Component Configuration Management for Frameworks

Ivica Crnkovic[1], Magnus Larsson[2], Kung-Kiu Lau[3*]

[1]Mälardalen University, Department of Computer Engineering, 721 23 Västerås, Sweden, Ivica.Crnkovic@mdh.se
[2]ABB Automation Products AB, LAB, 721 59 Västerås, Sweden, Magnus.Larsson@mdh.se
[3]University of Manchester, Department of Computer Science, Manchester M13 9PL, United Kingdom, kung-kiu@cs.man.ac.uk

Abstract: Object-oriented Design frameworks are increasingly recognized as better components than objects. In this paper, we briefly explain the framework concept, show a COM implementation, and discuss the accompanying configuration management issues.

Keywords: Components, configuration management, frameworks, COM, objects, CBD, CBSE

1 Introduction

Object-oriented Design (OOD) frameworks are increasingly recognized as better components in software development than objects (see e.g. [4] and [7]). The reason for this is that in practical systems, objects tend to have more than one role in more than one context, and OOD frameworks can capture this, whereas existing OOD methods (e.g. Fusion [1] and Syntropy [2]) cannot. The latter use classes or objects as the basic unit of design or reuse, and are based on the traditional view of an object, as shown in Figure 1, which regards an object as a closed entity with one fixed role.

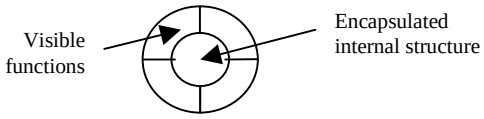

Figure 1. Objects with one fixed role.

On the other hand, frameworks allow objects that play different roles in different frameworks to be composed by composing frameworks. In *Catalysis* [3], for instance, this is depicted in Figure 2.

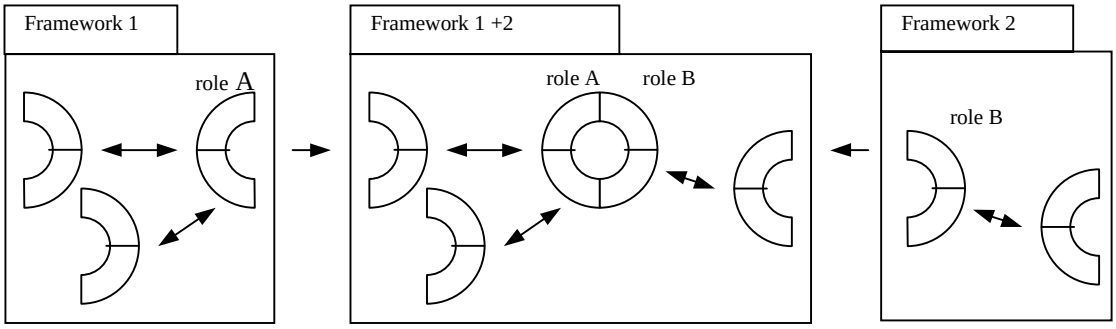

Figure 2. Objects with multiple roles in different frameworks.

In this paper we discuss a possible COM implementation of framework and the accompanying configuration management (CM) issues.

---

* This work was carried out while the third author was a visiting professor at Mälardalen University. He wishes to thank Professors Hans Hansson, Bjorn Lisper and Ivica Crnkovic for their invitation and their hospitality.

2 Frameworks: An Example

The following example illustrates the framework concept. Consider the framework for employees as depicted in Figure 3, in which a person plays the role of an employee of a company.

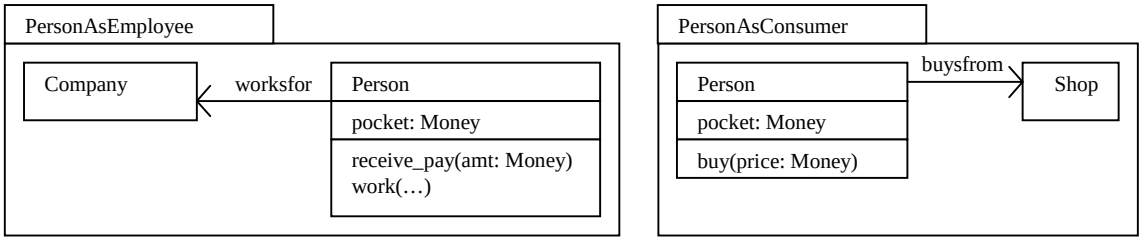

Figure 3. PersonAsEmployee and PersonAsConsumer framework.

A person as an employee has an attribute *pocket* representing the amount of money he possesses, and two actions *receive_pay* and *work*. Now consider another view of a person, e.g., a person plays the role of a consumer, as shown in the PersonAsConsumer framework in Figure 3. In this role a person also has the attribute *pocket*, but he has the action *buy* (instead of the actions *receive_pay* and *work*). We may compose the frameworks for PersonAsEmployee and PersonAsConsumer, to obtain a person with both roles together. A person now has all the actions of both roles, namely *receive_pay*, *work* and *buy*, and the attribute *pocket* in both roles. The composition is illustrated by Figure 4.

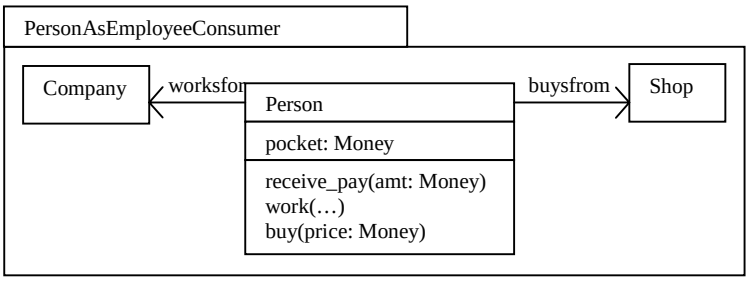

Figure 4. PersonAsEmployeeConsumer framework.

3 A COM Implementation of Frameworks

We illustrate the frameworks in Figure 3 and Figure 4 with an implementation example using COM [8]. COM suits multiple roles because it can use multiple interfaces for each role. We will use the aggregation mechanism in COM to compose frameworks. First, we implement the *Person* object, which corresponds to the encapsulated internal structure in Figure 1. The *Person* object is constructed so it supports aggregation of role objects and it has one *IPerson* interface (see Figure 5).

Figure 5. A COM object for the person object and the consumer role.

Second, the consumer and employee roles are implemented so they support being aggregated into a person object. Figure 5 shows the consumer role with one *IConsumer* interface. The consumer object needs also a reference to the person object to be able to work on the *pocket* variable. The person reference is set up when the consumer is aggregated into the person object (see Figure 6). In a similar way the *employee* role is implemented. Using aggregation we can reuse the different components that we have created. Figure 6 shows how *Person* aggregates the two already defined COM objects. Frameworks are created at run-time by adding roles to an object.

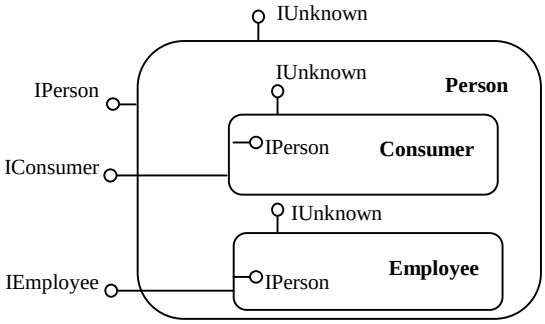

Figure 6. The Consumer and Employee roles are aggregated into the Person object.

The COM implementation of the framework concept has some limitations. The COM model defines frameworks as aggregates of the completed objects created at run-time, while a general framework model allows us to use incomplete objects (at run-time) or classes (at build-time).

4 Configuration Management Issues

Using frameworks instead of pure objects gives several advantages, but it also introduces an additional level of complexity when building them. Frameworks are composite types of entities – they have an internal structure which is built from objects, or from parts of them. A framework entity also has relations to other frameworks, and can be composed from other (sub)frameworks. The definition and creation of such a composite entity introduces configuration problems. Some of them will be illustrated here for a COM implementation.

Let us consider the following examples:

- Sharing objects in several frameworks;
- Composing frameworks from objects and frameworks.

4.1 Sharing objects in several frameworks

Suppose framework F_1 includes objects O_1 and O_2 with a relation R_{12} between them, and framework F_2 contains objects O_1 and O_3 with a relation R_{13}. The object O_1 is shared by two frameworks:

$$F_1 = \{O_1\ O_2\ ;\ R_{12}\},\quad F_2 = \{O_1\ O_3\ ;\ R_{13}\} \tag{1}$$

Suppose we now add a new property to the object O_1, a property that is required in (an improved version of) framework F_2. This creates a new version of the object $O_{1;v2}$, (v2 denotes the new version) which is included into the framework F_2:

$$F_2 = \{O_{1;v2}\ O_3\ ;R_{13}\ \} \tag{2}$$

However, if we do not take versioning into consideration, then the framework specifications will remain the same. In this case, we can be aware of the change of the object O_1 in the context of framework F_2, but not necessary in that of F_1. Our specification of F_1 is defined by (1), but in reality we have

$$F_1 = \{O_{1;v2}\ O_2\ ;R_{12}\ \} \tag{3}$$

If the role of the object $O_{1;v2}$ used in F_1 is changed, then the behaviour of F_1 will be changed unpredictably, and a system using F_1 can fail.

To avoid these unpredictable situations we can introduce basic configuration management methods – a version management of objects and configuration of frameworks [9]:

- An object is identified by its name and version.
- A framework is identified by a name and a version. A new framework version is derived from object versions included in the framework.

These rules imply that new versions of frameworks will be configured when a new object version is created, as shown in our example:

$$F_{1;vi}=\{O_{1;vm}\ O_{2;vn}\ ;\ R_{12}\ \},\ F_{2;vk}=\{O_{1;vm}\ O_{3;vk}\ ;\ R_{13}\ \} \tag{4}$$

$$F_{1;vi+1}=\{O_{1;vm+1}\ O_{2;vn}\ ;\ R_{12}\ \},\ F_{2;vk+1}=\{O_{1;vm+1}\ O_{3;vk}\ ;\ R_{13}\ \} \tag{5}$$

As several frameworks can share one object, and a framework can contain several objects, the number of generated frameworks can grow explosively. It is, however, possible to limit the number of interesting configurations. Typically, in a development process, we would implement the changes on all the objects we want, collect the versions of objects we want in a *baseline* and derive the frameworks from the baselined object versions. In such a case, experience for similar cases [10] shows that the number of derived entities does not necessarily grow rapidly.

A shared object is not necessarily completely shared, but different parts of the object, defined by the object's roles, are used in the frameworks. In the COM implementation a complete object will be included, but a part of it will be used. In a general framework model, a class (or an object at run-time) includes only those parts which are specified in the object's role.

When we define a new role for an object in a framework or re-define the existing one, we need to change a specific part of the object class. We call this specific part an object *aspect*. The change of an object aspect will affect only those frameworks where the aspect is included. Other frameworks, though containing the object (or part of it), are not affected by the change. In this case, it is better to keep version control on the aspect level, and relate a framework configuration to the object aspects.

If we declare an aspect as a subset of an object $A_i\ (O_k) \subseteq O_k$, then an object version is defined as a set of aspect versions:

$$O_{i;vk} = \{\ A_{j;vl}\ \} \tag{6}$$

and a framework version is defined as a set of aspect versions with relations between the aspects:

$$F_{vk} = \{\ \{A_{j;vl}\ (O_{i;vk})\}\ ;\ R_{jl}\ \} \tag{7}$$

Having control over changes on the aspect level, we can gain control over the changes on the framework level. Now we can more precisely identify the frameworks being affected by changes in object roles.

4.2 Composing frameworks from objects and frameworks

In the framework model it is possible to compose new frameworks from existing frameworks. A new framework is a superset of the classes and relations from the frameworks involved. If a new framework is created at run-time, as in a COM implementation, then the objects from the selected frameworks comprise the new framework.

The following example illustrates the merging process of two frameworks F_1 and F_2 into F_3:

$$F_1 = \{O_1\ O_2\ ;\ R_{12}\ \},\quad F_2 = \{O_1\ O_3\ ;\ R_{13}\ \},\quad F_3 = \{O_1\ O_2\ O_3\ ;R_{12}\ ,\ R_{13},\ R_{23}\} \tag{8}$$

The composition works fine as long as we do not need to consider the changes of objects within one framework.

Suppose we create a new object version (or a new object aspect version) in F_2 and keep the old version of the same object in $F_{1:}$

$$F_{1;vi} = \{O_{1;v1}\ \ O_{2;vk}\ ;\ R_{12}\ \},\quad F_{2;vj} = \{O_{1;v1+1}\ \ O_{3;vl}\ ;\ R_{13}\ \} \tag{9}$$

In the merging process we have to recognise if different versions of the same objects are included in the frameworks being merged. If that is the case, we have two possible solutions:

- Selecting one specific version of the object (for example the latest):

$$F_{3;v1} = \{\ O_{1;v1+1}\ \ O_{2;vk}\ O_{3\ ;v1}\ ;R_{12\ ,}\ R_{13,}\ R_{23}\} \tag{10}$$

- Selecting both versions and enable their consistence in the new framework:

$$F_{3;v1} = \{\ O_{1;v1}\ ,\ O_{1;v1+1}\ ,\ O_{2;vk}\ O_{3\ ;v1}\ ;R_{12\ ,}\ R_{13,}\ R_{23}\} \tag{11}$$

For the second case there must be support for identifying object versions. This support can be provided by introducing an identification interface [12] as the standard interface of an object. There must also be support for managing different versions of the same object in the running system.

5 Discussion

The framework approach gives a better possibility to reuse components in composing systems. A discussion on formal description of frameworks can be found in [5] and [6]. In this paper we have presented a possible implementation of the framework model using the COM technology. This implementation shows some limitations. Further investigation on how to improve the implementation should be carried out.

The second topic discussed in this paper is configuration management for frameworks. The paper emphasises a need for using CM methods for managing frameworks as composite objects. The CM issues are complicated and need further investigations: Questions of managing relations, concurrent versions of frameworks, inclusion of change management [11], etc., must be addressed. Since aspects and objects are not entities recognised as CM-items by standard CM tools (which recognise entities such as files, directories, etc.), new, semantic-based rules must be incorporated into the CM tools. For different OO and component-technologies, different tools have to be made. How different do they need to be, and are there possibilities to define common rules and implementation? These are questions for future investigation.

6 References

[1] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes: Object-Oriented Development: The Fusion Method, Prentice-Hall, 1994.

[2] S. Cook and J. Daniels: Designing Object Systems, Prentice-Hall, 1994.

[3] D.F. D'Souza and A.C. Wills: *Objects*, Components, and Frameworks with UML: The Catalysis Approach, Addison-Wesley, 1998.

[4] R. Helm, I.M. Holland, and D. Gangopadhay: Contracts - Specifying behavioral compositions in OO systems, Sigplan Notices 25(10) (Proc. ECOOP/OOPSLA 90).

[5] K.-K. Lau, S. Liu, M. Ornaghi, and A. Wills: Interacting frameworks in Catalysis. In J. Staples, M. Hinchey and S. Liu, editors, Proc.Second IEEE Int. Conf. on Formal Engineering Methods, pages 110-119, IEEE Computer Society Press, 1998.

[6] K.-K. Lau, M. Ornaghi, and A. Wills: Frameworks in catalysis: Pictorial notation and formal semantics, In M. Hinchey and S. Liu, editors, Proc. 1st IEEE Int. Conf. on Formal Engineering Methods, pages 213-220, IEEE Computer Society Press, 1997.

[7] R. Mauth: A better foundation: development frameworks let you build an application with reusable objects. BYTE 21(9):40IS 10-13, September 1996.

[8] D. Box, Essential COM, Addison-Wesely, 1998

[9] R. Conradi, B. Westfechtel, Version Models for Software Configuration Management, ACM Computing Surveys, Vol. 30, No.2, June 1998

[10] U. Asklund, L. Bendix, H.B. Cristensen, B. Magnusson - The Unified Extensional Versioning Model, System Configuration Managemnt SCM-9, Springer, 1999

[11] I. Crnkovic, Experience with Change Oriented SCM Tools, Software Configuration Management SCM-7, Springer, 1997

[12] M. Larsson, I. Crnkovic, New Challenges for Configuration Management, System Configuration Management SCM-9, Springer, 1999