STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume LI, Number 1, 2006

COMPONENTS MODELING IN UML 2

JEAN-MICHEL BRUEL AND ILEANA OBER

Abstract. The Object Management Group (OMG) has recently recommended for adoption the proposal that will be the basis for the new version of the Unified Modeling Language – UML. In this version 2.0 of UML, the component model has been completely modified. The first goal of this paper is to describe this new model. The second goal is to study if it is really more than a notation improvement. We go through the requirements for a general component modeling language, the existing efforts in this area, and the new UML 2.0 model itself.

1. Introduction

Nowadays, even the simplest software systems are complex. Generalization of the client/server architecture, importance of the notion of services, distributed in different nodes of the system, and distantly available, real-time and critical aspects more and more involved, are some of the reasons for such a complexity. As a matter of fact there is a big issue in the use of reusable pieces of software components. But when it comes with components and composition, there are two complementary problems to solve [12]: "[...] how to meet requirements using component-based designs and how to design components that work well together."

Components and composition have been an active research topic for a long time now. But it is undoubtable that there is no such a usage of software components as in hardware for example. One of the reason advocated by international surveys is the lack of support for composition description and evaluation [24]. Not so long ago, when you typed "Component Modeling Language" in a web-search engine, not only you did not find so many links, but you found 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 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 [23].

Received by the editors: June 10, 2006.

2000 *Mathematics Subject Classification.* 68N99.

1998 *CR Categories and Descriptors.* D2.2 [**Software Engineering**]: Design Tools and Techniques – *Object-oriented design methods.*

We do not aim here to fully explore the requirements for a potential component modeling language (see e.g., [19, 2] for such requirements), but we intend to give a critical overview of the new UML support for components using the following organization: in section 2 we will describe composition models and languages; in section 3 we will describe the UML 2.0 component model; in section 4, we will present ongoing efforts that aim at providing good support for composition; in section 5 we discuss this model from a critical point of view; and we will conclude in section 6.

2. Composition models and languages

Component based development offers a means to deal with the increasing software complexity. We do not start with the rather philosophical and surely controversial question on what it is a component. Instead we will focus on the main issues related to components.

According to various authors [18, 25] components are characterized by their *interfaces*, which specify their possible interaction with their environment and offer a basis for establishing the compatibility between two or several components. The *provided interfaces* specify how a component can be used and summarize what it offers to the outside world. Besides the provided interfaces, a component should specify what the deployed environment should provide in order for the component to be able to offer the functionality specified in the provided interface [20]. These context dependencies are specified using *required interfaces*.

For a good component specification, the structural specification of components done in terms of provided and required interfaces should be complemented with a *component contract* [6, 17]. Specifying required interfaces is a first step in this direction, however it only covers the structural/signature level. More advanced techniques for component specification include invariants, pre- and post-conditions of the services offered/required by the interfaces, various kinds of logics, automata-like specification techniques for component abstractions, etc. The nature of the component contract, and of the contract specification technique varies depending on whether the component is viewed as a unit with state (as argued in [18]), or a stateless unit (as argued in [25]).

The component contract can be specified at various levels of abstraction and can address various needs.

[6] defines a four level contract framework, where each level corresponds to a class of contracts (from the first to the forth): syntactic, behavioral – described in terms of pre- and post-conditions and invariants, synchronization, and quality of service. In this framework, contracts corresponding to the first two levels can be checked statically (provided that enough verification theoretical background exists), while the last two levels are basically focused on dynamic behavior and on the integration of the component on the execution platform.

One important and much studied issue in this context is the component *composition*: gathering together various components into a working system. Related to this, lots of efforts address the impact of composition on the *properties* of the composed components. A thorough analysis of the composition, formalizing it as an operation that considers the components and their integration constraints can be found in [14]. The paper also proposes a framework for component composition.

In what follows we present a list of expected features needed for a good components model. This list is established independently of UML, and will serve as as a basis for discussing the properties of the component model offered by UML.

Precise modeling concepts. Since the component modeling offers an abstract view of the system under modeling, it is crucial to have unambiguous definitions and a precise semantics of the basic modeling concepts.

Expressive power. For this the parts that need to be covered [15] are: (i) interface specification: it should be possible to associate sets of interfaces to a component specification and to specify their relationship with the component, i.e. whether they are required or offered; (ii) semantic specification: define and document semantic information about the component's operations; (iii) extra-functional properties specification: such as quality of service (QoS) and timing.

Modeling features. [9] defines several key requirements for a component model. Among the most general we mention: (i) encapsulation and identity; (ii) composition: the model should not only support dynamic composition, but also support different semantics of composition; (iii) life-cycle: a general model should support different forms of components throughout the entire development life-cycle.

3. UML 2.0 COMPONENT MODEL

Backwards compatibility and the possibility to include architecture representation into UML models 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, even at class level. We will first introduce the concepts and then discuss their pluses and minuses.

3.1. New concepts and improvements.

Justifying the drawbacks in the UML 1.5 version in terms of composition is out of the scope of this paper. For an introductory illustration of some major difficulties in using the UML 1.5 composition model, see [7]. UML 2.0 [23] provides support for decomposition through the new notion of structured classifiers[1]. A structured classifier is a classifier that can be internally decomposed (Classes, Collaboration, and Components). New constructs to support decomposition have been introduced: Part, Connectors, and Ports. Note that Part is a new name, but not a new concept. UML 2.0 allows the specification of

[1] In this paper, we use Sans Serif font to highlight the new UML 2.0 concepts.

physical components such as in UML 1.5, of logical components, i.e. specification level 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 syntactic level only. Notice also that ports are not reserved to components, but are available for any structured classifier.

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. This is not much different from UML 1.5 except that aggregation applies to properties as well as association ends. This is due to the unification of attributes and associations. Notice that instances (of a class) and parts have similar notations, which might be confusing. The part names are not objects identifier, but role names. Parts have to be seen as roles, and instances as the realization that satisfy these roles.

Connector: expresses the relationship between parts and between ports. It is a link (an instance of association) that enables communication between two or more instances, in addition to everything that ordinary links enable (e.g., navigating to a neighboring object to retrieve a property from it, modify it, destroy it, etc.). It may be realized by pointer, network connection, etc.

Port: a kind of part, but mainly used to represent the connection point via which messages are sent to/received by a component (or a class). Ports have a type which is given by a set of interfaces (provided and required) and can be described with a state machine. UML 2.0 has introduced a specific kind of state machines (that describes usage protocols of parts of the system): protocol state machines, and hence renamed the previous state machine (for describing the behavior of some entities): behavioral state machine.

The interface represents a signature given in terms of a set of public features (operations, attributes, signals). Interface attributes as well as association between interfaces is new in UML 2.0. The interface use has been extended from UML 1.5: a classifier or a port may *implement or require* an interface, in addition to providing an interface. 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" notion is not defined in the proposal.

3.2. **Component Diagrams.** In UML 1.5, component diagrams were support for physical components only. In UML 2.0, extends component diagrams from addressing physical components to logical ones. Also it allows component-based software engineering CBSE as it is now possible to trace from logical to physical components.

There is two possible views for components models: (i) an external one ("black box" view), where the focus is on contracts linking the component to its clients in terms of provided services; (ii) an internal view ("white box" view), hidden from the clients, where the focus is on how the component is organized in terms of parts, sub-components, connectors, etc.

There is two specific connectors for components: (i) an assembly connector is the link between a required (socket) and a provided (lollipop) interface of the same time; (ii) a delegation connector connects a port on the container to/from an internal port/part that has a compatible interface of the same kind (both provided or both required). An arrow indicate the delegation direction. To be more precise, connectors are between parts/ports that have compatible interfaces of different kinds (one provided, one required). It is one way to wire components together (the other way is to use dependencies as in UML 1.5 version, but these do not have an instance level counterpart).

3.3. **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 several notions of components. BasicComponents and PackagingComponents are capabilities of components modeled in separate packages for convenience of implementation. PackagingComponents are an extension of BasicComponents to define the grouping aspects of packaging components. A basic component inside a packaging component is informally called a *nested component*. Note that this notion is different from the one of a part, which is not specific to components, and which define an element of an internal structure. 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 to most impact the ongoing research (such as ours [3]) is the introduction of the StructuredClass 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 [11] or capsules in ROOM [22]. Note that components can also communicate directly point to point, using the same schema as in the Corba Component Model (CCM [10]). Nevertheless it has to be conclude that components add only a little to structured classes as far as composition is concerned.

4. Existing efforts

Several working directions projects address the issue of UML components. We can classify them into the following categories:

- commercial tools - the various existing commercial UML tools offer support for compositional modeling in UML to various extents, most often on the model editing level. Few tools, such as Rhapsody,Telelogic Tau 2, etc. offer support for more advanced component related treatment such as component specific code generation, verifications, symbolic execution, etc.
- standardization - related efforts: EDOC and SPT.
- research projects ACCORD, AIT-WOODDES, OMEGA, CML etc.

In what follows we give some more details on the efforts mentioned at the previous last two points.

The **EDOC** (Enterprise Distributed Object Computing) [21] has a standard profile for UML, which introduces the notion of Component Collaboration Architecture (CCA), where some concepts such as process components, ports and connections are defined.

The UML Profile for Schedulability, Performance and Time (**SPT**) [22] offers no particular treatment to components, although it is implied that the component approach is suitable for the real-time domain. When describing concurrency, components are mentioned as one of the different kinds of *concurrent units*.

ACCORD is a French project aiming to define a Platform Independent component model, based on UML [27]. The proposed model uses the extension mechanisms of UML 1.5 to support concepts such as components, ports and connectors. In this model, component operations are attached to ports and the architectural constraints are expressed in the means of OCL meta-level constraints. One of the main contributions of this project is that to illustrate the derivation of a general model towards Platform Specific Models (PSMs) such as CCM or EJB.

Related to ACCORD, the **AIT-WOODDES** project focused on the specification of embedded real-time systems. One of the results of the project is a specific component model [26] using concepts developed in ACCORD project with specific real-time aspects.

The IST project **OMEGA** aimed to develop a methodology for modeling real-time and embedded systems in UML, with the aid of formal techniques. In this context, the efforts go towards both finding the best theoretical framework for verification in this precise context, and towards implementing them into tool sets developed on the top of UML commercial tools. In this context, a particular attention is given to the use of an appropriate component model, important for enforcing property-preserving refinement and for enabling efficient validation. The OMEGA component model is based inspired from the UML 2.0 component model, however the tool sets was built upon UML 1.5 tools, for commercial tool availability reasons.

The **CML** project [8] is an ongoing project aiming at the definition of a general purpose component model, based on the UML notation, and on a formal framework based on the Whole-Part Relationship (WPR). Two notions of composability are used: *horizontal composability* - component binding and cooperation in distributed systems and *vertical composability* - whole (container) components process requests of client components. Part components are fully encapsulated (into whole components) and are not units of deployment in this particular context, they provide implementations for the components that they belong to by means of delegation. This approach [4] proposes to extend the semantic properties of the whole-part relationship [3] by adapting its formal base to software composition.

5. UML 2.0 components analyzed from different perspectives

5.1. **Critical point of view on the new UML 2.0 artifacts.** UML 2.0 provides useful notation artifacts that were missing in UML 1.5 [7]. We believe those improvements are not enough. This is why we have developed our own approach of composition using UML 2.0 [4], and why certainly others will do the same in the near future. Our theoretical framework is based on an adaptation to CBSE of a formalization of the whole-part relationship (WPR) [3]. We have extended the semantic properties of the WPR by adapting its formal base to software composition. We have determined which properties apply to component composition and defined formally these properties. We ground our approach on metamodeling and assertions in order to constrain the specification of composition relationships. Constraints are added to components at implementation time via generated contracts. In our approach, the notion of Whole component can be linked to the one of PackagingComponent and the Part components to the BasicComponent one. The benefits of our approach is the well definition of precise properties to characterize composition. For details and illustration of our approach, see [4, 5].

5.2. **Support for composition.** The requirements of the OMG RFP were explicit in terms of CBSE support: support for component assembly and plug-substitutability, support for the specification of common interaction patterns that might occur between components, support for modeling of component execution

contexts (e.g., EJB and CCM containers), and support for profiles definitions for specific component architectures models (such as EJB, CORBA, and COM+). The effort of the proposal to address these requirements is undoubtable as we have described in section 3. Most of the construct needed for CBSE support have been introduced. A set of recommendation had been produced based on the several UML 2.0 RFI responses. In terms of composition, they were mainly: (i) improve the semantics and notation to support component-based development, (ii) better support for interfaces, and (iii) usage protocol for interfaces formalization. As we have mentioned before, the first point have been missed, in our opinion, in the sense that there is no semantic provided for the added constructs.

To summarize the pluses of the new UML 2.0 constructs we could say that it provides: means to express architecture, means to express component contracts, and more expressive communication description. And in terms of the minuses, we could say that: there is too many overlapping constructs, relationship between the various constructs are not clear, and the semantics has too many traps (e.g., misuse of new concepts) and lacks (e.g., of explanation and illustration of the usage of new concepts, informal assumptions, ...).

5.3. **Support for Agile Modeling.** As an illustration of the concrete use of the improvements brought by the new version, let us mention some of those made by Scott Ambler in its overall *Agile Modeling* approach [1]. Here are some examples of his proposal. In the Component Diagram, Ambler suggests that the ports should be linked to one interface only. This seems to be only informally assumed in some of the UML 2 spec, but there is no explicit restriction of the number of interfaces on a port. This simplifies the delegation mechanisms (internal structure). Three different relationships between ports and parts have been identified: (i) delegates, which is the usual relationship between a port and a part in UML 2.0. (ii) stereotyped delegates, which specific notation comes in replacement of the previous in order to prevent confusion with the unidirectional association, which has the very same drawing. (iii) realizes, which indicates that it is the realization of a port. Ports are viewed in this case as logical modeling constructs realized by physical constructs such as classes.

Another example is the use of class to systematically implements ports (using the *Façade* design pattern [13]). These classes implement the public operations required by the interfaces.

The main conclusion of the use of UML 2.0 from an *agile* point of view is the fact that it eases the application of the heuristics that were already defined in terms of good practices by supplying new and useful constructs (differentiation between application, infrastructure and domain components, definition of component contracts, etc.).

5.4. **Expressive power of the architecture description.** [16] presents a deep analysis of UML 1.5's expressive power for modeling software architecture, in a way

natural to the way this is done in traditional architecture description languages. In the followings we start from the results of this analysis, and try to see how they are affected by the changes made in UML 2.0, and to what extent UML 2.0 answers or not the specific needs of software architecture modeling.

In [16], Medvidovic *et al.* identify a minimum set of requirements for evaluating the ability of UML to be used in software architecture modeling efficiently. In order to do this the authors apply two approaches for supporting architectural concerns within UML: the first is based on using UML "as it is", and the other on using standard UML extension mechanisms. The results of this study are that UML can be used to address architectural concerns, although this requires some extra-efforts and its it has some drawbacks when compared to the use of classical ADL for the same purpose.

The general conclusion is that the extensible design of UML renders it applicable to system architecture modeling. However, as the language was not primarily designed for this, there are some drawbacks in using UML for this purpose. Some architecture modeling specific concepts are missing, and they need to be added in UML, e.g. using extension mechanisms. As a result, the rules of architectural style (present in ADLs and maintained by ADL supporting tools) have to be managed and applied by the modeler and need to be documented in addition to the system/architecture modeling. This makes the design more difficult to manage and to maintain.

Another conclusion regards the modeling of behavior and interactions in the architecture modelled using UML 1.5. This is possible using sequence diagrams, collaboration diagrams, or state machine diagrams. However it is hard to establish the relationship between the specified behavior and the architectural elements, and it is hard to ensure that the intended behavior is correctly modelled in UML.

Apart of the general points mentioned before, we extract here those that seemed to us among the most noteworthy with respect to UML 1.5 suitability for architecture modeling:

(1) For UML 1.5 classes it is only possible to specify the list of events it can receive, not of those that can be sent. This is different from the common practice when using ADLs;
(2) As UML was primarily designed to address various kinds of concerns, although it can handle architecture specific needs, architecture modelers find the support for these aspects partially sufficient;
(3) Some of the software architecture-specific concepts are conceptually different of those existing in UML (and more generally in object-oriented design). It is the case for example of connectors. They can indeed be abstracted by a UML class (e.g., by using stereotypes), however some of the properties ADL connector's properties need to be explicitly modelled in UML. This is the case of some ADL, where connector interfaces are

context reflective. In UML this needs to be explicitly modelled, which makes the use of connectors in UML 1.5 heavier.

(4) The UML tool support do offer (as far as the authors of the cited paper and ourselves are aware) some of the features regularly offered by ADL supporting tools. In particular, the part concerning the infrastructure of the systems modelled (that enforces the desired topology, interface and interactions between system components) is not covered by general UML tools, and not offered by other UML supporting tools.

In the study above mentioned, among the weaknesses of UML 1.5 in supporting architecture modeling was the lack of some software architecture specific concepts. The situation changed a bit with the new UML 2.0, as some of these concepts are added to the language definition. In the followings, we will discuss on how much this impacted on the language ability to be used in architectural description.

The add of concepts like *port, port instance, connector*, and *architecture diagram* enriches the expressive power of UML, when it comes to architecture modeling. Indeed, having these concepts explicitly at language level (not having to model each of them) increases the readability of architecture designs done with UML, and avoids the use of conventions. However, as for the moment only very limited tool support exists for UML 2.0, it is hard to assess the impact of this improvement on the architectural modeling: how well the UML tools will manage the desired topologies, interfaces and interactions between system components.

A step forward was also done towards modeling the behavior and interactions in architecture modeling, by adding the possibility to specify/constrain the behavior as observed at ports or interfaces through protocol state machines. We see however a problem in the abundance of means existing in UML 2.0 to specify behavior and interactions in architecture, especially as the relationship between the various means is not clearly specified in the standard. Without a good modeling methodology and the right tool support, we fear that the new additions will not be used, due to the risk of confusion.

If we look at the list of more precise points raised on the UML 1.5 ability to address architecture modeling needs, we notice that some of them (point 1 and 3) are partially solved by adding new concepts to UML. Although it is still not possible to specify what events can an object generate to its environment, using implemented and required interfaces, it is possible to give information on both directions of the interaction of an object with its context.

Our discussion represents a first level analysis on the use of UML 2.0 for software architecture modeling, having as starting point the extensive study performed on UML 1.5 in [16], and the new UML 2.0 standard. This allowed us to infer the effect of the UML evolution. A deeper analysis, similar to the one performed for UML 1.5 is probably needed to correctly asses all the details of the UML 2.0 ability to address architecture modeling.

6. Conclusion

This paper aimed at discussing whether the recently adopted UML 2.0 proposal was good or not in terms of composition support. We have presented in this paper a brief overview of the new UML component model and we have done our best to found some good arguments in favor of the affirmative answer to this question. It is undoubtable that the improvements at the notation level will be useful for component-based systems developers. The new component model is more expressive and flexible than the one offered in the old version of the language. The main requests formulated in the RFI have been answered. However, some points still remain to be clarified, especially when it comes to the combined use of various concepts. This may require a non minor amount of work, if one wants to keep using all the concepts offered by UML 2.0. The application of the UML 2.0 on concrete case studies will tell how well new UML component model is adapted to the industry.

A step forward on the way of applying UML on system engineering is represented by the SysML initiative. SysML is a modeling language for systems engineering applications called Systems Modeling Language. SysML will customize UML 2.0 in order to support the specification, analysis, design, verification and validation of complex systems that include hardware and software components. It will be interesting to also follow what the people who already had some proposals for UML support for components, as presented in section 4, will answer to this same question. This might lead to interesting developments in this area.

References

[1] Scott W. Ambler. The Official Agile Modeling Site – The Diagrams of UML 2. Available at `http://www.agilemodeling.com`, 2003.

[2] ARTIST. Component-based Design and Integration Platforms : Roadmap. Technical Report W1.A2.N1.Y1, Project IST-2001-34820, 2003.

[3] Franck Barbier, Brian Henderson-Sellers, Annig Le Parc-Lacayrelle, and Jean-Michel Bruel. Formalization of the Whole-Part Relationship in the Unified Modeling Language. *IEEE Transactions on Software Engineering*, 29(5):459–470, May 2003.

[4] Nicolas Belloir, Jean-Michel Bruel, and Franck Barbier. Whole-Part Relationships for Software Component Combination. In Gerhard Chroust and Christian Hofer, editors, *Proceedings of the 29th Euromicro Conference on Component-Based Software Engineering*, pages 86–91. IEEE Computer Society Press, September 2003.

[5] Nicolas Belloir, Fabien Roméo, and Jean-Michel Bruel. Whole-Part based Composition Approach: a Case Study. In *Proceedings of the 30th Euromicro Conference – Component-Based Software Engineering Track (Euromicro'2004)*, March 2004. To be published.

[6] Antoine Beugnard, Jean-Marc Jézéquel, Noël Plouzeau, and Damien Watkins. Making components contract aware. *IEEE Computer*, 13(7):38–45, 1999.

[7] Conrad Bock. UML 2 Composition Model. *Journal of Object Technology*, 3(10):47–73, November-December 2004.

[8] Jean-Michel Bruel. CML – Component Modeling Language: project proposal. Technical report, French National Sciences Funds, 2003.

[9] E. Bruneton, T. Coupaye, and Jean-Bernard Stefani. Recursive and Dynamic Software Composition with Sharing. In *Proceedings of 7th International Workshop on Component-Oriented Programming – WCOP02 at ECOOP 2002*, June 2002.
[10] CCM. CORBA Component Model. OMG Report ptc/02-08-03. URL: `http://www.omg.org/`.
[11] Laurent Doldi. *UML 2 Illustrated – Developing Real-Time & Communications Systems*. TMSO, October 2003.
[12] Desmond D'Souza and Alan Cameron Wills. *Objects, Components and Frameworks With UML: The Catalysis Approach*. Addison-Wesley, 1998.
[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
[14] Gregor Goessler and Joseph Sifakis. Composition for Component-Based Modeling. In *Proceedings of FMCO, November 5-8, 2002*, volume 2852. LNCS, 2003.
[15] Frank Lüders, Kung-Kiu Lau, and Shui-Ming Ho. *Building reliable component-based software systems*, chapter Specification of Software Components, pages 23–38. Number 2. Artech House Publishers, Boston, 2002.
[16] N. Medvidovic, D. S. Rosenblum, D. F. Redmiles, and J. E. Robbins. Modeling software architectures in the Unified Modeling Language. *ACM Transactions on Software Engineering and Methodology*, 11(1), 2002.
[17] Bertrand Meyer. Contracts for components. *Software Development Online*, July, 2000.
[18] Bertrand Meyer. The grand challenge of trusted components. In *ICSE 25, Portland, Oregon, May 2003*. IEEE Computer Press, 2003.
[19] Oscar Nierstrasz and Theo Dirk Meijler. Requirements for a Composition Language. In *Proceedings of ECOOP 94 workshop on Models and Languages for Coordination of Parallelism ad Distribution*, LNCS, pages 147–161. Springer Verlag, 1994.
[20] A. Olafsson and D. Bryan. On the need for required interfaces to components. In *Special Issues in Object-Oriented Programming – ECOOP 96 Workshop Reader*, pages 159–171. dpunkt Verlag, Heidelberg, 1997.
[21] OMG. UML Profile for Enterprise Distributed Object Computing Specification (EDOC). OMG document, Object Management Group, may 2002.
[22] OMG. UML Profile for Schedulability, Performance, and Time Specification (PST), Draft Adopted Specification. OMG document, Object Management Group, January 2002.
[23] OMG. Unified Modeling Language: Infrastructure and Superstructure, version 2.0. OMG document formal/05-07-05 and formal/05-07-04, Object Management Group, March 2006.
[24] High Confidence Software and Systems Coordinating Group. High Confidence Software and Systems Research Needs. Technical report, Interagency Working Group on Information Technology Research and Development, january 2001.
[25] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, ACM Press, NY, 1999.
[26] Francois Terrier and Sébastien Gérard. UML 2: Component model and RT feedback of AIT-WOODDES project to the U2 proposal. In *Workshop SIVOES-MONA - UML'2002, Dresden*, 2002.
[27] Tewfik Ziadi, Bruno Traverson, and Jean-Marc Jézéquel. From a UML Platform Independent Component Model to Platform Specific Component Models. In Jean Bezivin and Robert France, editors, *Workshop in Software Model Engineering*, 2002.

LIUPPA, Université de Pau et des Pays de l'Adour, 64000 Pau, France
E-mail address: `Jean-Michel.Bruel@univ-pau.fr`

IRIT, Université Paul Sabatier, Toulouse, France
E-mail address: `ileana.ober@irit.fr`