Co-evolution of Metamodels and Models based on Instantiation

Tian Wei[1, a], Lisheng Zhang [2, b]

[1]School of Computer Science and Technology Chongqing University of Posts & Telecommunications Chongqing, China

[2]School of Software Engineering Chongqing University of Posts & Telecommunications Chongqing, China

[a]wellwave@qq.com, [b]zhangls@cqupt.edu.cn

Keywords: co-evolution, description logic, instantiation, model.

Abstract: In model-driven engineering, the metamodel is one of the key artifacts of any model-based project. Similar to other artifacts, the metamodel also evolves, so the corresponding model needs to be adjusted to maintain its validity. In the co-evolution of traditional consistency change propagation, the constraint rules are manually defined, and the meta-model constraints cannot be automatically generated according to the meta-model. According to the hierarchical idea and meta-modeling mechanism of MOF, the co-evolution of traditional consistency change propagation is improved. In this paper, a co-evolution prototype based on instantiation is proposed. Based on the co-evolution prototype, according to the constraint rules of generating meta-model from meta-model, and then generating the model constraint from meta-model, an automatic generation from meta-model is proposed. Model constraint method. The case study of employee system metamodel and model evolution shows that the method is feasible.

1. Introduction

In model-driven engineering (MDE), models are usually precisely specified and defined by metamodels [1]. Like any other software artifact, the metamodel is easily evolved during daily use to address improvements, extensions, and corrections [2]. Since the evolution of the metamodel leads to inconsistency of the model, the model needs to be changed to ensure consistency with the metamodel. Various methods have been proposed for the co-evolution of the metamodel and the model [3], but the proposed solution is usually limited to Specific metamodels, or not all types of possible changes are not fully supported. Therefore, the co-evolution of metamodels and models is still a problem that needs to be studied.

This paper is to improve the existing co-evolution method of consistency change propagation. In the co-evolution of consistency change propagation [4], the constraints of the meta-model are automatically generated by manually defining the constraint rules instead of the meta-model. According to the layered idea of MOF [5] and the meta-modeling mechanism [6], the common evolution of the traditional consistency change propagation is improved, and a co-evolution prototype based on instantiation is proposed. On the basis of the proposed co-evolution prototype, according to the meta-model generates the constraint rules of the meta-model and the idea of generating the model constraints from the meta-model. A method for automatically generating model constraints from the meta-model is proposed.

2. Motivating Example

Although some metamodels (such as UML) are rarely standardized and changed, metamodels in specific domains often change. The co-evolution of metamodels and models is described below by an example of a metamodel and model of an employee system.

2.1 Simple Class Diagram Metamodel

In order to facilitate the analysis, some elements are selected according to the UML standard [7], and a simplified class element model is proposed as the meta-model, as shown in Figure 1. Here only some meta-model elements are selected as the core. The main elements are Class element, Generalization element, Association element, Property element, Datetype element, and the association between these elements.

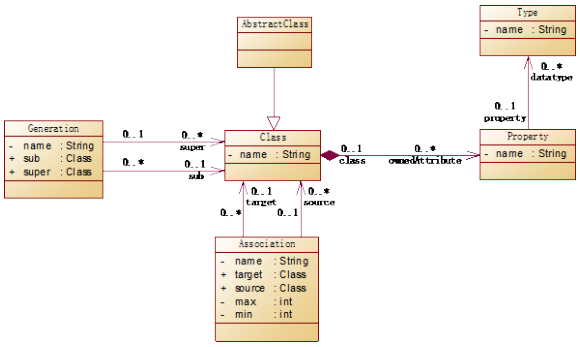


Figure 1. Meta-metamodel

2.2 Metamodel Evolution

2.2.1 Initial Employee System Metamodel

The metamodel is used to construct the metamodel. The metamodel in the example is based on the simplified version of the employee modeling language in the literature [8], as shown in Figure 2, in this simplified employee system metamodel. A person can be assigned to up to 2 Projects, and can choose to work in workplace or not in the workspace. Workplace mainly refers to Company or University.

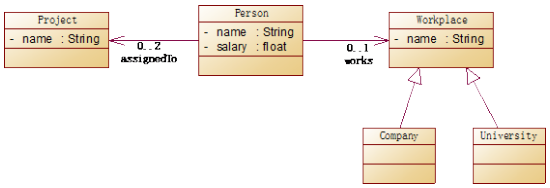


Figure 2. metamodel

From the object-oriented perspective, the employee system metamodel contains five classes: Project class, Person class, Workplace class, Company class and University class, of which Workplace class is abstract class. The Company class and the University class inherit the Workplace class. The Project class, the Person class, and the Workplace class all contain the name attribute. The data type is String. The Person class also contains the salary attribute, and its data type is float. There is an assignTo relationship between the Project class and the Person class, and the multiplicity of the association is 0..2. There is a work relationship between the Person class and the Workplace class, and the multiplicity of the association is 0..1.

2.2.2 Updated Employee System Metamodel

Due to changes in the company's business needs, the initial metamodel is to arrange personnel through the workplace. Now consider the arrangement of the roles of the personnel, the people working in the company as employees, and the people working in the school as external contacts. In the initial metamodel, people are assigned to up to two projects. Only employees can now assign work, and can only be assigned to at most one project. One project must specify a contact.

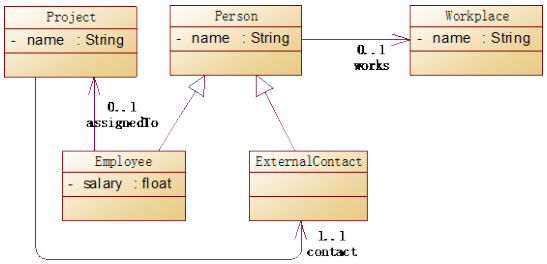


Figure 3. Updated metamodel

Therefore, the initial employee system metamodel of Figure 2 needs to be changed. The changed employee system metamodel is shown in Figure 3. The change to the initial employee system metamodel is to delete the Workplace subclass Company class and the University class, and the Workplace class is changed from the abstract class to the concrete class. Added Employee class and ExternalContact class, both of which inherit the Person class, and the Person class changes from concrete class to abstract class. The associated assignTo is changed from the initial meta model to the Person class associated with the Project class to the Employee class associated with the Project class, and its associated multiplicity is changed from 0...2 to 0...1. The new Project class points to the associated contact between the ExternalContact classes, and the associated multiplicity is 1.

2.3 Model Evolution

In Figure 4, a model that conforms to the initial employee system metamodel is depicted. The object graph instantiated as a model contains all the information about the model, and the object graph is used here to represent the model. The model contains two employees p1 and p2, two projects pr1 and pr2, the work site contains the school u1 and the company c1, the employee p1 works in the company c1 and is assigned to two projects pr1 and pr2, the employee p2 is The school works at u1 and is assigned to project pr2.

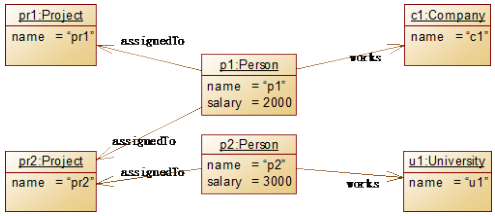


Figure 4. Initial employee system model

3. Co-evolution of Metamodels and Models based on Instantiation

For the metamodel evolution scenario mentioned above, we first explain how co-evolution is handled by existing methods. Then, we outline the key concepts involved in the instantiation of the metamodel based on the instantiation and the model, and discuss the main differences between our approach and the existing approach.

3.1 Traditional Co-evolution Approaches

Consistent change propagation [4] is a way to keep development artifacts consistent throughout the development process. Applying consistency change propagation to solve the problem of co-evolution of metamodel and model only considers the problem that the evolution of the metamodel leads to the common evolution of the model. Regardless of the evolution of the metamodel caused by the model, the change operation should not change the metamodel but only change the model. The co-evolution of consistency change propagation is shown in Figure 5.

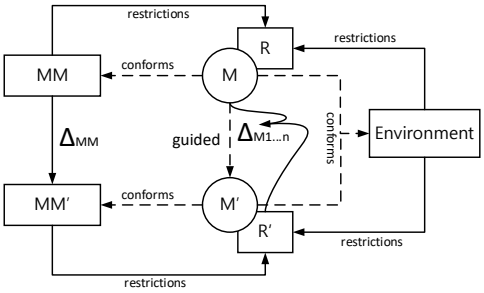


Figure 5. Co-evolution through consistent change propagation

The consistent change propagation process applicable to the processing of metamodel evolution in Figure 5 consists of two main phases, the first phase, which analyzes the consistency between the changed metamodel and the model through the constraints of the metamodel and the environment. In the second phase, all possible model changes are derived by making changes to the inconsistent elements in the model.

Although the coherent change propagation co-evolution method produces a model that conforms to the evolved metamodel, there are still some issues that remain unresolved. In the consistency change propagation, the constraint rules are manually defined, and the metamodel constraints cannot be automatically generated according to the metamodel. The constraints of the model are all derived from the metamodel, and the constraints of the metamodel should be automatically generated.

3.2 Co-evolution based on Instantiation

In order to solve the verification problem in co-evolution and improve the automation degree of verification, this paper describes the model idea and its instantiation method based on the meta-model, and proposes a co-evolution prototype based on instantiation based on the common evolution of traditional consistency communication. In Figure 6, the red part is used to represent an improvement to the traditional model evolution method.

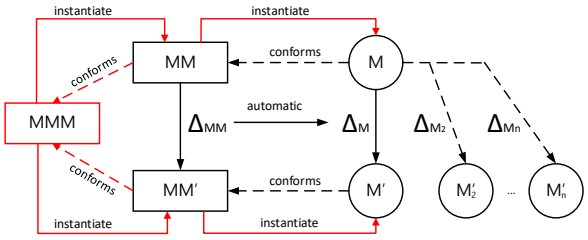


Figure 6. Co-evolution prototype based on instantiation

In the co-evolutionary framework based on instantiation-based consistency change propagation shown in Figure 6, MMM represents the meta-models before and after evolution, MM represents the meta-model before evolution, M represents the model before evolution, and MM' represents evolution. After the metamodel, M' denotes the evolved model, where M denotes one of the many models with subscripts, and denotes the set of changes between the metamodel and the model.

In the evolution process, the pre-evolution and post-evolution models must conform to the corresponding meta-models. These meta-models must conform to the common meta-model and be represented by conforms. In the instantiation of the metamodel into a metamodel, the metamodel must satisfy the constraint rules of all metamodels. The metamodel is instantiated into the model, and the model must satisfy the constraint rules of the metamodel. The evolved model must satisfy the constraint rules of the evolved metamodel.

According to the traditional co-evolution method of consistent propagation, the evolution process of the model can be regarded as the execution process of a series of changes. In the process of implementing these changes, it is necessary to verify whether the constraints specified by the corresponding metamodel are violated, due to the model and the metamodel. Inconsistent, the model

needs to be changed to ensure the consistency between the model and the metamodel. This paper adopts the method of verifying whether the metamodel can be instantiated as a model to verify the consistency of the model and the metamodel, and proposes a common evolution prototype based on instantiation.

4. Method for Automatically Generating Constraints

This section mainly discusses the method of automatically generating constraints from the metamodel. On the basis of the co-evolutionary prototype, according to the constraint rules of generating the metamodel from the metamodel, and then generating the model constraint from the metamodel, a kind of slave is proposed. The model automatically generates a model constraint method.

4.1 Generate Constraint Rules from the Metamodel

In order to generate the constraint rules of the metamodel from the metamodel, the elements, associations and inheritances and the associations between them are analyzed. The following describes the constraint rules using the description logic [9].

4.1.1 Class Element Constraint Rule

A class is the most basic model element in an object-oriented model. It describes a set of objects with the same characteristics, constraints, and semantics, and is the basis for constructing a class diagram. A class consists of three parts: a class name, an attribute, and an operation. The attribute is used to express the structure of the object, and the operation is used to express the behavior of the object. There is a generalization relationship between classes and classes, and the parent class is generalized into multiple subclasses.

According to the idea of the self-description of the class primitive model, the class is mainly described by the association between Class, Property and DataType. Use the description logic to represent the class element class primitive model as follows:

Class $\sqsubseteq \forall$OwnerAttribute.Property

Property $\sqsubseteq \geq 0$ class.Class$\sqcap \leq 1$ class.Class

Property $\sqsubseteq \geq 0$ dataType.DataType $\sqcap \leq 1$ dataType.DataType

Property $\sqsubseteq \geq 0$ class.Class$\bigwedge \leq 1$ class.ClassClass

DataType$\sqsubseteq \forall$ property.Property

4.1.2 Associated Meta-element Constraint Rules

Association is a structured relationship, which refers to the relationship between a class and another class. An association has two endpoints, called the association end, and the name of the association end is called a role. The multiplicity of the association end indicates the number of class objects connected to one end of the association. The value of the multiplicity may be a specific value or an integer interval.

According to the self-description of the class primitive model, the association is mainly described by the association between Class and Association and their association. The associated element class primitive model is represented by description logic as follows:

Association $\sqsubseteq \geq 0$ source.Class $\sqcap \leq 1$ source.Class

Class $\sqsubseteq \geq 0$ association.Association

Association $\sqsubseteq \geq 0$ target.Class$\sqcap \leq 1$ target.Class

4.1.3 Generalization Meta-element Constraint Rules

According to the self-description of the class primitive model, inheritance is one of the associations. The difference is that the meta-inherited element is Generalization, not Association. The inheritance element class primitive model is represented by the description logic as follows:

Generalization$\sqsubseteq \geq 0$ super.Class $\sqcap \leq 1$ super.Class

Class $\sqsubseteq \geq 0$ generalization.Generalization

Generalization⊑≥0 sub.Class ⊓≤1 sub.Class

4.2 Generate Constraints for the Metamodel

In order to obtain the constraints of the metamodel, the constraints of the metamodel are obtained according to the rules through the constraint rules of classes, associations and inheritance in the metamodel.

4.2.1 Generate a Knowledge base of Classes in the Metamodel

In the initial employee system metamodel shown in Figure 3, the Person class, Project class, Workplace class, and other elements are all objects instantiated by "Class" in the metamodel.

It can be obtained from the generalization relationship, the Person set contains the Employee set and the ExternalContact set. Use the description logic to represent the semantics of Employee as:

Employee⊑(∀name.String),Employee⊑(∀salary.float)

Use the description logic to represent the semantics of the ExternalContact class:

ExternalContact⊑ (∀name.String)

A parent class can be generalized to multiple subclasses, using description logic to represent a set of Employee classes and ExternalContact classes:

Employee⊑Person, ExternalContact⊑Person

Subclasses do not intersect each other, that is, the Employee class and the ExternalContact class do not intersect each other, and the description logic is expressed as:

Employee⊑¬ExternalContact, ExternalContact⊑¬Employee

The Project class defines a series of unique interrelated activities. The Project class contains the name attribute. The description logic is used to represent the semantics of Project:

Project ⊑ (∀name.String)

The Workplace class defines the place where the employee's daily work is located. The Workplace class contains the name attribute. The description logic is used to indicate the semantics of Workplace:

Workplace ⊑ (∀name.String)

For the multiplicity of attributes in the class of the employee system metamodel, the attribute attribute multiplicity in the Employee class is 1..1, the multiplicity of the salary attribute is 0..1, and the description logic is used to indicate the multiplicity of the name attribute:

Employee ⊑ ≥1name

Use description logic to indicate the multiplicity of the salary attribute:

Employee ⊑ (≥0salary) ⊓ (≤1salary)

4.2.2 Generate the Associated Knowledge base in the Metamodel

In the changed employee system metamodel, the multiplicity of associations between classes and classes implicitly defines the model constraints. The multiplicity of associated assignTo is 0..1, indicating that an employee can only participate in at most one project. The multiplicity of the associated contact is 1..1, indicating that a project needs to be designated as an external contact in a project. The multiplicity of the associated works is 0..1, indicating that an employee can only work in one work area at most.

Use the description logic to indicate the semantics of the associated assignTo as:

Employee ⊑ (≥0assignTo.Project) ⊓ (≤1assignTo.Project)

The semantics of using the description logic to represent the associated contact are:

Project⊑ (=1contact.ExternalContact)

Use the description logic to represent the semantics of the associated works as:

Person⊑ (≥0works.Workplace) ⊓ (≤1works.Workplace)

Since both the Employee class and the ExternalContact class inherit the Person class, the child class inherits all the properties of the parent class, so the Employee class and the ExternalContact class are associated with the Workplace class.

Employee⊑ (≥0works.Workplace) ⊓ (≤1works.Workplace)

ExternalContact⊑ (≥0works.Workplace) ⊓ (≤1works.Workplace)

4.2.3 Generating a Knowledge base for Metamodels

According to the instantiation idea, the initial employee system metamodel is instantiated from the metamodel and can be represented by an object graph, as shown in Figure 7 below.

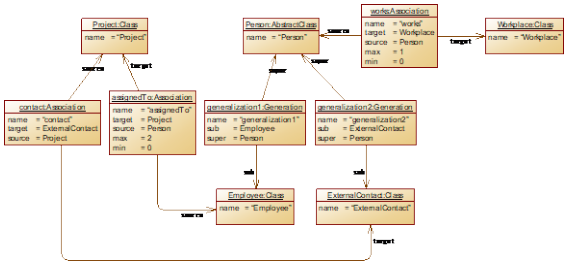


Figure 7. Object diagram of the metamodel after the change

Figure 7 is an object diagram of the simplified meta-model representation of the employee's system meta-model after the change, showing the rules for simplifying the class diagram of the simplified meta-model to the employee's system meta-model after the change, and using the description logic to represent the change according to the semantics of the object graph ABox knowledge base for post-employee system metamodels.

Class (Project), AbstractClass (Person), Class (Workplace), Class (Employee), Class= (External Contact),assignedTo(Employee,Project),works(Person,Workplace),Contact(Project,ExternalContact) ,generalization(Employee,Person),contact(ExternalContact,Person).

TBox and ABox form a complete domain-specific meta-model knowledge base, which is a constraint that the model must satisfy in the co-evolution of the meta-model and the model.

4.2.4 Generate Constraints for the Model

In the meta-model mechanism, the model is represented by a meta-model. One element in the model represents a set in the objective world, an instance of an element of the meta-model, and belongs to the individual concept. The object graph represents the process of instantiating the metamodel into a model. The element in the model is instantiated from which element in the metamodel. The ABox in the description logic represents the elements in the model and the metamodel. The corresponding relationship of elements.

The semantics of the metamodel is the constraint that the model must satisfy. The TBox in the description logic contains concepts and roles, which respectively represent the concepts and their connections in the metamodel, and completely represent the semantics of the metamodel or model. Therefore, the metamodel knowledge base The TBox in is the constraint that the model must satisfy.

In the meta-model, the class is instantiated as an individual in the meta-model, and the individual in the meta-model is used as a set in the model. The ABox in the meta-model knowledge base indicates which element in the model belongs to the set represented by the element in the meta-model. The element satisfies the constraints represented by the TBox in the metamodel knowledge base and must satisfy the expression in the TBox.

The meta-model is instantiated into the changed employee system metamodel. The individual elements in the meta-model represent the sets in the employee system meta-model, and the individual elements in the employee system meta-model represent the sets in the model, employees The elements in the system metamodel are a single entity for the metamodel, and for the model, the concept of a set, expressed as a description of the logical expression:

Project⊑Class,Workplace⊑Class,Person⊑AbstractClass,Employee⊑Class,ExternalContact⊑Clas s,assignTo⊑Association,contact⊑Association,works⊑ Association

The constraint rule knowledge base of the elements, related and inherited elements in the meta-model, the classes in the meta-model and the associated constraint knowledge base, the meta-model is instantiated into the changed employee system meta-model, and the represented

meta-elements The constraint knowledge base of the relationship between the model and the metamodel elements, the combination of these three phases is the constraint that the model must satisfy.

5. Automatic Evolution of the Model

5.1 Model Evolution Rules

According to the evolution of the meta-model, the evolution rules of the model are manually defined. In the evolution rules of the definition model, the constraints of the environment need to be considered. The constraints of the environment can only be handled manually, and constraints are imposed on the evolution of the model to ensure the model. In the case of satisfying the metamodel constraints, it is also necessary to satisfy the constraints of the environment. First, the evolution rules of the classes in the model are processed, and the associated evolution rules are processed.

In the evolution of the employee system metamodel class, the employees in the initial metamodel in the company's office become employees in the post-evolutionary metamodel, meaning that in the initial model, the instance element of the Person class is associated with the instance element of the Company class. Works, the instance element of the Person class needs to be changed to an instance of the Employee class. For example, p is an element in the Person set, c is an element in the Company set, and there is an associated works between p and c, so that p needs to be assigned to the Employee set, using the description logic expression as:

$$\text{Peson}(p) \wedge \text{Company}(c) \wedge \text{works}(e,c) \rightarrow \text{Employee}(e)$$

In the initial metamodel, the employees in the office at the office become the external contacts in the post-evolutionary metamodel, meaning that in the initial model, the instance elements of the Person class are associated with the instance elements of the University class, and the instance elements of the Person class. Need to change to an instance of the ExternalContact class. For example, p is an element in the Person set, u is an element in the University set, and there is an associated works between p and u, so that p needs to be assigned to the ExternalContact set, using the description logic expression as:

$$\text{Peson}(p) \wedge \text{University}\ (u) \wedge \text{works}(e,u) \rightarrow \text{ExternalContact}\ (e)$$

The model determines that the instance element of the Person class needs to be changed to an instance of the ExternalContact class. If an instance of the salary attribute exists in the instance element in the Person class, it needs to be deleted.

In the evolution of the employee system metamodel class, the schools and companies in the initial metamodel are collectively referred to as office locations after evolution, meaning that in the initial model, the instance elements of the Company class and the instance elements of the University class need to be changed to the Workplace class. An example. For example, c1 is an element in the Company set, u1 is an element in the University set, and both c1 and u1 need to be assigned to the Workplace set, using the description logic expression as

$$\text{University}\ (c1) \rightarrow \text{Workplace}\ (c1),\ \text{Company}\ (u1) \rightarrow \text{Workplace}\ (u1)$$

After the change of the class in the model is completed, considering the associated change, in the evolution of the employee system metamodel association, the employee in the initial metamodel is assigned to the maximum of two projects and becomes the post-evolutionary metamodel. Only the employee can be assigned at most. A project means that in the model, if the instance element of the Employee class is associated with the instance element of the Project class, the instance element of the Employee class needs to be changed to be associated with the instance of the Project class, and at most one instance of the Project class can be associated with it. . For example, p is an element in the Employee set, c1 and c2 are both elements in the Project set, and there is an association between the

p and c1, p and c2, then one of the associations must be deleted. At this point, you need to manually select which association to delete, using the description logic expression as:

$$\text{Employee}(p) \wedge \text{Project}(c1) \wedge \text{Project}(c2) \wedge \text{assignedTo}(p,c1) \wedge \text{assignedTo}(p,c2) \rightarrow \text{assignedTo}(p,c1)$$

$$\text{Employee}(p) \wedge \text{Project}(c1) \wedge \text{Project}(c2) \wedge \text{assignedTo}(p,c1) \wedge \text{assignedTo}(p,c2) \rightarrow \text{assignedTo}(p,c2)$$

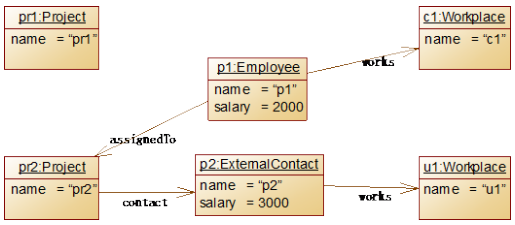
Similarly, projects in the post-evolution metamodel must specify an external contact, meaning that in the model, an associated contact is required between the instance element of the Project class and the instance element of the ExternalContact class. For example, p1 and p2 are both elements in the ExternalContact set, c is an element in the Project set, and there is an associated assignedTo between p1 and c, p2 and c, then one of the associations needs to be deleted, and the associated pointer is changed. Point to the instance of ExternalContact for the instance of Project, and change the association name to contact. At this point, you need to manually select which association to delete, using the description logic expression as:

$$\text{ExternalContact}(p1) \wedge \text{ExternalContact}(p2) \wedge \text{Project}(c) \wedge \text{assignedTo}(p1,c) \wedge \text{assignedTo}(p2,c) \rightarrow \text{contact}(p1,c)$$

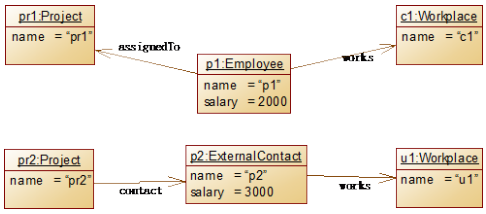
$$\text{ExternalContact}(p1) \wedge \text{ExternalContact}(p2) \wedge \text{Project}(c) \wedge \text{assignedTo}(p1,c) \wedge \text{assignedTo}(p2,c) \rightarrow \text{contact}(p2,c)$$

If there are special case model requirements, the evolution rules can only be made manually.

5.2 Automatic Generation of Evolved Models



(a)Revised ModelVersion 1



(b)Revised ModelVersion 2

Figure 8. Example Model Evloution

According to the constraint rules of the metamodel and the rules of model evolution, the evolved model is automatically generated and finally selected manually. In our method, multiple models can be generated by manually defining the evolution rules of the model according to the automatically generated metamodel constraints. In the initial employee system model, according to the evolution rule, the type of the object element in the model is first evolved. In the initial employee system model, the two object elements of the company c1 and the school u1 need to be changed to the Workplace type, and p1 needs to be changed to the Employee type, p2 Need to change to the ExternalContact type. In the re-evolution model, the association between p1 and c1, p2 and u1 does not need to be changed. P1 is associated with both pr1 and pr2, and it is necessary to manually select which association to delete. The association between p2 and pr2 needs to be changed from p2 to pr2

to pr2 to p2, and the association name is changed to contact. Figure 8 below is an example of two evolved models generated.

5.3 Model Consistency Automatic Verification

If a model is consistent with the metamodel, and the corresponding structural semantic set of the model satisfies the constraint rule set of the metamodel, then the model corresponds to the metamodel being logically consistent; conversely, if a model is inconsistent with the metamodel, the model corresponds The structural semantic set does not satisfy the constraint rule set of the class primitive model, then we call the model corresponding to the metamodel is logically inconsistent.

The description logic is used to describe the metamodel of the model, and the TBox in the model knowledge base is obtained. The description logic is used to describe the object graph that instantiates the metamodel, and the ABox in the model knowledge base is obtained. The TBox in the model knowledge base represents the constraints that the model must satisfy. The ABox in the model knowledge base describes which constraints the elements in the model need to satisfy.

In Figure 6, the description logic knowledge base of the metamodel MM and the metametamodel MMM is generated by the previous method. For the convenience of discussion, it is respectively recorded as K_{MM} and K_{MMM}. Each knowledge base contains two parts of TBox and ABox respectively. For TK_{MM}, AK_{MM} and TK_{MMM} and AK_{MMM}.

According to the idea of instantiation, the meta-meta-model MMM is instantiated into the meta-model MM. The set concept in TK_{MM} is that the AK_{MMM} contains the individual. Therefore, through the assertion in AK_{MMM}, it is judged which elements in the model MM must satisfy the constraints specified by the TK_{MMM}. condition.

Similarly, the elements in the model M must not only satisfy the constraints specified in the TK_M, but also must meet the constraints specified in TK_{MM} and TK_{MMM}. Using the recursive idea, it is easy to design a recursive algorithm to verify the consistency of the metamodel and the model.

6. Conclusion

In this paper, the common evolution of traditional consistency change propagation is improved, and a co-evolution prototype based on instantiation is proposed. On the basis of the proposed co-evolution prototype, the constraint rules from the meta-model are generated and the model constraints are generated from the meta-model. The idea proposes a method for automatically generating model constraints from meta-models, and explains how to deal with the evolution of employee system meta-models leading to the evolution of employee systems by instantiating co-evolution prototypes. Case studies show the effectiveness of proposed methods.

References

[1] Schmidt D C. Guest Editor's Introduction: Model-Driven Engineering[J]. Computer, 2006, 39(2):25-31.

[2] Fernandes K J. Evolving models in Model-Driven Engineering: State-of-the-art and future challenges[J]. Journal of Systems & Software, 2016, 111(3):272-280.

[3] Hebig R, Khelladi D E, Bendraou R. Approaches to Co-Evolution of Metamodels and Models: A Survey[J]. IEEE Transactions on Software Engineering, 2017, 43(5): 396-414.

[4] A. Demuth, M. Riedl-Ehrenleitner, R. E. Lopez-Herrejon, et al. Co-evolution of metamodels and models through consistent change propagation [J]. Journal of Systems and Software, 2016, 111 (6): 281 -297.

[5] OMG. formal / 2016-06-03. Meta Object Facility (MOF) 2.0 Query/ View/ Transformation Specification [S]. Needham: OMG, 2016-06.

[6] Liu Hui, Ma Zhiyi, Shao Weizhong. Research Progress of Metamodeling Technology [J]. Journal of Software, 2008, 19(6):1317-1327.

[7] OMG. formal/11-08-06. Unified Modeling Language (UML) Superstructure specification version 2.4.1[S]. Needham: OMG, 2011-08.

[8] Ruscio D D, Etzlstorfer J, Iovino L, et al. Supporting Variability Exploration and Resolution During Model Migration[C]// Modelling Foundations and Wael Kessentini, HouariSahraoui, Manuel Wimmer, Automated metamodel model co-evolution: A search-based approach [J]. Journal of Systems and Software, 2019, 106(1):74-88.

[9] Shi Lian, Sun Jigui. Summary of Description Logic [J]. Computer Science, 2006, 33(1):194-197.