http://www.zju.edu.cn/jzus; E-mail: jzus@zju.edu.cn
ISSN 1009-3095 *Journal of Zhejiang University SCIENCE* V.5, No.1, P.92-98, Jan., 2004

A UML profile for framework modeling

XU Xiao-liang(徐小良)†, WANG Le-yu(汪乐宇), ZHOU Hong(周泓)
(*Department of Instrumentation Science & Engineering, College of Biomedical Engineering & Instrument Science, Zhejiang University, Hangzhou 310027, China*)
†E-mail: zjuxxl@sohu.com
Received Dec.12, 2002; revision accepted Apr.4, 2003

Abstract: The current standard Unified Modeling Language (UML) could not model framework flexibility and extendibility adequately due to lack of appropriate constructs to distinguish framework hot-spots from kernel elements. A new UML profile that may customize UML for framework modeling was presented using the extension mechanisms of UML, providing a group of UML extensions to meet the needs of framework modeling. In this profile, the extended class diagrams and sequence diagrams were defined to straightforwardly identify the hot-spots and describe their instantiation restrictions. A transformation model based on design patterns was also put forward, such that the profile based framework design diagrams could be automatically mapped to the corresponding implementation diagrams. It was proved that the presented profile makes framework modeling more straightforwardly and therefore easier to understand and instantiate.

Key words: Object-oriented frameworks, Unified Modeling Language (UML), UML profile, Hot-spots, Design patterns
Document code: A **CLC Number:** TP311

INTRODUCTION

Object-oriented (OO) frameworks became more and more popular in the software industry during the 1990s. A framework is a set of cooperating classes that make up a reusable design for a specific class of software (Deutsch, 1989; Johnson and Foote, 1988). Applications built on top of such a framework reuse not only its source code but also more important its architecture design (Pree, 1994; 1995; Fayad and Schmidt, 1997; Markiewicz and Lucena, 2001). Consequently, the use of frameworks reduces the costs and improves the software quality greatly.

However, development of such OO frameworks is difficult because the framework hot-spots must be modeled in a straightforward and flexible way. As an OMG standard, the Unified Modeling Language (UML) (Booch *et al.*, 1999; OMG, 2002) may be regard as a notational basis for OO framework modeling. However, the current standard UML does not support framework modeling adequately due to lack of appropriate elements and structures to describe the framework hot-spots and their instantiation restrictions. Although some techniques, such as design patterns, have been used to represent hot-spots with the combination of standard object structures (Hakala *et al.*, 2001; Gamma *et al.*, 1995), they may lead to the complexity of the framework design diagrams and difficulty in identifing the hot-spots in these diagrams. Fortunately, UML has extension mechanisms that specify how specific UML modeling elements are customized and extended with new semantics. A coherent set of such extensions, defined for specific purposes, constitutes a UML profile. A profile is a stereotyped package that contains model elements that have been customized for a specific domain or purpose by using stereotypes, constraints and tagged values (OMG, 2002; D'Souza *et al.*, 1999).

FRAMEWORK MODELING USING STANDARD UML

Fig.1 is a simple representation of the web-based "Order Framework" in EC (Electronic Commerce) domain with standard UML, where (a) shows a static view of the framework with UML class diagram, and (b) provides a dynamic view with UML sequence diagram. The static

view describes the inherent properties of a class, while the dynamic view illustrates the interaction between instances of classes. There are four major methods in the OrderProduct class: order( ), which allows customers to order the desired goods, addtoCar( ), which confirms the order and saves corresponding information, showOrderInfo( ), which shows information on the confirmed orders, and tipsperDay( ) shows the notices or information on how to order goods.

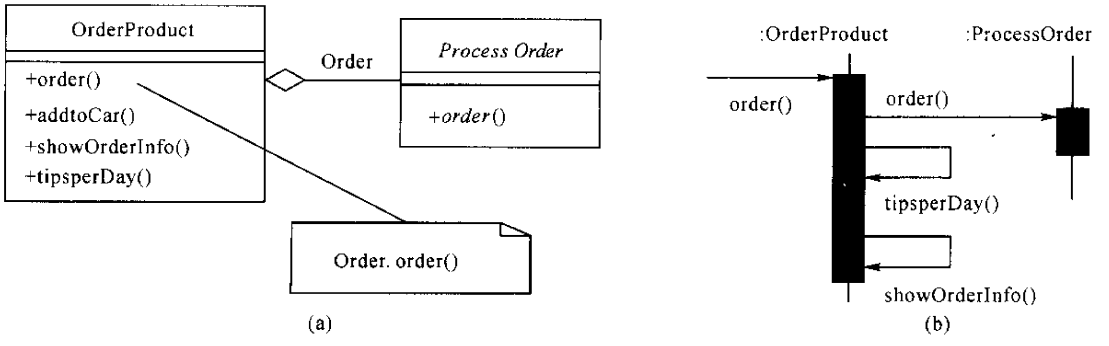

Fig.1 Representation of a framework with UML

(a) a static view of the framework with UML class diagram; (b) a dynamic view with UML sequence diagram

The method order( ) is a framework hot-spot since it may have different implementations in different specific EC applications based on this framework. For example, it may require the members to log on or non-members to fill out personal information forms when starting to order goods, show everyday tips on ordering goods or not, set discount on goods or not, and so on. There are numerous possibilities that depend on the specific application requirements. Fig. 1 models order( ) as an abstract method of an abstract class ProcessOrder. During framework instantiation, the framework users have to create subclasses of ProcessOrder and then provide concrete implementations of the order( ) method. The problem with this representation is that there is no indication that this method is a hot-spot in the design diagrams. There is also no indication of how it should be instantiated. Although the name of the abstract method is italicized, this notation does not represent a hot-spot because there are differences between abstract methods and hot-spots.

The tipsperDay( ) is also a framework hot-spot because it may be wanted in some framework instances but unwanted in others. The framework should provide only the methods and information that are useful for all the possible instantiated applications and the extra functionality should be provided only in framework instances. The inclusion of this kind of method to a framework class would lead to a complex interface of the class, and thus result in changing the implementation of an existing class and compiling it every time when new functionality is added.

The Product class hierarchy is used to let new types of products be defined depending on the specific application requirements. For example, the default product types that must exist in any framework instance are books and computers; however, new types may be needed such as telephones and cameras. Therefore, the Product class hierarchy also represents a hot-spot since it allows the definition of new subclasses depending on the requirements of a specific framework instance. Unfortunately, this hot-spot is not properly indicated in UML diagrams presented in Fig. 2. Fortunately, UML offers a constraint {incomplete} to indicate that new classes may be added to a given generalization relationship.

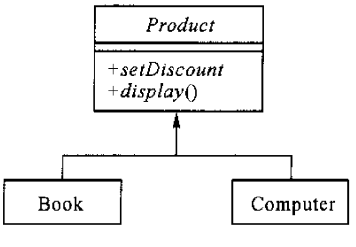

Fig.2 The product class hierarchy

This constraint is adopted as part of the profile elements by altering its semantics slightly, as will be illustrated in the next section.

A UML PROFILE FOR FRAMEWORK MODELING

The preceding section shows that the standard UML does not provide appropriate constructs to explicitly distinguish framework hot-spots from kernel elements. This section presents a UML profile that may customize UML for specific requirement of framework modeling. The profile is a group of extensions to a UML subset, which stresses class, object, and sequence diagrams that suffice for framework modeling. However, this does not mean that the rest of UML should not be used. Instead the profile helps the framework modelers to identify a subset of UML that allows the modeling of the important features of a framework.

Now we introduce the profile for modeling several important kinds of hot-spots, i.e. the variable methods, method-extended classes and type-extended interfaces presented above. The variable methods are methods that have a well-defined signature, but whose implementation may vary in each framework instance. The method-extended classes are classes that may have their interfaces extended during the framework instantiation. Finally, type-extended interfaces are interfaces or abstract classes that allow the creation of concrete subclasses during the framework instantiation.

Fig.3 is an illustration of these three important kinds of hot-spots with extended class diagram in the profile, which represents and classifies the three hot-spots explicitly. The tagged values {variable} and {extensible} extend the UML class definitions to identify two different kinds of hot-spots: the variable method and the method-extended class hot-spots. In this example, {variable} refers to the order( ) method, and {extensible} applies to the OrderProduct class. The {incomplete} constraint is used to identify type-extended interface hot-spots, meaning that new subclass of Product may be defined in framework instances. The stereotype 《instance》 indicates classes that are not part of the framework structure, but may be defined in framework instances. The {incomplete} constraint allows several 《instance》 classes to be created from a given type-extended interface during framework instantiation. In this example, applications created from this framework always include two types of products, i.e. books and computers, but several other types may be defined depending on the specific application requirements. The constraint {restricted} is used together with {incomplete}, meaning that the 《instance》 classes created from the relationship should not extend its interface. In addition, each hot-spot must be identified either by the tagged value {dynamic} or {static}, which is used for indicating whether it needs run-time binding and reconfiguration or not.

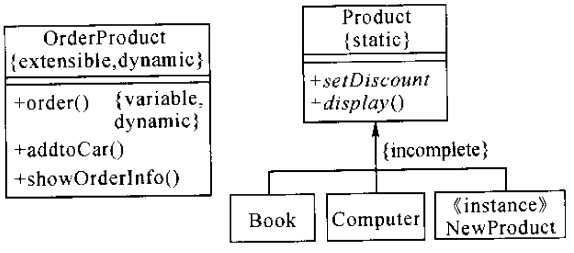

Fig.3 The extended class diagram

The profile also provides two ways to specify instantiation restrictions for the hot-spots: notes and hot-spot sequence diagrams. The notes are OCL specifications (OMG, 2002) that are similar to the standard UML. However, the former has semantics different from those of the latter when attached to the hot-spots. In the case of variable method hot-spot, it means that all variable methods' implementations should follow the specification during instantiation. In the case of method-extended class hot-spot, it applies to all methods that might be added to the class during instantiation. In the case of type-extended interface hot-spot, it applies to all methods that may be overridden in each 《instance》 class. Hot-spot sequence diagrams are extensions to UML sequence diagrams. They are an alternative way to describe desired interaction sequences of the framework hot-spots. A simple example is presented in Fig.4, where the {xor} constraint is applied to a set of interactions, specifying that only one interaction over that set is manifest for each framework instance. The {optional} constraint indicates the interaction that perhaps do not occur in the framework instance. Fig.4

shows that a concrete method which may instantiate from the order( ) variable method must have the following behaviour:

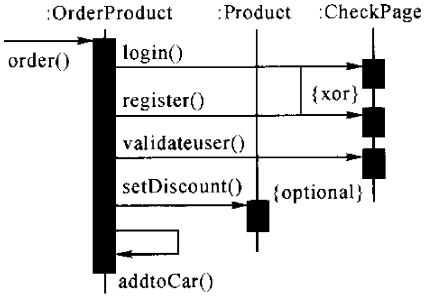

Fig.4 Hot-spot sequence diagram

(1) It requires the customer to log on or to fill out register forms when starting to order goods, depending on whether the customer is a member;

(2) It has to validate the data by checking whether the login or register information is valid;

(3) It may set different discount to the goods depending upon different members. Non-members may have not the qualification of this favourable service.

(4) It has to save the corresponding information of the order.

Generally, hot-spot sequence diagrams are used to describe a behaviour set that should be followed during the hot-spot instantiation; while OCL notes are generally used to specify invariants that should be satisfied in the hot-spot instances. Thus, both of them complement each other for describing the instantiation restrictions of framework hot-spots.

Except the three kinds of hot-spots mentioned above, there are other ones, such as variable attribute types, variable method parameter types, and so on. Coplien(1999) described several kinds of variability problems in his multi-paradigm design work. They will also be integrated well into the profile using similar principles to the ones described in this paper. To avoid the explosion of the number of extensions and to illustrate the presented profile feasible, this paper mainly focuses on the three kinds of hot-spots.

Table 1 summarizes the new elements and their informal semantics presented in the profile for the three hot-spots and their instantiation restrictions.

Table 1 Summary of the new elements and their semantics in the profile

New elements	Extension type	Applies to	Semantics
《instance》	Stereotype	Class	New classes that may be defined in the framework instances, but not exist in the framework kernel.
{variable}	Tagged value	Method	It is used to identify the variable methods. It means that the method implementations may vary depending on the framework instantiation.
{extensible}	Tagged value	Class	It is used to identify the method-extended classes. It means that new methods may be defined to extend the class functionality during the framework instantiation.
{static}	Tagged value	Three proposed framework hot-spots	It means that the hot-spot can be decided at compile time and does not require run-time reconfiguration.
{dynamic}	Tagged value	Three proposed framework hot-spots	It means that the hot-spot must require run-time reconfiguration.
{incomplete}	Constraint	Generalization and Realization	It identifies the type-extended interfaces. It means that new subclasses, which satisfy the generalization or realization, may be defined during framework instantiation.
{restricted}	Constraint	Generalization and Realization	It is used together with {incomplete} and means that the instance classes created from the relationship should not extend their interface.

Continued in the next page

New Elements	Extension Type	Applies to	Semantics
{optional}	Constraint	Interaction	It means that the given interaction may be omitted during the framework instantiation.
{xor}	Constraint	A set of interactions	It represents a dashed line connecting a set of events, specifying that only one over that set is manifest for each framework instance. It corresponds to if-then-else or case programming constructs.

FRAMEWORK IMPLEMENTAION

Standard UML does not provide important constructs for explicitly representing framework flexibility and variability requirements. The profile for framework modeling addresses this problem by representing the hot-spots as first-class citizens and thus makes the framework design more straightforward. Unfortunately, the new elements are not concerned with how to implement these hot-spots, but just with how to represent them appropriately at design level. Neither the variable methods nor method-extended classes can be directly implemented since standard OO languages lack appropriate constructs to represent them, but the type-extended interface hot-spots can be directly implemented through standard inheritance.

To bridge this design-implementation gap, Design patterns are considered as one possible solution. Design patterns are design techniques that do solve recurring design problems (Gamma *et al*., 1995). Based only on type-extended interfaces, several patterns provide solutions to the flexibility and extensibility problems. The strategy design pattern allows a family of interchangeable algorithms to be used and lets them vary independently from clients that use them. The visitor strategy lets you define a new method without changing the class on which it operates (Gamma *et al*., 1995). Consequently, the strategy and visitor design patterns can be respectively used to describe the variable method and method-extended class hot-spots with type-extended interface hot-spots, and thus the design diagrams using the profile can be mapped into appropriate implementation diagrams. Fig. 5 shows this mapping approach between design and implementation, which consists of the following major steps:

(1) Search all variable method hot-spots and transform them to type-extended interfaces based on the strategy design pattern;

(2) Search all method-extended class hot-spots and transform them to type-extended interfaces based on the visitor design pattern;

(3) Copy all standard UML elements and type-extended interfaces elements from the transformed models to the corresponding implementation models.

```
designPatterns(project, NewProject):
  (1) forall (variableMethod(...), strategy(...))
  (2) forall(extensibleClass(...), visitor(...))
  (3) copyUMLElements(...)
      forall (extensibleInterface(…), copyExtensibleInterface(…)
      …
```

Fig.5 Design pattern-based implementation model

Fig.6 illustrates such a mapping. This mapping can be completed automatically using computer-aided tools and thus may greatly reduce the framework development efforts.

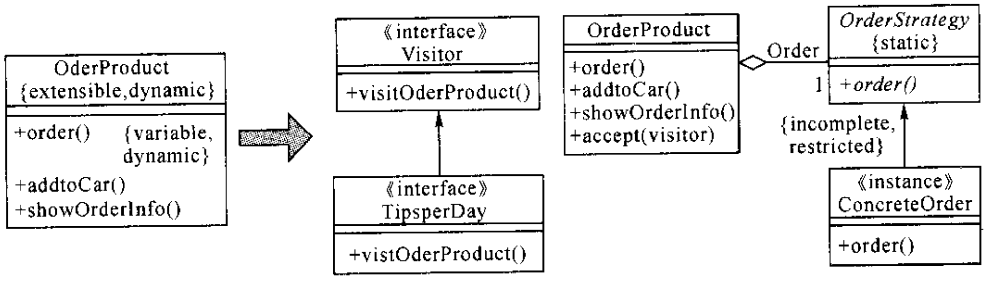

Fig.6 Mapping from design to implementation for framework's hot-spots

During framework instantiation the hot-spots have to be filled with application specific code. Since the variable methods and method-extended classes have been eliminated during implementation, the application developers just provide《instance》classes to complete the definition of the type-extended interface hot-spots in the implementation models. Obviously, the diagrams using the profile make this task very straightforward since all the type-extended interfaces and their corresponding instantiation restrictions are marked in the diagrams explicitly. When accomplishing the instantiation, all type-extended interfaces also disappear from the design diagrams, where the {incomplete} tagged values become {complete}, and then a complete application is created from the framework.

CASE STUDY

In this section, we propose a case study for the whole EC domain framework design using the presented profile. Generally, the framework of EC domain applications is composed of three major sub-frameworks: "Order Framework", "Payment Framework" and "Delivery Framework". The "Order Framework", which had been discussed in the preceding sections, consists of three main classes: Customer, Product and OrderProduct class, where Product class is a type-extended interface hot-spot, OrderProduct class is a method-extended class hot-spot, and order() method in OrderProduct is a variable method hot-spot. In the "Payment Framework", Payment class is the core class and consists of three major methods: getOrder(), which gets the order information indexed by order id, amount(), which quantifies total prices for an order, and pay(), which manages checking out payment for goods. Among these methods, pay() is a variable method hot-spot since there are several different payment approaches, such as payment by cash or by cheque. Delivery class, which is the core class of the "Delivery Framework", manages accounting of delivery cost, delivery, return of ordered goods, and so on. It is a type-extended interface hot-spot since there are different ways of delivery depending on the specific requirements. Fig. 7 shows the extended class diagram for the whole framework with the profile (its implementation class diagram is presented in Fig. 8) and thus framework instances will also be constructed just by providing specific《instance》classes to complete the definition of the type-extended interface hot-spots shown in Fig. 8.

CONCLUSIONS

The current UML does not provide appropriate constructs to explicitly identify and classify framework hot-spots and their instantiation restrictions. The UML profile for framework modeling addresses this problem through the extension to UML 1.4. The new elements in the profile are used to model flexibility and extensibility mechanisms of the framework as built-in elements. In

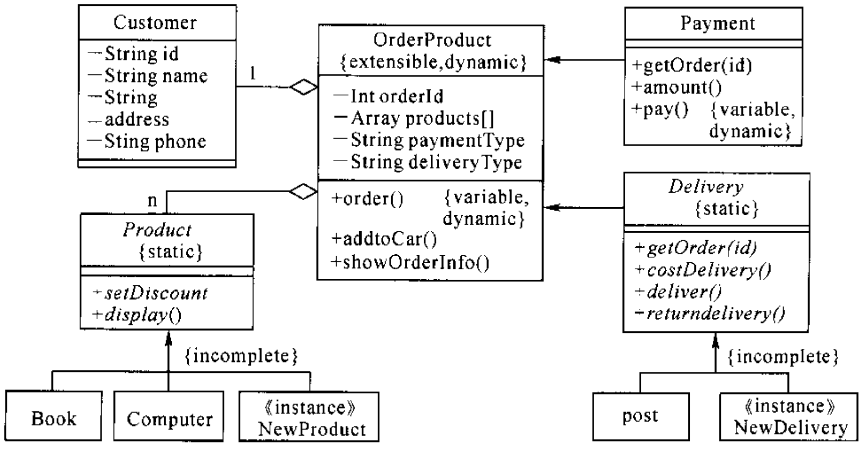

Fig. 7 Extended class diagram for EC domain framework

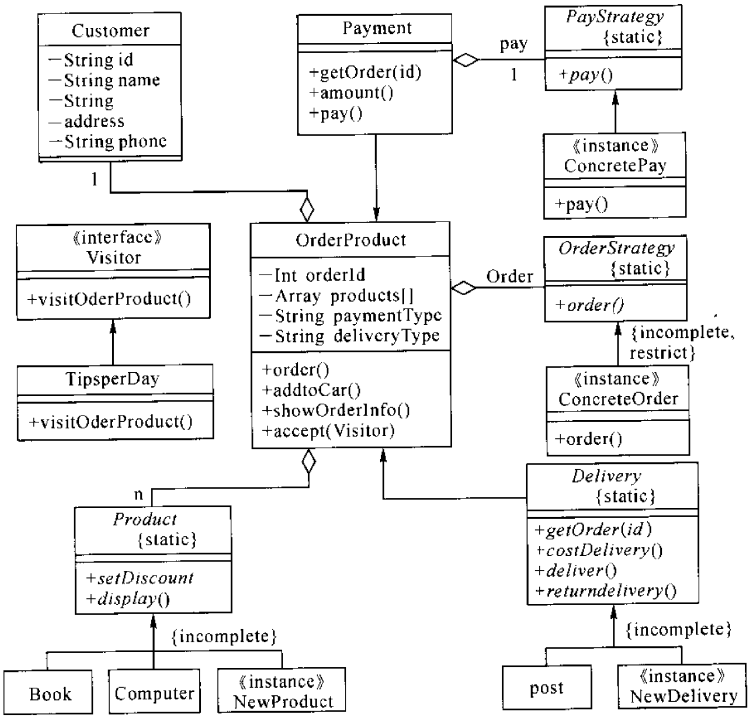

Fig.8 Implementation diagram for EC domain framework

addition, a transformation model based on design patterns is put forward to bridge the design-implementation gap for the new elements. Consequently, the UML profile makes the framework modeling more straightforward and easier to understand and instantiate.

Computer-aided tools based on UML and the profile may be developed to assist framework design, implementation and instantiation, and thus make the framework development and adaptation more systematical and efficient.

References

Booch, G., Rumbaugh, J. and Jacobson, I., 1999. The Unified Modeling Language User Guide. Addison Wesley, Boston, USA.

Coplien, J., 1999. Multi-Paradigm Design for C++. Addison-Wesley.

Deutsch, L.P, 1989. Design Reuse and Frameworks in the Smalltalk-80 System. *In*: Biggerstaff T. J. and Perlis A. J. editors, Software Reusability, Vol. 2, ACM Press.

D'Souza, D., Sane, A. and Birchenough, A., 1999. First Class Extensibility for UML – Packaging of Pro les, Stereotypes, Patterns. Proceedings of the Second International Conference on the Unified Modeling Language (UML), LNCS1723, Springer-Verlag, p. 265 – 277.

Fayad, M. and Schmidt, D. C., 1997. Object-oriented application frameworks. *Communications of the ACM*, **40** (10): 32 – 38.

Gamma, E., Helm, R., Johnson, R. and Vlissides J., 1995. Design Patterns - Elements of Reusable Object Oriented Software. Addison Wesley, Boston, USA.

Hakala, M., Hautamaki, J., Koskimies, K., Paakki, J., Viljamaa, A. and Viljamma, J., 2001. Annotating Reusable Software Architectures with Specialization Patterns. IEEE/IFIP Conference on Software Architecture (WICSA'01), Amsterdam, Netherlands, p. 171 – 180.

Johnson, R.E. and Foote, B., 1988. Designing reusable classes. *Journal of Object-Oriented Programming*, **1** (2).

Markiewicz, M. E. and Lucena, C. J.P., 2001. Object oriented framework development. *ACM Crossroads Student Magazine*, **7**(4):22 – 35.

OMG, 2002. OMG Unified Modeling Language Specification (Action Semantics) V1.4. http://www.omg.org.

Pree, W., 1994. Meta Patterns - A Means for Capturing the Essential of Reusable Object-Oriented Design. Proceedings ECOOP'94.

Pree, W., 1995. Design Patterns for Object-Oriented Software Development. Wokingham: Addison-Wesley/ACM Press.