Conflict Analysis at Collaborative Development of Domain Specific Models using Description Logics

Christian Bartelt
University of Clausthal
cristian.bartelt@tu-clausthal.de

Abstract

Today the distribution of development locations, the co-evolution of models and the concurrency of work are typical for collaborative modeling in software projects. Software engineering teams demand modeling techniques at several abstraction levels to manage the complexity of software descriptions. Besides, software models are applied more and more for the specification of safety-critical systems. Hence software models take a hybrid role – as a matter of team communication and precise specification for refinement. Both aspects are considered in the research area of Model Driven Engineering (MDE). It provides methods to deal with formal specified meta-models of graphical (intuitive) modeling languages. Unfortunately the syntactical a semantically correct (consistent) integration of concurrently evolved models is poorly considered by the most MDE approaches. Especially the detection and analyzing of model merge conflicts can be automatized by using logical inference techniques. Therefore this paper proposes an approach based on description logics.

1. Introduction

Nowadays complex software systems are highly integrated in large infrastructures and pervade all areas of life. Besides many software-intensive systems have to be highly dependable because they are used in safety-critical environments (e.g. automotive- or medical technology). At the same time the demands on productivity of software engineering projects are increasing. This situation influences widely the software development processes. Large teams of software engineers have to use precise description techniques in all development phases from requirements engineering and design till coding. That ensures a high quality of software documents. On the one hand design or specification documents should be intuitively understandable as a matter of communication and on the other hand the description should be precisely formulated to enable a reliable (possibly automated) verification. Many approaches in the research area of Model Driven Engineering (MDE) are focused these mentioned challenges. MDE focuses on specification methods and techniques by models with a well-defined syntax and semantics. Thereby syntax and semantics of models are defined by meta-models of domain specific or general purpose modeling languages. Further these languages define a graphical (concrete) syntax. The usage of well-defined modeling languages is one precondition for an efficient collaborative modeling in creative design phases. But a further precondition is a sufficient computer-aided support of parallelized and geographically distributed team-work. Complex and highly integrated software systems are often developed in global software engineering projects [3]. Therefore distributed programming in the realization phase is well-supported by several version management tools. Popular tools as CVS [8] or collaboration development platforms as [11,20] enable the integration of parallel evolving text or source code. But similar to source code, models in MDE are subjected to a continuing evolution. In industrial practice a lot of versions, configurations, or variants are created as account for evolution. The management of distributed and parallel, evolving models is an ambitious challenge. Especially the preserving of syntactic and semantic correctness after an automatical merge of model versions is not sufficient supported by software tools. A pure three-way-merge of changes at atomic model artifacts was discussed in [4]. But a pure merge of two parallel evolved models is an incomplete integration because the correctness of merged models is not proven. Even if two parallel changed model versions are correct regarding syntax and semantics, the merged result is potentially incorrect. But a manual conflict resolving in large and complex model repositories is a very time-consuming and error-prune work. However this process can be supported by computer-aided conflict analysis and visualization. To address this kind of tool support [5] has sketched the expected behavior and visualization of conflict detection. Furthermore it has proposed an approach for a meta-model independent
syntactical analysis based on OCL. But the application range of OCL based conflict detection is restricted. OCL engines are only able to check static semantics of meta-model constraints (e.g. cyclic inheritance). But in case of serious syntax violation of a merged model version cannot be parsed so that an OCL checker cannot be applied. One example for these serious syntactical conflicts is the concurrent composition of model elements in EMF-models. It will be discussed in detail in Section 4.

The problem of correctness-preserving model integration in collaborative software engineering is well-known and discussed in [14, 21]. Many approaches focus on the correctness analysis of UML models. One example is the teamwork support in the popular CASE tool MagicDraw [16]. For consistency preserving, it provides a pessimistic locking mechanism. But pessimistic locking mechanisms are unsatisfactory for modeling large models because they restrict concurrent work too hard [7]. Some software prototypes which implement merge concepts for collaborative modeling are published [4, 12]. However they provide only an consistency management support for certain modeling languages like UML and is provided for any domain specific models.

Description logics (DL) [1] allow extensive analyzing of software models. For the first time the usage of DL to analyse UML-diagrams is proposed by [2]. The following sections illustrate a transfer of the approach in [2, 21] to design an automatic conflict analyzing of models independent of a certain meta-model.

2. Description Logics and their implementations

Description logics (DL) are a family of formal languages for knowledge representation [1]. They resemble a subset of first-order predicate logic. Especially DL provides a more comprehensive expressiveness than propositional logic but is also decidable. A description logic can be divided into two parts: One part (TBox) contains the concept knowledge of the application domain (terminological knowledge) and the second part (ABox) contains a base of valid facts (assertional knowledge). Description logics have well-defined semantics. There are a lot of DL-based systems for computer-aided reasoning: KL-One, LOOM, Fact, Racer, etc. Today, the web ontology language OWL-DL [17] is one of the most popular languages for knowledge representation.

2.1. OWL and conjunctive queries

The proposed approach in this paper uses the OWL2 DL language. It bases on the description logic SROIQ(D). There are several reasons for the design of OWL (2) DL: Maximum of expressiveness possible while retaining computational completeness, decidability, and the availability of practical reasoning algorithms.

The proposed approach utilizes some elementary advantages of OWL: OWL has a definite semantics description in contrast to several modeling languages as MOF or UML. There are a lot of efficient reasoners for correctness checking and analysis and there is an implemented draft of a query language which supports conjunctive queries [6].

2.1.1. Reasoning services for OWL resp. DL-Lite. There are several tools which provide reasoning for OWL ontologies. The most popular tools are Pellet [19], KAON2, and RacerPro. Noteworthy is the fact that all these tools provide interesting checks of consistency and satisfiability:

Consistency: An ontology is consistent iff it contains no contradictions. Two examples of inconsistent ontologies (DL-syntax) are:

\[2R \subseteq C \land \neg1R \subseteq C\],

\[\neg1R \subseteq C \land C(a) \land C(b) \land C(c) \land R(a,c) \land R(b,c)\]

Concept satisfiability: A concept \(C\) is unsatisfiable iff \(C \equiv \bot\). An example for an unsatisfiable concept is: \(\neg C \subseteq C'\).

These kinds of checks are the foundation for the inconsistency identification and analysis which are discussed in section 5. For the analysis of software models concept satisfiability checks in finite domains are mainly interesting [6].

2.1.2. Analysis of OWL-ontologies by conjunctive queries. Conjunctive queries are one research draft for analyzing OWL-ontologies. Tools like Pellet can process this kind of queries. One important use case for conjunctive queries is a retrieval for individuals with certain characteristics in a knowledge base. For example, the following query requests all components which inherits from itself:

\((Component(x) \land \text{inherit}(x,x))\)

If \(x\) is a distinguished variable then the answer of the query is \(a, b, c\) for the following knowledge base:

\((T \subseteq \text{inherit.Component} \land \exists \text{inherit}\ T \subseteq Component \land \text{Trans(inherit) Component(a) Component(b) Component(c)} \land \text{ inherit(b,a)} \land \text{ inherit(c,b) \land inherit(a,c) }\)
For this example, the query mechanisms enable the identification of all individuals which are involved in a cyclic inheritance relation. Conjunctive queries are the foundation of pre-defined or user-defined inconsistency.

2.2. Open vs. closed world assumption

The closed world assumption (CWA) is the dominant paradigm in the area of model driven software engineering. It is the presumption that each statement which is not deducible to be true, is false. But all popular OWL reasoners support only the open world assumption (OWA). The OWA is the presumption that statements are false if and only if they are deducible to be false. There are few ambitions to extent the mentioned reasoners by a CWA support [13]. The presented approach “emulates” the CWA semantics by explicitly limiting the universe to known individuals and by explicit restrictions for concept and role specifications.

3. Co-Evolution example scenario

In this section an example scenario for a collaborative and domain specific modeling process is explained. It enables a vivid depiction for research which will be described in Section 4 and 5. The scenario introduces a model of a domain specific language (DSL) and depicts two concurrent modeling branches. Further it presents an intuitive model merge result.

3.1. A DSL to support the co-evolution of requirements and SW-architecture

The exemplary DSL supports the design of models in early phases of the software engineering process – requirements engineering (RE) and architecture design. The meta-model-excerpt of the DSL (Figure 1) describes RE and architectural artifacts and several relations between them. The meta-model is based on the ECore-language [9]. Especially RE and architecture design are interlaced and require collaborative and pronounced creative modeling activities. Usually a (possibly distributed) team of several software engineers work in parallel on requirements and architecture.

Figure 1. Excerpt of the ReqArch meta-model
3.2. Concurrent collaborative modeling

A simplified concurrent engineering process is exemplarily illustrated in Figure 2. The figure shows two software engineers Alice and Bob who adapt independently a origin document version VO and create two revisions VA and VB. With the help of a SCM-software, VA and VB can be merged.

One approach to decompose a model in fine-grained artifacts for the application of such merge mechanism is described in [4]. But the merge result VM is potentially inconsistent.

3.2.1. Preserving model consistency at integration of modeling branches. To finish an integration of development branches, all inconsistencies have to be identified and resolved. Syntactical inconsistencies in merged models can be identified automatically based on a related formal meta-model. However, semantic inconsistencies cannot be identified in general because the modeling intention of engineers is not automatically analyzable but some kinds of semantic inconsistencies can be automatically detected. A

![Figure 2. Simplified, concurrent modeling scenario](image)

![Figure 3. Example - Merged model VM](image)
computer-aided analysis of these inconsistencies (e.g. a retrieval for causative model modifications or involved modelers) can support a manual resolution. Furthermore, each inconsistency can be automatically classified related to its violated constraints in the metamodel. All these computable activities can be supported by a computer-aided integration of parallel modeling branches.

3.2.2. Parallel modifications of requirements and software architecture. Alice and Bob modify dependently an origin model version VO. The merged model VM is shown in Figure 3. The changes (additions) of Alice and Bob are marked by small green balloons. Alice has extented the origin model with two requirements – #NF-A42 and #F-A40 (additions A0, A1). She has formalized #NF-A42 by an architectural pattern – Layers – (A3) and has bound the introduced roles on all components (binding is represented by symbols: eye, cog, and DVD – A5-A12). Further, she has specified the semantics of the depend-relation by an example of concatenated representative relations (Usage+InterfaceRealization→2 – A4). Because of her new requirement #F-A40, Alice has added a new use case PrepareOffer for the system role MarketingSupportSystem (A2). Bob has added a new requirement #F-B17 (B0) and concerning its contents he has added a new use case PrepareOffer (B1). Furthermore, Bob extented the model by the new interface IFBankCounselorClient and two directed relations (usage / interface realization) (B2). The merged model VM contains several inconsistencies (regarding the meta-model in Figure 1). The identification and analysis of these inconsistencies is the topic of section 5.

4. Design of modeling languages

On the basis of modeling languages, different modelers are able to design an intuitive, generally intelligible, and semantically definite description of specific circumstances. For a consideration of computer-aided inconsistency management, the formal relations between model and its modeling languages play an important role. Furthermore, the presented approach should be applicable for any general purpose and domain specific modeling languages.

4.1. Model syntax definition by Meta Object Facility (MOF)

The Object Management Group provides the standard – Meta Object Facility (MOF) – language for the design of (domain specific or general purpose) modeling languages – especially meta-models. Unfortunately, the MOF specification provides only informal semantic descriptions. For this reason there are several efforts in the Eclipse Modeling Project (Eclipse Modeling Framework – EMF [9]) to give an implicit definition respectively Java implementation of a formal semantic specification for the essential MOF – EMOF/ECore. This approach based on source code generation from EMF models to Java code based on Java Emitter templates (JET).

4.1.1. Semantic mapping of domain specific metamodels basing on ECore. ECore represents a subset of the MOF meta-meta-model and facilitates the design of domain specific meta-models. The EMF project provides model-to-text transformations from ECore models to Java code. Additional OCL constraints can enrich ECore models to extent their expressiveness. This paper presents an alternative approach. In opposition to Java code respectively OCL, the semantic domain of the proposed mapping is represented by the OWL2 DL expressions. Therefore, it is necessary to implement a transformation which translates ECore models into an OWL2 representation of terminological knowledge (TBox). In the past the EODM project (currently attend by IBM) has implemented such transformation. The realized mapping is sketched in Figure 4. However, this mapping is nonsatisfying for extensive syntactic checking of models which based on ECore specified meta-models.

One primary disadvantage the EODM- transformation is the incomplete semantic mapping (e.g. attributes like containment and abstract are ignored). Furthermore the transformation proceed on the open world assumption but this fact does not
correspond to the generally used closed world semantics of ECore models.

![Diagram showing UML elements and relationships]

**Figure 5. Focus on use case containment**

For these reasons we implement an alternative and more complete ECore-to-OWL transformation which limits the ontology regarding CWA. The result of the transformation is an OWL-TBox which formalizes the semantics of an Ecore-based meta-model. A language designer can manually extent the automatically generated TBox by further OWL-expressions or by a library of conjunctive queries to prove additional constraints of static semantics similar to OCL-expressions. The modeler of the DSL-model (e.g. analyst or architect) does not have to specified any formal expressions to process a syntax check. To illustrate the approach, an exemplary semantic mapping of an excerpt of the introduced example model will be explained in the following. Therefore, Figure 5 depicts an excerpt of the ReqArch meta model from Figure 1. It focuses on the containment relation between system roles and use cases. The semantics of the depicted containment relation implies that the maximum one system role can contain an use case. The generated OWL syntax is shown in Figure 6.

Besides the semantic mapping of the ECore model ReqArch, the bank modeling which is illustrated in Figure 3, has to be translated in OWL. Therefore, all instantiated classes, references, and attributes are represented by facts in an ABox. An excerpt of the OWL representation of the example which is illustrated in Figure 3, is listed in Figure 7. Within this excerpt, the relations between use cases and roles are described. In [2, 21] a similar mapping from UML-diagrams to a DL ABox is presented. But in contrast to our TBox-generation approach, these researches presuppose an explicit TBox specification of an (UML) meta-model for consistency checks.

![Code snippet showing OWL representation]

**Figure 7. Excerpt of OWL ABox - Bank Infrastructure**

For a syntactical consistency check of the collaboratively developed bank model in the example scenario, the generated OWL expressions of the TBox (Figure 6) and the ABox (Figure 7) are joined together in one ontology. Section 5 explains how reasoning and query mechanisms enable this kind of analysis. However, for processing a semantic analysis of the domain specific model, an alternative knowledge base has to be generated. This translation is sketched in the next subsection.

### 4.1.1. Semantic mapping of domain specific metamodels basing on ECore

![Diagram showing containment relation between SystemRole and UseCase – represented in OWL TBox]
In contrast to the semantic mapping of ECore models, the semantics of a domain specific model cannot be specified independently of the DSL. It is rather one part of the DSL definition. In this subsection, the semantic mapping for the introduced example (Figure 3) is exemplarily illustrated. Goal of the mapping is the semantics definition based on an OWL TBox representation to analyze the model semantics regarding inconsistencies.

The model excerpt of the bank example focuses on the changes which are marked by A3, A4, and B3 in Figure 3. The OWL representation of Bob’s new interface relation (B3) is depicted in Figure 9. These changes cause an conflict to the specified layers pattern in Figure 8. The topic of the following section is the identification and analysis of this kind of inconsistencies.

```xml
<owl:ObjectProperty rdf:resource="depend"/>
<owl:Class rdf:about="PresentationLayer"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="depend"/>
  <owl:allValuesFrom rdf:resource="ApplicationLayer"/>
</owl:Restriction>
</owl:Class>
<owl:Class rdf:about="ApplicationLayer"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="depend"/>
  <owl:allValuesFrom rdf:resource="DataLayer"/>
</owl:Restriction>
</owl:Class>
<owl:Class rdf:about="DataLayer"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="depend"/>
  <owl:allValuesFrom rdf:resource="#Nothing"/>
</owl:Restriction>
</owl:Class>
```

Figure 8. TBox representation of Layers pattern

5. Inconsistency management based on an OWL representation of models

The generated OWL representation of models enables an inconsistency analysis by reasoning tools. Therefore the mentioned reasoning services (Section 2.1.1) can be applied. The illustrated model excerpt of Figure 3 contains two inconsistencies, one syntactic and one semantic inconsistency which are discussed in the following subsections.

5.1. Syntactic analysis

The containment relations between the use case PrepareOffer and the system roles CustomerManagement and MarketingSupport violate the meta-model constraint which is illustrated in Figure 5.

```xml
<owl:Restriction>
  <owl:onProperty>
    <owl:inverseOf rdf:resource="providedIFBankCounselorClient"/>
    <owl:ObjectProperty/>
    <owl:minCardinality rdf:datatype="xsd:nonNegativeInteger"/>
  </owl:onProperty>
</owl:Restriction>
```

Figure 9. OWL TBox – IFBankCounselorClient

This syntactic inconsistency can be identified by a OWL reasoner (cf. Subsection 2.1.1). The syntactic analysis for the presented and some further examples was evaluated by the Pellet reasoner. The following listing shows the output of Pellet analyzing the mentioned knowledge base.
The reasoner detects and analyses one inconsistency. It identifies that the use case PrepareOffer is contained by more than one individual. After the identification and initial analysis, an inconsistency can be analyzed by conjunctive queries on the generated ontology. In the presented example the use case PrepareOffer is contained by “more than 1” roles. Alice and Bob ask themselves: Which systems contain the use case PrepareOffer? Sure, in a small model the question is easy to answer but not in large model repositories. For this case, they can use (conjunctive) queries to analyze the detected inconsistency. The following listing (SPARQL [18] syntax) shows the mentioned question in head and the answer of the Pellet engine afterwards.

```sparql
SELECT DISTINCT ?X WHERE <
>  

| ?X |-------------------|
|-------------------|
| <http://bank/CustomerManagement> | |
| <http://bank/MarketingSupportSystem> | |
```

Accordingly, PrepareOffer is contained by CustomerManagement as well by MarketingSupportSystem. After the analysis Alice and Bob have to resolve the inconsistency in a collaborative way. Besides a syntactic analysis, a semantic analysis can be operated.

### 5.2. Semantic analysis

The basis of a semantic analysis is a DL-Lite TBox representation of the model (cf. Figure 1, Figure 9). To illustrate a semantic analysis of the merged bank model, the changes A3, A4, and B3 are focused in the following. Alice has defined a new architectural pattern (A3) and has specified the semantic of the depend-relation by two instances of the relations Usage and InterfaceRealization. Therefore, Alice's specification implies that the depend relation is interpreted as a concatenated relation of Usage and InterfaceRealization. But the usage relation from CustomerFile to BankCounselorClient which was added by Bob is in conflict with the approved dependencies. This is a semantic inconsistency and it can be detected by a class satisfiability check of an DL reasoner in a finite model as listed in the following output:

```
Reasoner Report
Warning (Unsatisfiable class):
http://bank.de#IBankCounselorClient

Because of the unsatisfiable depend relation, an individual (instance) of type IBankCounselorClient cannot exist. In the exact same manner of syntactic analysis, the identified inconsistency can be analyzed more comprehensively with conjunctive queries.

### 6. Conclusions and further work

The paper proposes an approach for a syntactically and semantically formal representation of models in model driven software engineering based on description logics. This representation enables the presented computer-aided conflict identification and analysis. Therefore, popular OWL2 and DL-Lite reasoners as Pellet and QuOnto provide consistency and concept satisfiability checks which can be used for conflict management. Furthermore, the paper presents an extented analysis of inconsistencies by conjunctive queries on the OWL representation of a merged model.

The further work is structured into two areas: The generation of an OWL representation from a (meta-) model will be refined to cover the whole ECore meta-model. Furthermore, other representation approaches especially for CWA modeling [13] should be evaluated for a semantically precise representation of models. On the other hand, the analyzing mechanisms providing by ontology debugging approaches should be evaluated regarding aptitude for conflict management in collaborative modelling.

### 7. References


