Feature Model Based on Description Logics*

Shaofeng Fan and Naixiao Zhang

LMAM, Department of Information Science,
School of Mathematical Sciences,
Peking University, Beijing 100871, China
{fsf, znx}@is.pku.edu.cn

Abstract. Intelligent interactive systems have begun to adopt knowledge and software engineering technologies in an attempt to effective development. Feature models have been widely used in knowledge and software engineering for the reuse purpose. However, due to the lack of a formal semantics of feature models, it is rather difficult to perform rigorous consistency reasoning on them. Without guaranteed consistency of feature models, the quality of interactive systems based on them, can not be guaranteed. In this paper, how to formalize feature models with Description Logics is investigated. Following the proposed translation principles, each feature model is formalized into an $\mathcal{ALCQI}$ knowledge base. Hence the consistency reasoning on the feature model turns into the consistency reasoning on the corresponding $\mathcal{ALCQI}$ knowledge base. Especially, the latter reasoning can be automatically performed via the description logic reasoner RACER.

1 Introduction

There is growing awareness of the importance of the development processes for intelligent interactive systems through knowledge engineering and software engineering approaches [7,10]. Domain engineering is a software reuse approach, which aims to develop reusable software assets focusing on a particular application domain. The most important result of domain engineering is the feature model [6]. The prominent and distinctive user requirements are denoted by common and variant features, which are in turn captured into a graphical feature model. So far quite a number of feature-centered domain engineering methods have been proposed, such as FODA [8], ODM [11] and KAPTUR [2].

By capturing user requirements of a particular application domain, feature models act as the start-point of software development. The quality of the interactive systems is strongly affected by the consistency of the feature models. Any system, based on an inconsistent feature model, is quite prone to be inconsistent. Therefore, the consistency reasoning becomes a critical problem. However, due to the lack of a formal semantics, there is no automated tool to perform consistency checking of a feature model. It is neither efficient nor reliable to validate

* This paper was supported by the National Natural Science Foundation of China under Grant. 60473056.

B. Gabrys, R.J. Howlett, and L.C. Jain (Eds.): KES 2006, Part II, LNAI 4252, pp. 1144–1151, 2006.

it by hand. Especially it becomes infeasible to accomplish the reasoning when there are a great deal of features and constraints between them.

Description Logics(DLs) is a formalism for representing knowledge and reasoning about it [1]. DL languages have formal model-theoretic semantics, and their main strength lies in the support of powerful reasoning mechanisms. Much attention has been paid on the application of DLs [1].

We believe that there is a strong similarity between description logics systems and feature models, both of which represent concepts in a particular domain and define how various properties relate among them. Hence, in this paper we propose feature models based on description logics. We present how to formalize a feature model with the DL $\mathcal{ALCQI}$. Thus the feature model is provided with the DLs formal semantics, and the consistency of feature models is translated to the consistency of the knowledge base consequently. Then the automated reasoning on the DL representation of the feature model can be performed using the DL reasoner–RACER. Thus the consistency of the feature model is checked automatically with high efficiency and reliability.

The remainder of the paper is organized as follows. Section 2 gives a brief overview of feature models and DLs. Section 3 presents our proposal of feature models based on DLs. A case study is given in section 4 to demonstrate our approach. In section 5, related works are compared and distinguished. Section 6 concludes the paper and indicates the future work.

2 Background

2.1 Graphic Feature Model

A feature model consists of a feature diagram and some additional information, such as rationale, constraints and dependency rules. A feature diagram provides a graphical tree-like notation that shows the hierarchical organization of features. It consists of a set of nodes, a set of directed edges, and a set of edge decorations. The root of the tree represents a concept node. All other nodes represent features.

Here we take the graphical notation introduced in [6]. Assuming a feature is selected, we have the following definitions on its child features:

- *Mandatory* feature: The feature must be included into the description of a concept instance, pointed to by a simple edge ending with a filled circle.
- *Optional* feature: The feature may or may not be included into a concept instance, pointed to by a simple edge ending with an empty circle.
- *Alternative* feature: Exactly one feature from a set of features can be included into a concept instance. The nodes of a set of alternative features are pointed to by edges connected by an arc.
- *Or* feature: One or more features from a set of features can be included into a concept instance. The nodes of a set of or features are pointed to by edges connected by a filled arc.

However not all arbitrary feature configurations have practical meaning. We identify three kinds of inter-dependencies among features, i.e. feature constraints:

- *Require* constraint: The presence of some feature in a concept instance requires the presence of some other feature.
- *Exclude* constraint: The presence of some feature excludes the presence of some other feature.
- *Cardinality* constraint: The cardinality constraint represents the quantity relation between features.

An instance of a feature model consists of an actual choice of features matching the constraints imposed by the diagram.

Definition 1. *Given a feature model, if its extension is not empty, i.e. there exists an instance satisfying the feature diagram and feature constraints, the feature model is said to be consistent.*

2.2 Description Logic $\mathcal{ALCQI}$

In $\mathcal{ALCQI}$, concepts and roles are built inductively from atomic concepts and atomic roles with constructors. The syntax and semantics of $\mathcal{ALCQI}$ is summarized in Table 1, where A and P denote atomic concepts and atomic roles, C and D denote concepts, R denotes roles, n denotes a strict positive integer. Then we can define $\bot$ as $\neg\top$, $\forall R.C$ as $\neg(\exists R.\neg C)$, and $(\exists^{\leq n} R.C)$ as $\neg(\exists^{\geq n+1} R.C)$. The semantics is specified through the notion of *interpretation* $\mathcal{I}$ that consists of a non-empty set $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$.

Table 1. The syntax and semantics of $\mathcal{ALCQI}$

Constructor	**Syntax**	**Semantics**
universal concept	$\top$	$\Delta^{\mathcal{I}}$
atomic concept	A	$A^{\mathcal{I}}$
concept negation	$\neg C$	$\Delta^{\mathcal{I}} - C^{\mathcal{I}}$
intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}}, (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
qualified number restriction	$\exists^{\geq n} R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geq n\}$
reverse role	P^{-}	$\{(o, o') \in \Delta^{I} \times \Delta^{I} \mid (o', o) \in P^{I}\}$

A DLs *knowledge base*(KB) $\mathcal{K}$ comprises two components, the TBox and the ABox. TBox (denoted as $\mathcal{T}$) is a finite set of terminological axioms. A concept C is satisfiable with respect to $\mathcal{T}$ if there is an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}}$ is nonempty. A TBox $\mathcal{T}$ is consistent if there is an interpretation $\mathcal{I}$ such that all concepts are satisfiable. Then the interpretation $\mathcal{I}$ is named a *model* of $\mathcal{T}$.

3 Formalization of Feature Models with $\mathcal{ALCQI}$

In order to automatically perform consistency checking on feature models, here we present the formalization of feature models into the $\mathcal{ALCQI}$ KB. Then the consistency checking of feature models will be translated into the consistency reasoning about the $\mathcal{ALCQI}$ KB.

3.1 Translation Rules

Given a feature model FM including a feature diagram FD and feature constraints, the corresponding KB $\mathcal{K}{=}\varphi(FM)$ can be gained by the following rules:

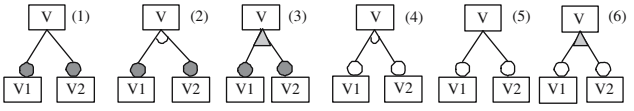

Fig. 1. Basic structures in feature diagrams

Step 1. Every node V in FD is formalized into an $\mathcal{ALCQI}$ concept C;

Step 2. Suppose node V and its child nodes $V_i (i \geq 1)$ are formalized into $\mathcal{ALCQI}$ concept C and C_i respectively. Then each edge connecting the node V and its child nodes V_i is translated into an $\mathcal{ALCQI}$ role R_i, which represents the relation between the concept C and C_i;

Step 3. The edge decorations can be translated into $\mathcal{ALCQI}$ terminological axioms. For each basic structure[1], as shown in Fig.1, we have:

(1) For structure (1), i.e. $mandatory(V_1, V_2)$, denoting V_1 and V_2 are *mandatory* features of V, we use the following terminology axiom to model it:

$$C \sqsubseteq \forall R_1.C_1 \sqcap \forall R_2.C_2$$

(2) Structure (2), i.e. $alternative(V_1, V_2)$, represents that V_1 and V_2 are *alternative* features of V. We introduce the following terminology axioms:

$$C \sqsubseteq C_1 \sqcup C_2,\ C_1 \sqsubseteq C,\ C_2 \sqsubseteq C \sqcap \neg C_1$$

The first axiom expresses the covering of subconcepts, and the latter two axioms ensure that the concept C_1 and C_2 are disjoint.

(3) Structure (3), i.e. $or(V_1, V_2)$, means that V_1 and V_2 are the *or* features of V. It is obvious that this structure can be expressed by the the combination of structure (1) and (2). Then making use of the rules (1) and (2), the following terminology axioms are introduced for structure (3):

$$C \sqsubseteq C_1 \sqcup C_2 \sqcup (\forall R_1.C_1 \sqcap \forall R_2.C_2)$$

$$C1 \sqsubseteq C,\ C_2 \sqsubseteq C \sqcap \neg C_1$$

$$(\forall R_1.C_1 \sqcap \forall R_2.C_2) \sqsubseteq C \sqcap \neg(C_1 \sqcup C_2)$$

[1] Feature diagrams can be normalized into the basic structures. Without losing the universality, we discuss the structures including two features in order to keep the presentation tersely.

(4) Structure (4), i.e. $mandatory(optional(V_1), optional(V_2))$, denotes parent feature V has two *optional* feature V_1 and V_2. It means that if V is included in a concept instance, V_1, V_2, neither, or both of them are included in the instance. This structure can also be expressed by the combination of structures (1) and (2), i.e.

$$alternative(V_1, V_2, mandatory(V_1, V_2), None)$$

Here $None$ represents the situation that none of the subfeatures of a parent feature is included in a concept instance. For $None$, we introduce an $\mathcal{ALCQI}$ atomic concept $Null$. Note that $Null$ is not equal with the $\mathcal{ALCQI}$ concept $\bot$, since $\bot^I = \emptyset$, while $Null^I \neq \emptyset$. Then corresponding axioms can be introduced by making use of the rules (1) and (2).

(5) For structure (5) and (6), the translation can be similarly accomplished as previous.

Step 4. For feature constraints, we have:

(1) V_1 *require* V_2, means that if feature V_1 is included in a concept instance, then V_2 must be included. To formalize this constraint, we need to find out the related information of edges between features. The edges from the nearest common ancestor node[2] of V_1 and V_2 to V_1, denoted by $E_{11} \ldots E_{1i}$, can be gained by the following algorithm:

```
// temp as variable and parent(temp) as parent node of temp
//CommonV as the nearest common ancestor node of V1 and V2
temp = V1
while(temp!=CommonV)
{ if it is mandatory structure between temp and its sibling node
      then note the directed edge E1i from parent(temp) to temp
  else   temp=parent(temp)
}
```

For feature V_2, the effective directed edges $E_{21} \ldots E_{2j}$ can be gained analogously. Then the following terminology axiom can be introduced:

$$(\forall R_{11}...(\forall R_{1i}.C_1)) \sqsubseteq (\forall R_{21}...(\forall R_{2j}.C_2)),$$

in which $R_{11}, ..., R_{1i}$ are atomic roles corresponding to $E_{11}, \ldots, E_{1i}$, and $R_{21}, ..., R_{2j}$ corresponding to $E_{21}, \ldots, E_{2j}$.

(2) Constraint V_1 *exclude* V_2 means that if feature V_1 is included in a concept instance, then V_2 must not be included, and vice versa. Then the corresponding terminology axiom is introduced:

$$(\forall R_{11}...(\forall R_{1i}.C_1)) \sqcap (\forall R_{21}...(\forall R_{2j}.C_2)) \sqsubseteq \bot,$$

declaring that no concept instance includes both feature V_1 and V_2.

[2] Obviously, there must be a common ancestor node of V_1 and V_2, because the root is one of their common ancestor.

(3) The *cardinality* constrains between features can be translated into terminology axioms with qualified number restriction constructors in $\mathcal{ALCQI}$.

The translation rules have covered all the elements of feature models. Hence any feature model can be translated into an $\mathcal{ALCQI}$ KB.

The correctness of the translation function φ can be proven by building two mappings, one from the concept instance of FM to the model of $\varphi(FM)$ and the other from the model of $\varphi(FM)$ to the concept instance of FM. Since the translation rules proposed here are properly intuitionistic, the proof details are omitted here due to space limitation. Readers interested in the validity of φ are referred to [5]. Note that the ABox of the KB $\varphi(FM)$ is empty because no individuals are involved in a feature model FM. Therefore the following discussion on KB reasoning is limited to the TBox of $\varphi(FM)$.

3.2 Automatic Reasoning Using RACER

Given a feature model FM, each instance of FM corresponds to a model of KB $\varphi(FM)$. If FM is consistent, then the set of instances of FM is nonempty, i.e. there must be an instance satisfying FM. Thus there must be a corresponding model of $\varphi(FM)$, so $\varphi(FM)$ is consistent, and vice versa. Hence we have:

Theorem 1. *Given a feature model FM and its corresponding $\mathcal{ALCQI}$ KB $\varphi(FM)$, FM is said to be consistent if and only if $\varphi(FM)$ is consistent.*

Now the consistency checking of a feature model is translated into the consistency reasoning of the corresponding knowledge base.

To accomplish the automatic consistency reasoning, we adopt the description logic reasoner RACER(*Renamed ABox and Concept Expression Reasoner*) [9]. From a KB $\varphi(FM)$, we can define its corresponding RACER script. By adopting RICE [9] as the reasoning client, we can load the script. Then we can use the following command to check the consistency of the knowledge base:

$$(check - tbox - coherence\ knowledgebasename\ (tbox(current - tbox)))$$

If the result is NIL, all the concepts are satisfiable, i.e. the knowledge base is consistent, which demonstrates that the corresponding feature model is consistent. Otherwise, the corresponding feature model is inconsistent.

4 Case Study

To clarify our approach, we present how to translate a feature model of the course concept into an $\mathcal{ALCQI}$ KB. The example feature diagram is shown in Fig.2. The constraints between the features include:

(1) "*Required*" *exclude* "*TA*" to ensure the teaching quality of the course.
(2)"*Phd*" *require* "*Prof*" to ensure the profundity of the course.
(3) Every course can only be taught by one teacher to avoid it to be reopened.
(4) Every teacher teaches at most one course to ensure the fair assignment of teaching tasks.

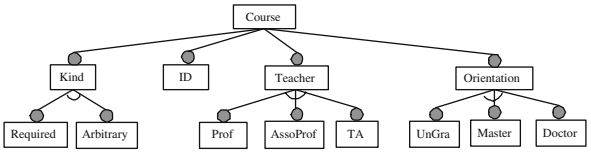

Fig. 2. Feature diagram of Course concept

According to the translation rules proposed in this paper, the above feature model is formalized into the following $\mathcal{ALCQI}$ KB, as shown in Table 2. Then we define the RACER script, and load it to the RACER knowledge base. Using the consistency checking command, we get the result 'NIL', which means that the current feature model is consistent.

Table 2. The fragment of $\mathcal{K}$ corresponding to the feature model of Course

$CCourse \sqsubseteq \forall kind.CKind \sqcap \forall id.CId \sqcap \forall taughtby.CTeacher \sqcap \forall orient.COrient$
$CKind \sqsubseteq CRequired \sqcup CArbitrary$
$CRequired \sqsubseteq CKind\ CArbitrary \sqsubseteq CKind \sqcap \neg CRequired$
.....
$(\forall kind.\ CRequired) \sqcap (\forall taughtby.\ CTA) \sqsubseteq \bot$
$\forall orient.\ CPhd \sqsubseteq \forall taughtby.\ CProf$
$CCourse \sqsubseteq (\exists^{=1} taughtby.\ CTeacher)$
$CTeacher \sqsubseteq (\exists^{\leq 1}\ taughtby^{-}.CCourse)$

If another constraint "every professor should teach two courses" is added, the corresponding RACER script should be updated by introducing another axiom: (implies *CProf* (at-least 2 *taught CCourse*)). Then the knowledge base will be checked to be no longer consistent. Through modifying the feather constraints, it is feasible to maintain the consistency of the feature model.

5 Related Work

With feature models being more and more widely used in the development of intelligent interactive systems, the formalization of feature models and the consistency reasoning on them have become considerably worthwhile issues. An adapted form of OCL is proposed in [12] to formally describe feature relations in feature models. However, the automated consistency checking of feature models was not further explored. In [13], the first-order logic in Z is used to formalize and verify feature models. Yet the cardinality constraint between features was not identified, which is naturally formalized using DLs in this paper. Benavides et al. present an algorithm to transform an extended feature model into a CSP [3], and further propose to use constraint programming to reason on feature models [4]. However, their feature models still lack the support of feature constraints, which is pointed out in [4] to be one challenge they have to face in the future.

6 Conclusion

In this paper, we have proposed an approach to formalizing feature models with description logics. The consistency checking of feature models can be performed automatically. Moreover, the reasoning is extraordinarily reliable within rigorous logic framework. In fact, through this approach feature models are provided with formal description logics semantics, which will facilitate the development of feature-oriented interactive systems in practice. Now we are investigating how to introduce typical concrete domains, such as numbers and time intervals, into $\mathcal{ALCQI}$ to express more kinds of feature constraints and exploring the experimental evaluation results about the consistency checking.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.
2. S. Bailin. Domain Analysis with KAPTUR. In Tutorial of TRIAda'93, Vol. I, ACM, NewYork, NY, September 1993.
3. D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Coping with automatic reasoning on software product lines. In *Groningen Workshop on Software Variability Management*, pages 1–14, 2004.
4. D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated reasoning on feature models. In *CAiSE'05,* LNCS. Springer, pages 491–503, 2005.
5. D. Calvanese, M. Lenzerini, D. Nardi Unifying class-based representation formalisms. Journal of Artificial Intelligence Research, 1999, 11(2):199–240.
6. K. Czarnecki, U. Eisenecker, Generative Programming: Methods, Tools, And Applications, Addison-Wesley, 2000.
7. Eui-Chul Jung, et.al. DIF Knowledge Management System: Bridging Viewpoints for Interactive System Design Proceedings of 11th Human Computer Interaction International Las Vegas, Nevada USA, July 22-27, 2005
8. Kang KC, Cohen SG, Hess JA, Novak WE, Peterson AS. Feature-Oriented Domain Analysis (FODA) feasibility study. Technique Report, CMU/SEI-90-TR-21.
9. V. Haarslev and R. MÄoler. RACER Users Guide and Reference Manual, 2004.
10. Ralf MÄoler. Reasoning about domain knowledge and user actions for interactive systems development. In: Proceedings IFIP Working Groups 8.1/13.2 Conference, Domain Knowledge for Interactive System Design, May, 1996.
11. M. Simos, D. Creps, C. Klinger, L. Levine, and D. Allemang. Organization Domain Modeling (ODM) Guidebook, Version 2.0. Technical Report for STARS, 1996.
12. D. Streitferdt, M. Riebisch, and K. Philippow. Details of formalized relations in feature models using ocl. In *Proceedings of the 10th International Conference and Workshop on the Engineering of Computer-Based Systems*, pages 297–304, 2003.
13. J. Sun, H. Zhang, Y. Li, and H. Wang. Formal semantics and verification for feature modeling. In *Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems.(ICECCS 2005)*, pages 303–312, 2005.