2009 International Conference on Computer Engineering and Applications
IPCSIT vol.2 (2011) © (2011) IACSIT Press, Singapore

Framework Feature Oriented Model and Its Application in GEF Framework Modeling

Tiange Zhang[1], Xiaochun Xiao[1], Huan Wang[1] and Leqiu Qian[1]

[1] School of Computer Science, Fudan University, Shanghai, China 200433

Abstract. Object-oriented framework is increasingly recognized as an efficient reuse mechanism in software development, but because of their intrinsic abstract and complex, frameworks are still difficult to develop, integrate and instantiate. This is mainly due to the lack of framework specific modeling language. We introduce a framework feature oriented model language (FFOML) to model object-oriented framework specific concepts in a modular and compositional manner. FFOML is defined as an extension of UML metamodel following the MDA approach. GEF is a Graph Editing Framework, An experimental study on modeling and instantiation of GEF framework with FFOML is provided in this paper.

Keywords: Object-Oriented Framework, Metamodel, MDA

1. Introduction

A framework is a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact [1]. A set of customizable hotspots [7] are predefined as abstract classes or operations. Framework can be instantiate to a concrete application through implementation of these hotspots.

Framework has been proved to be one of the most efficient reuse mechanisms and is widely used in nowadays software application development [10]. But both framework development and framework instantiation are challenging. There are still few tools supporting framework development due to lack of framework model language on which tools can be built. Some works have researched on the formal and informal definition of framework [1][4][8], we argue that a model of object-oriented framework should has the following properties:

- **Modularity**: A real world framework can be large in size and it is usually difficult to be understood as a whole. To make framework easy to design and understand, it should be expressed in a modular way. We use feature concept as the natural basis of modularity in framework.
- **Compositionality**: More recently, it's become clear that application development is often based on multiple frameworks that have to be integrated with one another [3], so the model should support composition of framework in both internal and external level.

This paper addresses these principles by adopting a framework feature oriented model language (FFOML) as an extension of UML metamodel. Feature and role have been recognized as important concepts of framework for a long time [3][4][9], we integrate them into mainstream technologies and tools in a compositional and extensible manner.

The rest of this paper is organized as follow. In the next section introduces the definitions for basic conceptions such as framework, feature, role etc. Section 3 introduces advanced concepts such as feature extension and feature reference mechanism. Section 4 gives the example of GEF framework modelling and instantiation. Finally we exams some related work in this area and summarize this paper.

2. Metamodel for Basic Framework Concepts

Core concepts in framework modelling is feature, it is building block of framework model. A framework feature describes structure of roles and properties. Each role of specific role type performing a specialized

functions, and each property specify some aspects of this feature, they collectively specify some desired functionality of a framework feature. Framework features can also reuse other features form same or different framework by extension them or declaring references to them. This will be introduced in section 3.

Framework feature is a modular mechanism for framework modelling, it encapsulate hotspots and can provide proper granularity for designing, documenting, and understanding frameworks.

Figure 1 shows the metamodel of FFOML which is an extension of UML metamodel. Grey colored boxes are meta-classes from UML metamodel.

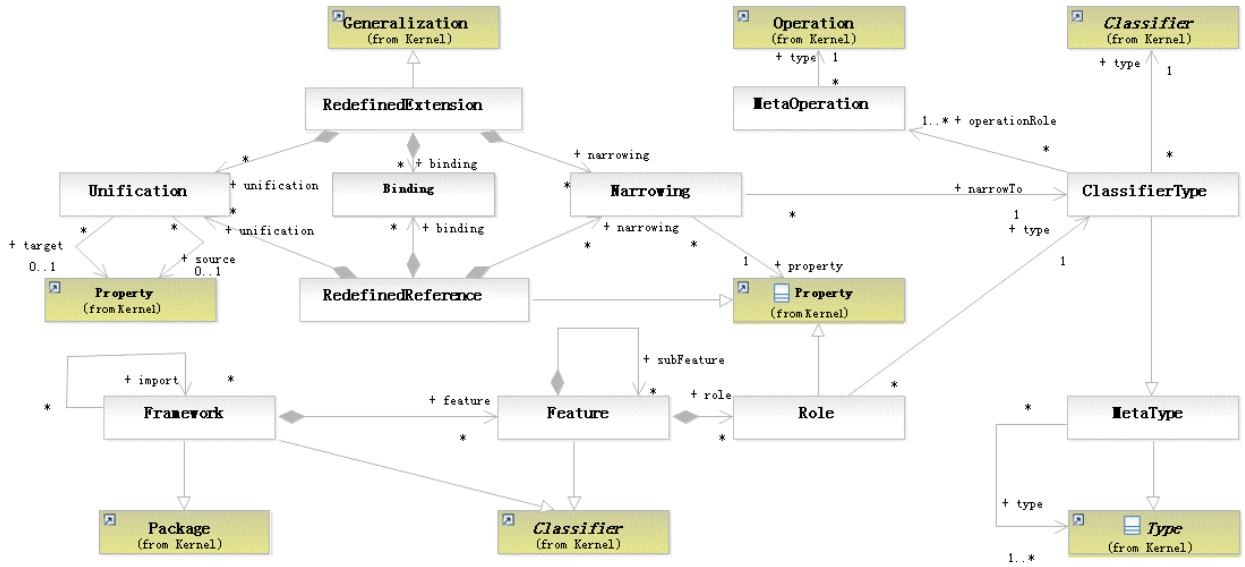

Fig. 1: FFOML Metamodel

Framework. *Framework* is defined as a *Package* from UML metamodel. It can be container of other *Packageable* model elements such as classes, interfaces and their relationships. It should be able to define properties necessary for its instantiation, such as title, deployment descriptors and so on, so framework is also defined as a subclass of *StructuredClassifier* which can own properties. The most important part of a framework is that it can own *Framework Features*.

Framework need to be integrated with each another in nowadays application development, so a framework can import other frameworks.

Framework Feature. As said before, framework feature describes a structure of role types and properties. It is a subclass of *StructuredClassifier* from UML metamodel. It should be noticed that a feature can contain not only roles descript in the next section, it also include normal properties such as title, size and so on. A feature can also contain sub-features and references to other features.

The concept of feature is borrowed from Feature-oriented domain analysis (FODA), which is used in product line software engineering [12]. It provides a parameterize mechanism for modelling product family. Framework is similar to product line software in that they all try to identify commonality and variability in a domain, but FODA is not sufficient for framework modelling because more complicated parameterize mechanism is needed in framework modelling.

Role. A role performs a specialized function for a feature. The concept of collaboration role is widely adopted by most of works on framework and design pattern modelling. Type of role should be *MetaType* which is defined below.

Meta Type. *MetaType* is type of type. Instance of a *MetaType* is a normal type from UML metamodel. *ClassifierType* is used to model role type as a specific *MetaType*, whose instance is subclasses of a *Classifier*. Because *MetaType* and *MetaClassifier* will be used heavily, we introduce a symbol @ into our concrete syntax to represent them. For example, *@[Integer, Float]* define a type whose instance is an *Integer* type or a *Float* type, and *@Thread* define a type whose instance is one of sub-class of *Thread*, *@Object* represents all classes in an object-oriented language system.

Meta Operation. Hotspots are key building block of framework. They are modelled by *MetaOperation* in our model. It is owned by *MetaClassifier* just as an operation is owned by a classifier. An instance of MetaOperation is an implementation of an operation. For example, *@AClassifier.aOperation* is a type whose instance is implementer of operation with signature *AClassifier::aOperation*. Instance of *@Thread.run* is an operation that overrides run operation declared in class *Thread*.

Sub-Feature. Features can be further organized in a hierarchy structure for modularity purpose. *Feature* can have sub-features. For example, a *Calculator* can own a sub-feature named *needPersist* that indicate whether it should be persisted for later query.

Application developer can instantiate sub-feature as an optional feature. In this way, functionality of a framework can be defined incrementally.

3. Metamodel for Compositional Framework Concepts

Modularity is the key to successful modelling. Model should be ability to decompose a problem into parts and then glue them together. In our case, new features could be defined by composing existing features. Just as class can be reused by extension or by composition, we can also identify two kinds of reuse mechanism in framework: feature extension and feature reference.

We find it is necessary to impose constraints while extent or make reference to other framework features. Three kinds of commonly used constraints are narrowing, binding and unification. They serve as glue for feature extension and feature reference. Additional constraints can also be defined using OCL supported by UML.

Narrowing. As we had defined, *@AClassifier* is a type whose instance is one of sub-class of *AClassifier*. Sometimes, a property of *@AClassifier* should be further restrict to a more concrete type, for example, *@BClassifier*, where *BClassifier* is a sub-class of *AClassifier*. We use narrowing to define this constraint.

A *Narrowing* specifies that a type should be 'narrow to' a new type which is a more concrete than the original one. We use notation →* to represent narrowing constraint in concrete syntax. Here is an example:

```
model →* @ConnectionModelElement
```

If the original type of model is *@ModelElement*, This narrowing expression will constraint its type from *@ModelElement* to *@ConnnectionModelElement*. *ConnnectionModelElement* should be subclass of *ModelElement*.

Narrowing is usually used in feature extension.

Binding. Assigning a role or a property to a value or instance of the corresponding type is a binding. If each role and property of a feature is bind to its instance, it is a full feature binding. Otherwise, it is a partial valued feature. A partial valued feature is still a feature which is to be instantiated by additional binding.

We use normal assignment to represent binding. Here is an example:

```
model = EllipticalShape
```

It means that the role of *model* will be played by concrete class *EllipticalShape* which is a subclass of *ModelElement*.

Unification. Binding a role or property to another role or property is an operation called unification. The unification operation *role1 = role2* constraints the values of *role1* and *role2* to be equal.

Any two partial valued roles or properties can be unified. If the partial values are already equal, then unification does nothing. If the partial values are incompatible, then they cannot be unified, and exception will be raised. Unification algorithm should support 'deep unification' to unify roles or properties with complex structure in a recursive manner.

We use normal assignment to represent unification too, just same as binding does. Here is an example:

```
Feature ContentOutlineSupport {
  editor:@Editor
  contentOutlineAdapter: AdapterFactory[
     adaptee = editor.editor;
```

```
    ]
    }
```

This states that *adaptee* property of referenced feature *AdapterFactory* should be unified with *editor* property of *editor* role.

Narrowing, *Binding* and *Unification* can be used to redefine a feature before reuse it. We support two kinds of reuse: reuse by extension and reuse by reference.

Feature Extension. Framework users (application developers) may want to extend the existing framework for their specific requirements. We model feature extension as a generalization. So the semantics of feature extension is same as class generalization concept from UML metamodel except that it can include additional constraints such as binding and narrowing.

Feature reference. A feature can reference to other features as its feature properties. Just as normal property, feature property can define multiplicity to specify the allowable cardinalities for an instantiation of this feature, such as optional [0..1], mandatory[1] or collection[0..*] etc.

4. Modelling and Instantiating GEF Framework

GEF is a graphical editing framework [16] based on eclipse RCP platform. It allows us to develop graphical representations for existing models. Just as most of complicated frameworks, it is difficult to be instantiated for beginners. There are several generators that can assist the instantiation of GEF [17][18], all of them are based on their own GEF model. We model GEF framework using FFOML, so it can integrate with other frameworks that modelled with FFOML too. Functionality of GEF framework is simplified here for the size limitation of paper.

Core concept of GEF framework is visual element. It specifies how model elements are mapped into visual figures. There are two different types of visual element: *GraphicalVisualElement*, *ConnectionVisualElement*. Visual elements can be grouped, which are modelled by *VisualElementGroup* feature. A *GEFEditor* feature has a root visual element and several visual element groups.

GEF framework is an extension of Eclipse Editor Framework. This provides us a chance to show how we can reuse existing framework features in model level. An eclipse editor can show content outline. To implement a *ContentOutlineSupport* feature, an *AdapterFacory* feature should be defined to adapt an editor to *ContentOutlinePage*.

```
Framework Editor{                                   (1)
    Feature Editor{                                 (2)
        id: String;                                 (3)
        title:String;
        icon: String;
        extensions[1..*]:String;
        allowMutiple: boolean default false;
        editor: @EditorPart{                        (4)
            createPartControl();                    (5)
        }
        isSelectionProvider: boolean;
    }
    Feature ContentOutlineSupport{
        editor: Editor[                             (6)
              editor.isSelectionProvider = true;
        ];
        adapter: AdapterFactory[
            target ->* @ContentOutlinePage;
            adaptee = editor.editor;
        ]
    }
    Feature AdapterFactory extends Adapter[
        adapter →: @IAdapterFactory;
        target→: @IAdaptable
    ]{
    }
    Class EditorPart{                               (7)
        createPartControl();
    }
    Class ContentOutlinePage{
        void createControl();
    }
}

Framework GEF{
    Import Editor as e;                             (8)
    Feature GEFEditor extends e::Editor[            (9)
        editor->*@GraphicalEditor;
        editor.createPartControl
              ->*@GraphicalEditor.createPartControl;
    ]{
        rootElement: VisualElement;
        elementGroup[0..*]: VisualElementGroup;
    }
    Feature VisualElementGroup{
        name: String;
        elements[0..*]: VisualElement;
    }
    Feature GraphicalVisualElement{
        name: String;
        icon: String;
        model: @ModelElement;
        editPart: @AbstractGraphicalEditPart{
              IFigure createFigure();
        }
    }
    Feature ConnectionElement extends VisualElement[
        model ->*@ConnectionModelElement;
        editPart->*@AbstractConnectionEditPart
    ]{
        source: @ModelElement;
        target: @ModelElement;
    }
    Class GraphicalEditor{}
    Class AbstractGraphicalEditPart{
        IFigure createFigure();
    }
    Class AbstractConnectionEditPart extends
                                  AbstractGraphicalEditPart{}
    Class ModelElement{}
    Class ConnectionModelElement extends ModelElement{
        ModelElement source;
        ModelElement target;
    }
}
```

There are two kinds of elements in a framework feature model (1): framework import (8) and feature definitions (2). Because FFOML is an extension to UML, it can contain UML elements such as interfaces and classes definitions as well (7).

A feature contains properties (3), roles (4), and feature references (6). *createPartControl*(5) is a hotspot contained in role of *@EditorPart* (4), which means that instance of a *@EditorPart* should extends *EditorPart* class and implements its *createPartControl* operation.

Feature can reference features defined in other frameworks. *GEFEditor* feature reference an *Editor* feature (9) that defined in an imported *Editor Framework* (8). Framework importing can make it possible to reuse other frameworks and tools that associated with these frameworks, such as generators, wizards.

FFOML is implemented as an Eclipse plugin. FFOML metamodel is defined in EMF[15]. Figure 2 is a user interface shows model and instance model of a GEF editor. Three steps are necessary in framework definition and instantiation.

Define framework model. Frameworks, classes, features and other model elements are defined in this phase. Fragments (2) and (3) in figure 2 are the model of GEF and Eclipse Editor Framework.

Instantiate framework. UML metamodel supports instance specification for classifier, which is super class of Feature. We can use UML instance specification model to model framework and feature instance. Fragment (1) in figure 2 specifies a GEF application as instance of GEF framework. This GEF application supports drawing of ellipses and connections that link two ellipses.

Generate Code. Code generator is implemented with xPand, a Model-to-Text template engine from oAW project[14]. Due to the modularity and compositionality of our framework model, the generator of Editor framework can be reused by GEF framework.

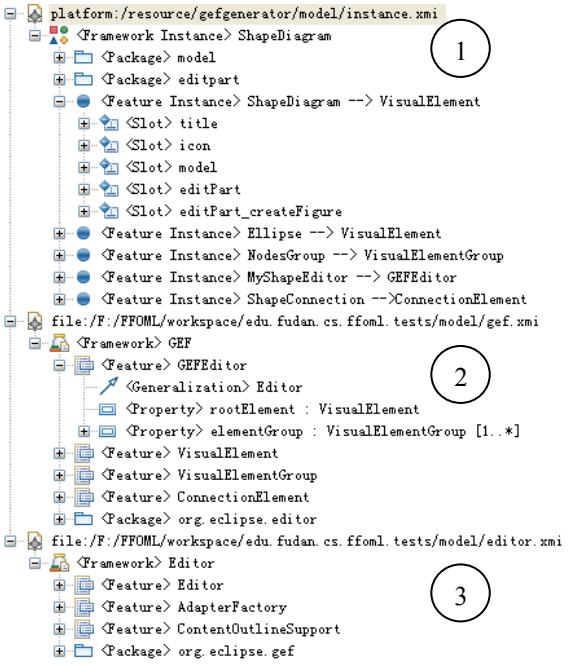

Fig. 2: GEF Model and instantiation

5. Conclusion & Related Work

This paper introduces an extension of UML metamodel for framework feature-oriented modelling. A number of works have been adopted for this field.

Ralph E. Johnson [1] is pioneer of object-oriented framework researcher. He identified most of essential features of framework and its relationship with other technology such as components, design patterns and domain specific language.

Nadia Bouassida et al.[11] propose a language as an UML extension for frameworks modelling, its formal semantic is defined using Object-Z. They focus on the modelling of the hot-spots, which is first introduced by Wolfgang Pree[7]. Hot-spots are implemented through hook classes and hook methods, which are modelled by MetaClassifier and MetaOperation in our work. We believe hot-spot is low-level concepts of framework. They are encapsulated in framework feature, and thus level of abstraction is raised.

Antkiewicz, M. et al[3] present a framework-specific modelling language approach to modelling and instantiation framework, it use feature model to descript functional requirements of framework. Some works use use-case diagram [11]. We chose feature model and make it a compositional unit to define new features.

Roles are often used in describing patterns and frameworks [4][8][9]. Roles are modelled by MetaType in our model which can be bind to concrete classifier and operation during instantiation process.

Overall, this work contributes a modular and compositional way to model important concepts of object-oriented framework. Results of our experimental study on modelling GEF shows that tools can be built based on it to guide the development and instantiation of framework.

6. References (This is "Header 1" style)

[1] Ralph E. Johnson, 'Frameworks = (components + patterns)', Communications of the ACM , Volume 40 , Issue 10 (October 1997) , Pages. 39 - 42.

[2] M. M. Bosch, M. E. Fayad , 'Framework Integration Problems, Causes, Solutions', Communications of the ACM, Volume 42 , Issue 10, Pages: 80 - 87

[3] Antkiewicz, M., Czarnecki, K.: 'Framework-Specific Modeling Languages with round-trip engineering', MoDELS, Lecture Notes in Computer Science, vol. 4199, pp.692-706 (2006)

[4] Unified Modeling Language, Superstructure, V2.1.2, OMG, 2007

[5] S.Kim, D.Carrington, "Using Integrated Metamodeling to Define OO Design Patterns with Object-Z and UML", Proceedings of the 11th Asia-Pacific Software Engineering Conference, Pages: 257-264

[6] Gamma, E., Helm, R., Johnson, R., and Vlissides, J.: 'Design patterns: elements of reusable object-oriented systems' , Addison-Wesley, 1995

[7] W. Pree, Hot-Spot-Driven Framework Development， http://www.softwareresearch.net/site/publications/J016.pdf

[8] Davide Brugali,Katia Sycara, 'Frameworks and Pattern Languages: an Intriguing Relationship', ACM Computing Surveys (CSUR), Volume 32 , Issue 1es , Article No. 2 (March 2000)

[9] Dirk Riehle,Thomas Gross, 'Role Model Based Framework Design and Integration', ACM OOPSLA' 98, Pages:117-133

[10] Mohamed E. Fayad,Ralph E. Johnson, 'Domain-Specific Application Frameworks: Frameworks Experience by Industry', John Wiley & Sons, 2000

[11] N. Bouassida,H. Ben-Abdallah,F. Gargouri,A. B. Hamadou, 'Formalizing the Framework Design Language F-UML', Proceedings of the First International Conference on Software Engineering and Formal Methods (SEFM'03)

[12] Kwanwoo Lee,Kyo Chul Kang,Jaejoon Lee, 'Concepts and Guidelines of Feature Modeling for Product Line Software Engineering', Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools,2002

[13] Eclipse Foundation:Eclipse. http://www.eclipse.org

[14] openArchitectureWare(oAW) : http://www.openarchitectureware.org/

[15] Eclipse Foundation: Eclipse Modeling Framework Project(EMF) http://www.eclipse.org/modeling/emf/

[16] Eclipse Foundation: Graphical Editor Framework (GEF). http://www.eclipse.org/gef

[17] Eclipse Foundation: The Eclipse Graphical Modeling Framework (GMF). http://www.eclipse.org/modeling/gmf/

[18] The Open-Source Toolkit for Critical Systems (Topcased). http://topcased.gforge.enseeiht.fr/