The new UML 2.0 Component Model: Critical View

Jean-Michel Bruel
LIUPPA, University of Pau
64013 Pau CEDEX, France
Jean-Michel.Bruel@univ-pau.fr

Ileana Ober
VERIMAG
Grenoble, France
ileana.ober@imag.fr

1 Introduction

At the time we write this paper, when you type "Component Modeling Language" in a web-search engine, not only you do not find so many links, but you find among the first ones a link to the Rational Rose web page. Looking through this page, and also through the recently adopted U2-partners proposal [4] for the upcoming 2.0 version of the UML, it seems that UML 2.0 aims to be, among other things, a Component Modeling Language (CML). It is not the aim of this short article to fully explore the requirements for a potential CML (see [3] or [1] for ongoing researches on this topic), but indeed, most people do not use UML in its current status (1.x [5]) for modeling components. It is true that the current version of the standard is very limited in terms of component modeling abilities, as it basically supported only physical components. In this paper we are going to give an overview of the UML new support for components, highlighting mainly the improvements and new concepts recently adopted.

2 UML 2.0 component model: critical view

Reusability and architecture representation have been the key concerns in the definition of the new UML version. The component model itself has been improved, the overall concept of composition has been integrated, and even at the class level. For details on this section, see the full specification [4].

2.1 Overview

UML 2.0 provides support for decomposition through the new notion of **structured classifiers** (in this paper we use **verdana bold** font to highlight the new UML 2.0 concepts). A **structured classifiers** is something that can be internally decomposed (Classes, Collaboration, and Components). In addition, some new constructs to support decomposition have been introduced: **Part**, **Connectors**, and **Ports**. It supports the specification of *physical* components such as in UML 1.X (e.g., EJB, CORBA, COM+ and .NET components), but also *logical* components (e.g., business components, process components) as well as *deployed* components (such as artifacts and nodes). A component is viewed as a "*self contained unit that encapsulates state + behavior of a set of classifiers*". It may have its own behavior specification and specifies a contract of provided/required services, through the definition of ports. It is hence a substitutable unit that can be replaced as long as port definitions do not change. Notice that the notion of "change" here is not defined and has to be taken at a syntactical level only.

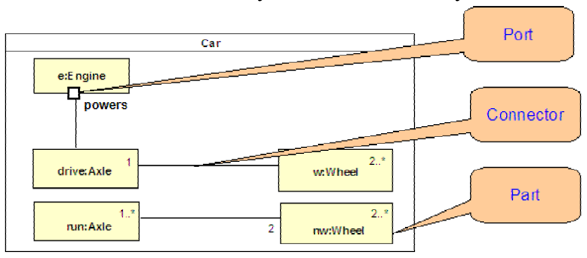

Fig. 1 – Example of a composite structure

2.2 New constructs

Three new constructs are part of the component model. Note that those constructs can be used together with any composite diagram. These new concepts are:

- **Part**: something that is internal to a composite structure. Notice that, as illustrated in the Fig. 1, instances (of a class) and parts have similar notations. Parts have to be seen as roles, and instance are the realization that satisfy these roles.
- **Connector**: expresses the relationship between parts. It is a link (an instance of *association*) that enables communication between two or more instances. It may be realized by pointer, network connection, etc.
- **Port**: the connection point via which messages are sent to/received by a class (or a component). Ports have type which is given by a set of interfaces (provided and required), and can be described with a state machine.

The notion of **interface** has not changed, it represents a signature given in terms of a set of public features (operations, attributes, signals). However the interface use was extended, a classifier may *implement* or *require* an interface. The graphical notation of a classifier with interface has evolved to cover this extension. In Fig. 2, we have a component with a required (NetworkServices) and an implemented interface (EstablishCall).

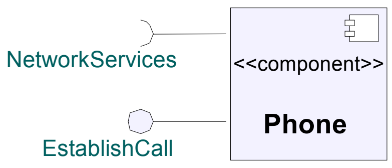

Fig. 2 – Notation for component with interfaces

Interfaces can be attached to ports. A **required interface** attached to a port characterizes the behavioral features that the owning classifier expects from its environment *via the given port*, while a **provided interface** attached to a port characterizes the behavioral features offered by the owning classifier via the given port. Note the distinction between a port and an interface: an *interface* specifies a service offered/required by a classifier, while a *port* specifies the services offered/ required by the classifier *via that particular interaction point (port)*. It is possible to attach to a port or to an interface, a **protocol state machine** that allows the definition of a more precise external view by making dynamic constraints on the sequence of operation calls and signal exchanges explicit. The protocol state machine of a port (if present) shall be compatible with the protocol state machines of all interfaces attached to it. *However,* this "compatibility" is not define in the proposal. In addition, there is no evidence nor heuristic in the proposal on the difference of attaching the protocol state machine to an interface or to a port.

2.3 Component Diagrams

In UML 1.x, component diagrams offer support for physical components only. Now, not only they integrate logical components, but they also support component-based software engineering (CBSE). Components models can be viewed under two views:

- An external one, corresponding to the usual "black box" view, describing the contracts of the component with its environment in terms of provided and required services;
- An internal view, hidden from the clients, corresponding to the classical "white box" view describing how the component is organized in terms of parts, sub-components, connectors, etc.

There is two specific connectors for components:

- an **assembly connector** is the link between a required (socket) and a provided (ball) interfaces. It is one way to wire components together (the other way is to use dependencies as in UML 1.x);
- a **delegation connector** is the same idea but from an internal point of view: an arrow indicate the delegation direction.

2.4 Support for composition

New constructs and new approaches have been introduced in UML 2.0 with direct impact on the support for composition. To describe the links between a composite and its sub-components, UML 2.0 uses the notion of **nested components**. As we have shown, the internal structure of a component can be described by a component diagram. In fact, despite the added/modified notations for components constructs, the main change, or the one that is going to have the more impact on the ongoing researches (such as ours [2]) is the introduction of the *CompositeStructures* first class element, and at a lower level the new distinction of *required interface*. Components communicate together via messages going through their ports, using the same idea as processes in SDL or capsules in ROOM. Note that components can also communicate directly point to point, using the same idea as in usual component models such as CCM.

3 Conclusion

We have presented in this short paper the improved notation of the new UML 2.0 specification that could benefit to CBSE. Going briefly through the introduced concepts and changes, we have highlighted that it was mainly a change at the syntactical level only. These improvements, already existing in some approach will be useful for component modeling as long as this modeling is used for documentation only. As soon as some more rigorous development strengths will be needed (validation purposes, etc.) UML 2.0 users will still encounter the same lack of well-defined semantic problems, in which UML 2.0 has failed to bring them concrete answers.

4 References

[1] ARTIST, Project IST-2001-34820, Component based Design and Integration Platforms : Roadmap. Available at http://www.artist-embedded.org.

[2] Nicolas Belloir, Jean-Michel Bruel and Franck Barbier, *Whole-Part Relationships for Software Component Combination*. Proceedings of the 29th Euromicro Conf. on CBSE Belek, Turkey, 2003. To be published.

[3] Jean-Michel Bruel, *CML – Component Modeling Language : un langage de composition pour composants logiciels*. Projet d'Action Concertée Incitative 2003.

[4] http://www.u2-partners.org/uml2-proposals.htm

[5] Object Management Group, UML Specification, version 1.4, http://www.omg.org, 2001.