if the expressiveness of the used description logic is too high. Hence, we can use only the subset of OCL that

corresponds to SWRL. SWRL also offers only limited possibilities to express rules. The formulas are based on first order logic but can only contain conjunctions of atomic formulas. There is no support for quantifiers or complex terms. SWRL also can't express negations, which requires the user to create formulas on a special way and limits the expressiveness of SWRL rules.

Another problem is the necessity to use individuals to process SWRL rules. This requires adding several individuals of every class to the domain ontology without knowing what rules will later be modeled in TESSI. It also requires having meaningful properties set to these objects. Otherwise it will not be possible to validate the model with SWRL rules. On the other side, there are hundreds of constraints in most UML models which we can describe using SWRL, e.g., value ranges of attributes and other not very complicated conditions. If we cannot describe a constraint in SWRL, then the only consequence is that our prototype cannot use it for checking.

7. Conclusions

We have shown that using ontologies supports consistency checking which is critical to the process of requirements engineering. Reasoning as a part of ontology discovers contradictions in the class hierarchies and the constraints of UML models and thereby helps in indicating inconsistencies. In the introduction of this paper we stated the motivated questions. Now, we can conclude

- It possible to improve requirements specifications by using ontologies. We presented a concept and a tool that finds contradictions and inconsistencies in UML models.
- Using ontologies, we can improve only those parts of the UML model that can be transformed into ontologies. These are the structural part (static part of the model-class hierarchy) and the constraint part of the model. We cannot improve the behavioral part (the dynamic part of the model-state machine diagrams, sequence diagrams, etc.)
- Using our approach we can gain indications about contradictions in class hierarchy and in constraints in the problem ontology and indications about contradictions between the problem ontology and the domain ontology

8. References

1. *B. Boehm and H. In, "Identifying quality requirements conflicts," IEEE Software, pp. 25-35, March 1996.*
2. *CHAOS Report. The Standish Group, 1995. http://www.projectsmart.co.uk/docs/chaos-report.pdf*
3. *L.L. Christopherson, "Use of an ontology-based note-taking tool to improve communication between analysts and their clients," A Masters Paper for the M.S. in I.S.degree, University of North Carolina, November, 2005.*
4. *S. Easterbrook and B. Nuseibeh, "Using ViewPoints for Inconsistency Management," IEEE Software Engineering Journal, November 1995.*
5. *S. Easterbrook, J. Callahan, and V. Wiels, "V & V Through Inconsistency Tracking and Analysis," Proceedings of International Workshop on Software Specification and Design, Kyoto, 1998.*
6. *H. Eriksson, "Using JessTab to integrate Protege and Jess,"Intelligent Systems, Volume 18, Issue 2, pp. 43-50, IEEE Mar- Apr 2003.*
7. *K. Falkovych, "Ontology Extraction from UML Diagrams," Master's thesis, Vrije Universiteit Amsterdam, 2002.*
8. *D. Gasevic, D. Djurevic, and V. Devedzic, "Model Driven Architecture and Ontology Development," Springer, 2006.*
9. *C.L. Heitmeyer, R.D. Jeffords, and B.G. Labaw: "Automated COnsistency Checking of Requirements Specifications," ACM Transactions on Software Engineering and Methodology, Vol. 5, No. 3, pp. 231-261, July 1996.*
10. *G. Hillairet, "ATL Use Case - ODM Implementation (Bridging UML and OWL)," http://www.eclipse.org /m2m/atl/usecases/ODMImplementation.*
11. *M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe, "A Practical Guide to Building OWL Ontologies - Using the Protege-OWL Plugin and CO-ODE Tools," Edition 1.0., http://protege.stanford.edu/doc/users.html.*
12. *A. Hunter and B. Nuseibeh, "Managing Inconsistent Specifications: Reasoning, Analysis and Action," ACM Transactions on Software Engineering and Methodology, Vol. 7, No. 4, pp. 335-367, 1998.*
13. *R. Janetzko, "Applying ontology for checking of requirements specification," M.Sc. Thesis, Faculty of COmputer Science, TU Chemnitz, 2009. (In German)*
14. *E. Friedman-Hill, "Jess in Action - Rule-based Systems in Java," Manning, Greenwich, 2003.*
15. *D. Jiang, S. Zhang, and Y. Wang, "Towards a formalized ontology-based requirements model," Journal of Shanghai Jiaotong University (Science), 10(1), pp. 34-39, 2005.*
16. *Z. Jin, "Ontology-Based Requirements Elicitation," Journal of Computers, 23(5), pp. 486-492, 2003.*
17. *F. Jouault, F. Allilaire, J. Bezivin, and I. Kurtev, "ATL: A model transformation tool," Science of Computer Programming, Vol. 72, No. 1-2. pp. 31-39, June 2008.*
18. *H. Kaiya and M. Saeki, "Using domain ontology as Domain Knowledge for Requirements Elicitation," Proceedings of 14th IEEE International Requirements Engineering Conference, Minnesota, pp. 186-195, 2006.*
19. *S. Kremer-Davidson and Y. Shaham-Gafnil, "UML 2.0 Model Consistency - The Rule of Explicit and Implicit Usage Dependencies," UML 2004 Modeling Languages and Applications, 3rd Workshop on Consistency Problems in UML-based Software Development: Understanding and Usage of Dependency Relationships, Lisboa, 2004.*
20. *Y. Thierry-Mieg and L. Hilla, "UML behavioral consistency checking using instatiable Petri nets," Journal Innovations in Systems and Software Engineering, Springer, Vol. 4, No. 3, 2008.*
21. *http://www.omg.org/spec/UML/2.0/*
22 *M. Uschold, M. King, S. Moralee and Y. Zorgios: "The Enterprise Ontology," Knowledge Engineering Review, 13(1), pp. 31-89, 1998.*
23. *X. Zhu and J. Zhi, "Inconsistency Measurement of Software Requirements Specifications an Ontology-Based Approach," Proceedings of the 10th IEEE International Conference on engineering of Complex Computer Systems, 2005.*
24. *X. Li, Z. Liu, and J. He, "Consistency checking of UML requirements," Proceedings of the 10th IEEE International Conference on Engineering of COmplex Computer Systems ICECC, 2005.*
25. *Z. Li, Z. Wang, A. Zhang, and Y. Xu, "The Domain Ontology and Domain Rules Based Requirements Model Checking," International Journal of Software Engineering and Its Applications, Vol. 1, No. 1, July, 2007.*

26. *B. Nuseibeh, S. Easterbrook, and A. Russo, "Leveraging Inconsistency in Software Development," IEEE Computer, April 2000.*
27. *http://pellet.owldl.com*
28. *http://protege.stanford.edu/overview/index.html*
29. *W. Robinson and S. Fickas, "Supporting Multiple Perspective Requirements Engineering," Proceedings of the 1st International Conference on Requirements Engineering (ICRE 94), IEEE Computer Society Press, pp.206-215, 1994.*
30. *G. Spanoudakis and A. Zisman, "Inconsistency Management in Software Engineering: Survey and Open Research Issues," Handbook of Software Engineering and Knowledge Engineering, World Scientific Publishing Co., pp. 329-380, 2001.*