From a UML Platform Independent Component Model to Platform Specific Component Models

Tewfik Ziadi [1], Bruno Traverson [2], Jean-Marc Jézéquel [1]

[1] IRISA-INRIA /Rennes1 University, Campus de Beaulieu, F-35042 Rennes France
[2] EDF Division R&D, 1 avenue du Général De Gaulle, F-92141 Clamart France

Tewfik.Ziadi@irisa.fr, Bruno.Traverson@der.edf.fr, Jean-Marc.Jezequel@irisa.fr

Abstract. In this paper, we propose a set of UML extensions to describe a platform independent model for software components. We use UML extensions to define software component concepts (components, ports and connectors). Architectural constraints are specified as OCL meta-level constraints and mapping rules are defined to map our platform independent model to platform specific models. In particular, our model supports component port adaptation by using conversion connectors.

1 Introduction

A software component is a software unit for a third-party composition. The composition is realized through the set of specified interfaces. It depends on a specific context and it can be deployed autonomously [1].

The standardization of software components consists in defining specific component models and architectures supporting these models. EJB (Enterprise Java Beans), CCM (CORBA Component Model) and DotNet are examples of architectures that define and implement a specific component model.

This paper proposes the use of UML (Unified Modeling Language) to define a Platform Independent Component Model, which enables the description of software architectures independently of the implementation choices. The Platform Independent Component Model describes key concepts related to a software architecture description such as: *Components*, *Ports* and *Connectors*. In particular, we focus on the specific problem of component port adaptation and we propose to use connectors to solve it.

The paper is organized as follows: Section 2 describes the future component model in UML 2.0 and it presents UML extension mechanisms in UML 1.4, section 3 proposes our Platform Independent Component Model, section 4 defines mapping rules to specific platforms and section 5 concludes this paper.

2 Background in UML

This section recaps some present and future features in UML that are related to our work: component model in UML 2.0 and extension mechanisms in UML 1.4.

Component Model in UML 2.0

UML 2.0 will provide new concepts and major improvements in the support of component-based design. In [2], we find a submission including a set of constructs about components and their assembly. A component is defined in the meta-model as a specialization of classifier. Its description may include a set of *ports*, a set of *parts* , a set of *connectors* and a behavior. A *Port* is a named interface on a component, it defines a set of operations and events that are *provided* by a component or that are *required* from its environment. A part represents a sub-component. A connector defines a relationship between two ports. We find two types of connectors: The Delegation Connector and the Assembly Connector. The Delegation Connector represents the forwarding of messages between a port of a component and a port of one of its part. The Assembly Connector must only exist between a provided Port and a required one. The component behavior may be defined with a *state machine*.

Extension mechanisms in UML 1.4

Currently, UML1.4 component model [3] is not rich and the UML 2.0 is not a stable basis. Also, UML tools do not support it yet. So, we have decided to use UML extension mechanisms to define a Platform Independent Component Model. In this sub-section, we briefly present UML extension mechanisms in UML1.4.
UML extension mechanisms are presented as a part of the UML specification [UML1.4] pp 2-74. It includes: (1) A **Stereotype** is a means to classify model elements. Stereotypes allows UML automatic tools to identify a specific element among the other model elements of the same type and process them specifically. (2) A **Tag definition** specifies the tagged values that can be attached to a kind of model elements. (3) A **Constraint** allows users to refine and specify new semantics for any model element. Constraint attached to a stereotype must be observed by all model elements branded by the stereotype. OCL (Object Constraint Language) [4] may be used to describe the constraints in UML.
A specific package, called a **UML profile,** can gather a set of extensions in the same structure. We find, for example, a UML profile for CORBA [1] which constitutes a customized version of UML for CORBA (Common Object Request Broker Architecture) based systems.

[1] http://www.omg.org/technology/documents/formal/profile_corba.htm

3 Our UML Platform Independent Component Model

To illustrate our model, we use an ad-hoc example in the banking area. We distinguish two components: **Bank** and **Account**. The Account component provides to its environment three operations: *deposit()*, *withdraw()*, and *getBalance()*. The Bank component provides a *transfer()* operation and requires *depositAccount()* and *withdrawAccount()* operations to realize the transfer operation, see Fig.1.

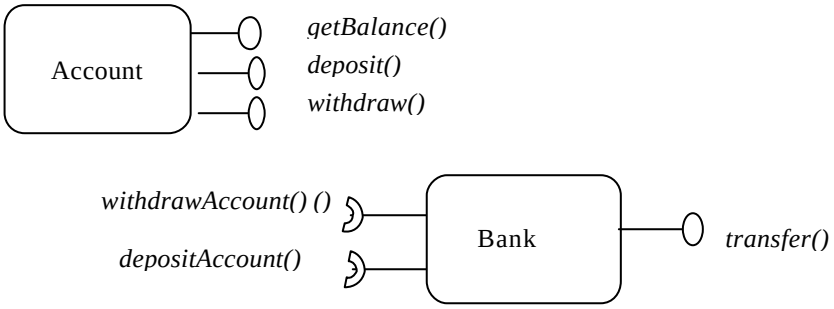

Fig. 1. Bank application components

Definition and Specification

1. **Components and Ports.** We specify a component as a UML class stereotyped <<component>>. A component defines a set of ports. A port is a connection point defined by its category. The basic type of ports is the operation port, which define an operation (method) provided or required by a component. In our model, the operation port is specified as a UML class stereotyped <<operationPort>>. It includes only one UML operation that describe the port structure. Ports are connected to components by a UML dependency stereotyped <<provided>> or <<required>>. For our bank example, the UML specification of components is illustrated in Fig.2.

? **Connectors and Assembly.** The specification of the component assembly consists in defining relationships between component ports. The notion of *required* and *provided* ports allows this connection between two ports. We define a connector as the key element in the assembly activity. It may also be used to adapt incompatible ports. In our model, we specify connectors as a UML dependency between a provided port and a required one with the stereotype <<**connect**_type of ports>>. For operation ports, we use a <<connectOperation>> stereotype, see Fig.2.

? **Ports Adaptation.** As already stated, connectors may be used to adapt two ports. For example, in our Bank model, the provided operation *deposit(amount)* of the Account component is functionally required by the Bank component (*depositAccount(amount, account_no)* port) but the assembly needs some adaptation between these ports. We propose to add a tagged value {adapt} to the

connector (a dependency) when the adaptation of ports is needed, see Fig.2. The adaptation is realized at the mapping time. We will see in the following paragraphs how the adaptation is realized in the EJB and CCM context.

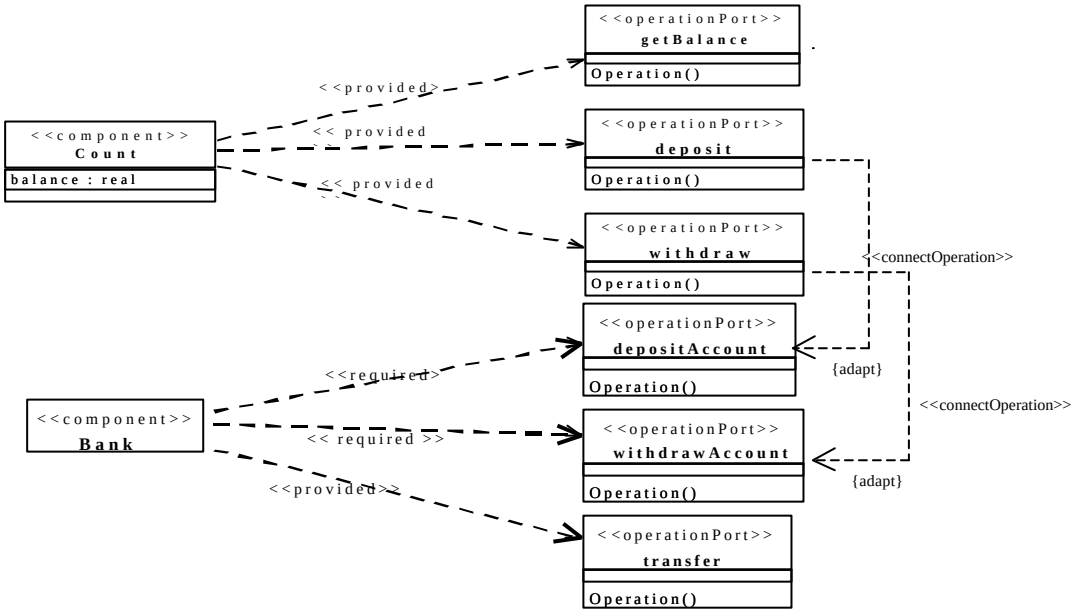

Fig. 2. Banking example: UML Specification of Components, Ports and Connectors

Architectural Constraints

We can add to our Component Model a set of architectural constraints that represent structural rules to any model based on these extensions. We specify architectural constraints as OCL meta- level constraints. For example, to show that the operation connector (a dependency stereotyped <<connectOperation>>) should only connect operation ports (classes stereotyped <<operationPort>>), we add the following OCL meta-model constraint as a UML Dependency invariant:

```
context Dependency

inv  self.isStereotyped[2](``connectOperation'') implies   ((
self.supplier → forAll(S:ModelElement | S.isStereotyped(``
operationPort'')) and       (self.client →    forAll(C:ModelElement |
C.isStereotyped(`` operationPort'')))
```

[2] *isStereotyped()* is our own high-level OCL operation.

4 The Mapping to Platfom Specific Component Models

In the context of component platforms, we find standards such as EJB (Enterprise Java Beans) [5] and CCM (CORBA Component Model) [6]. We have implemented our UML Independent Component Model as a UML profile in the Objecteering UML tool [3] (Profile Builder part of Objecteering) which allows implementing UML profiles [7] as modules that could be integrated in the Modeler part of Objecteering. We have also implemented two modules: the *EJB Projection* and the *CCM Projection*.

EJB (Enterprise Java Beans) Mapping

We have defined some mapping rules to generate from our Platform Independent Component Model some EJB Skeletons of classes. At this stage of the work, the generation consists only in Session beans, see Fig.3:

- Each Component (UML class stereotyped <<component>>) will be implemented as an EJB bean with two classes: the remote interface, and implementation class.
- Each provided operation port (UML class stereotyped <<operationPort>>) will be projected as a method in the **remote interface** of its component (bean).
- To realize the adaptation between two incompatible ports, we create a specific EJB bean which provides, exactly, the operation required and the implementation of this operation is realized by the delegation to the operation provided, see Fig.3.

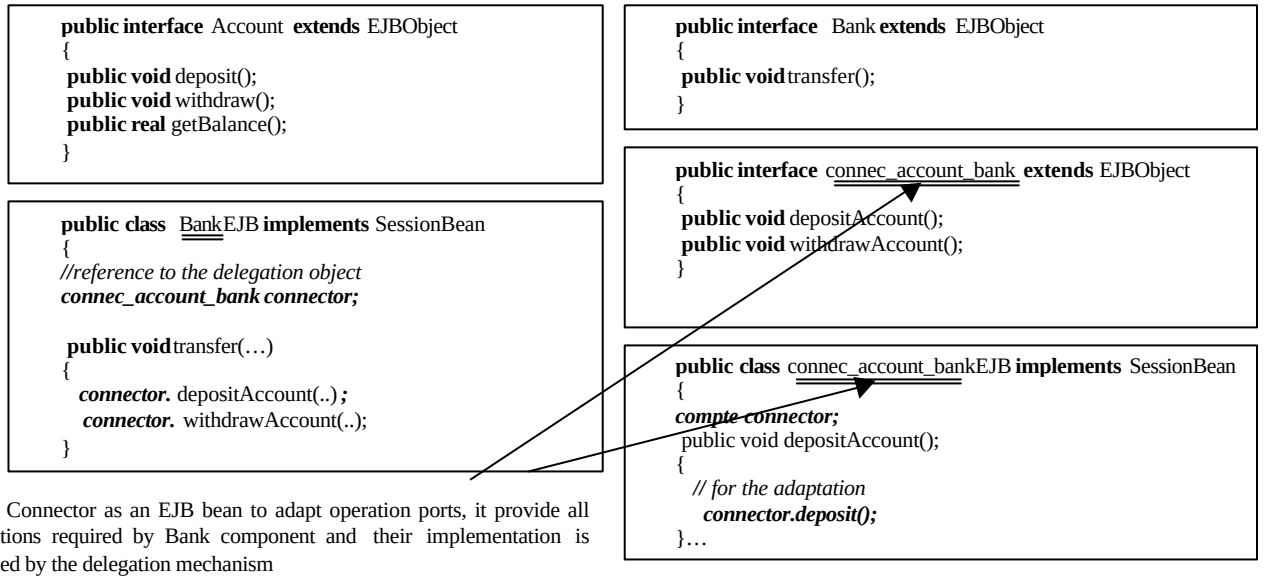

Fig. 3. Example of classes skeletons generated by the EJB Projection module

[3] www.objecteering.com

CCM (CORBA Component Model) Mapping

The CCM is the OMG standard for component platforms [6]. It defines a container for the specification and the deployment of component-based applications. It extends the standard OMG IDL (Interface Definition Language) with new constructs to support component specifications. A CCM component defines a set of attributes, a set of provided ports, a set of required ports, published events and the received ones.

We have defined some mapping rules from our Platform Independent Component Model to the standard CCM. In this stage of the work, we focused on the generation of the IDL 3 (Extended OMG IDL) file. Some mapping rules are defined to generate IDL3 file from the UML specification, see Fig.4 :

- Each Component (UML class stereotyped <<component>>) will be projected as a CCM component (with the key word **component)**.
- We gather component operation ports (*provided* or *required*) and we will project them as CCM interfaces (with the key word **interface)**.
- The provided (respectively required) operation ports will be declared as a provided interface with the key word **provided** (respectively a required interface with the key word **uses**).
- To realize the adaptation between two incompatible ports, we generate a specific CCM component, which provides operations required by the first component and uses operations provided by the second one.

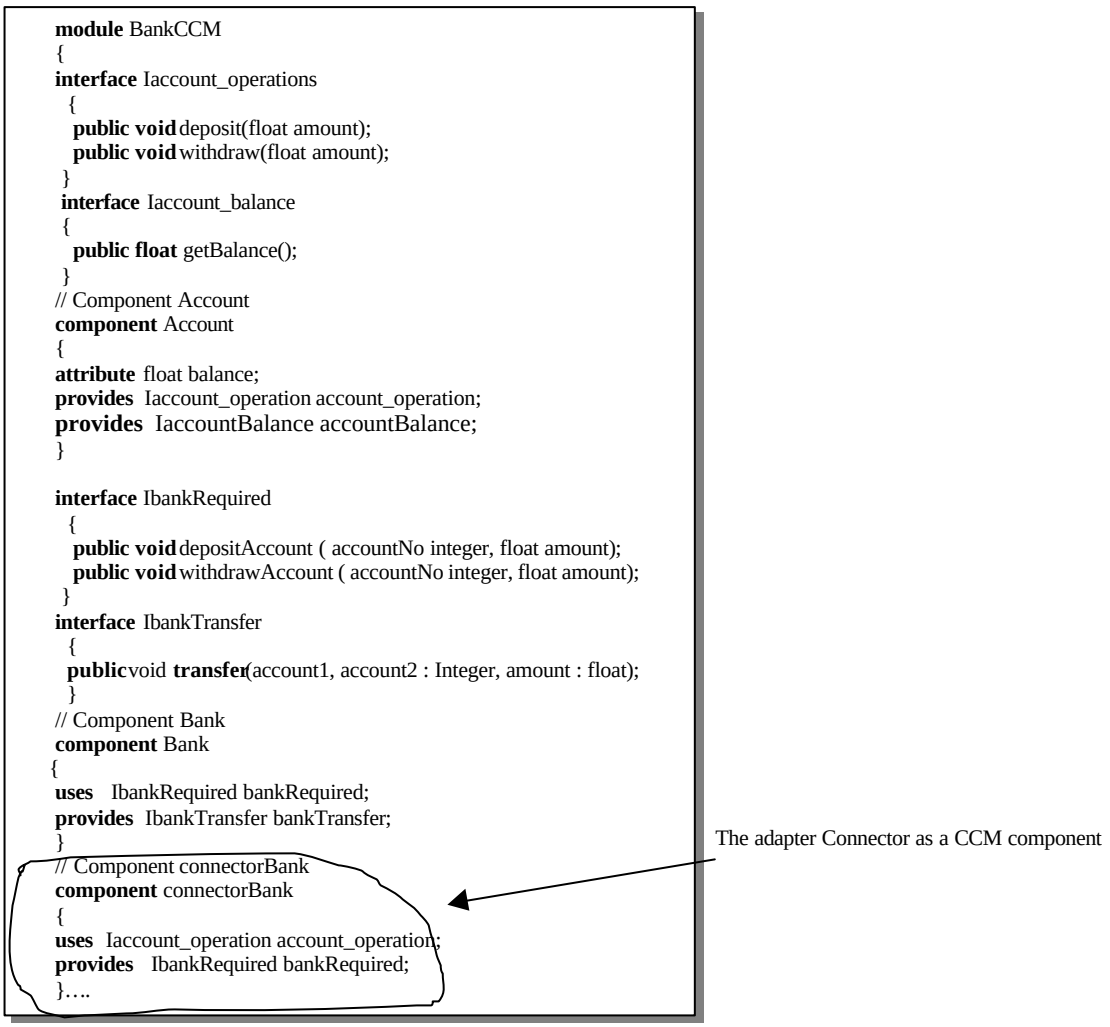

Fig. 4. Example of the IDL 3 file generated for the CORBA Components Model

5 Conclusion and Perspectives

We have presented in this paper a Platform Independent Component Model based on UML extensions. Our solution conforms to the MDA (Model Driven Architecture) approach [8]: the Platform Independent Component Model may be considered as the PIM (Platform Independent Model) and the EJB component Model or the CORBA Component Model as examples of PSMs (Platform Specific Models).

We have implemented our solution as a UML profile in the Objecteering tool. Objecteering profile builder allows us to define a set of UML extensions and to implement methods (in J language) to customize the mapping process. However, Objecteering does not support OCL meta-level constraint definitions. So, the solution to specify architectural constraints is to translate them as J checker methods.

Our mapping process consists in generating source class skeletons and IDL files. It may be extended to generate UML Models for specific platforms based on the UML profile for EJB and the UML profile for CCM (work under progress in a national project).

Then, the component port adaptation is handled in the mapping process. So, it can adapt to specific model targets. We think that it is important to abstract this adaptation capability and define it at the Independent Model level.

Finally, we have concentrated our effort on operation ports; this work may be extended to support other types of ports and their connectors.

References

[1] Szyperski C. *Component Software –Beyond Object-Oriented Proagramming*. Addison-Wesley, 1998.

[2] U2 Partners. *Proposal for UML 2.0 Infrastructure and Superstructure.* ad/00-09-01 and ad/00-09-02. http://www.u2-partners.org/artifacts.htm

.

[3] Object Management Group. *UML Specification, 1.4*. (2001).

[4] Warmer, J., and Kleppe, A. 1998. *The Object Constraint Language – Precise Modeling with UML*. Object Technology Series. Addison-Wesley.

[5] Enterprise Java Bean technology.
http://java.sun.com/products/ejb/

[6] Object Management Group. Document **ccm/02-04-01** (*CORBA Component Model Tutorial*). http://www.omg.org/cgi-bin/doc?ccm/2002-04-01 .

.

[7] White paper Softeam 99. *UML profile and the J language*. Softeam 99.
http://www.objecteering.com/pdf/us/uml_profiles.pdf

[8] Desmond Dsouza. *Model Driven Architecture: Opportunities and Challenges*., 2001. ftp://ftp.omg.org/pub/docs/ab/01-03-02.pdf