Towards a Base UML Profile for Architecture Description*

Apostolos Zarras, Valérie Issarny, Christos Kloukinas, Viet Khoi Nguyen
INRIA Rocquencourt
Domaine de Voluceau, BP 105, 78 153 Le Chesnay Cédex, France
{Apostolos.Zarras, Valerie.Issarny, Christos.Kloukinas, knguyen}@inria.fr

ABSTRACT
This paper discusses a base UML profile for architecture description as supported by existing Architecture Description Languages (ADLs). The profile may be extended so as to enable architecture modeling both as expressed in conventional ADLs and according to existing runtime infrastructures (e.g., system based on middleware architectures).

1 INTRODUCTION

Architectural description of a software system is now recognized as a sound practice towards assisting the system's design, analysis and construction as well as for coping with the system's deployment and evolution. It is further argued that architectural description should not be based on a single notation but instead rely on a number of Architecture Description Languages (ADL) according to the system's views that need be characterized [4]. However, one issue that remains and that is being examined by a number of people is whether the notations underpinning the definition of an ADL should be based on object modeling notations or not. In the latter case, the ADL, qualified as *conventional*, is a declarative language that is either based on an IDL (Interface Definition Language) with adequate extensions so as to assist the construction phase, or on existing formal notations that come along with methods and tools for mechanical analyses [6]. There has been a number of studies on the mapping of architectural models as expressed in conventional ADLs, into object notations. Two approaches are considered: (i) describing architectures using the object notations *as is* [7, 5], (ii) extending the object notations so as to explicitly distinguish the key architectural elements from object-oriented notations. The latter approach has been addressed in [10], which examines the mapping into UML of two specific ADLs (i.e., Wright [1] and C2 [11]). A broader perspective is undertaken in [2], which studies various alternatives to modeling architectures using UML, both *as is* and through extension. The conclusion that arises from these proposals is that there is no consensus on the best practice to modeling architectures, leaving open the relevance of using object notations and the mapping of conventional ADLs to these notations when they are to be used.

From a pragmatic standpoint, architecture description as addressed in conventional ADLs would have a greater impact if it was coupled with object notations. This would promote the exploitation of architecture description notations by software engineers, would enable the use of available commercial tools for assisting architecture-oriented software development, and would support software development in a unified environment from the architecting of the system to the implementation of its composing elements. In addition, a major advance brought by conventional ADLs lies in enabling the practical exploitation of formal methods for reasoning about the behavior of large software systems, hence promoting system robustness. However, there is still a long way to go before formal notations get used by a majority of software engineers. This will be easier to achieve if formal notations were offered in a conventional software development environment, possibly in their simplest form. This position paper proposes a base UML profile for characterizing architectural elements, which may be extended so as to allow the exploitation of the rich set of both architecture models as expressed by conventional ADLs, and available infrastructures for software development (e.g., middleware infrastructures). This effort relates to the definition of architecture meta-languages (e.g., ACME [3], AML [12]) where we are proposing the core generic notations for architecture description in a UML setting. The next section characterizes the base architectural elements that need be modeled and serves defining the corresponding base UML profile in Section 3. Sections 4 and 5 then sketch extensions of the proposed UML elements, so as to enable the exploitation of existing supports for software system development from architecting to implementation. Finally, Section 6 summarizes our contribution and discusses our ongoing and future work.

2 BASE ARCHITECTURAL ELEMENTS

A primary benefit of software architecture description comes from the focus on the system's structure and resulting abstraction of implementation details. Hence, this enables

*This work has been partially funded by the IST DSoS Project IST-1999-11585 (http://www.newcastle.research.ec.org/dsos/index.html).

defining tractable system models, which may further be processed for mechanizing parts of the design, analysis, construction or deployment phases. In addition, this eases dealing with the evolution of the software system by enabling reasoning about the impact of changes to the system's structure as well as devising runtime support for handling such changes dynamically. Basically, software architecture description lies in the following notions:

- **Component:** This abstracts a unit of computation or storage of the system. A component may have a number of interfaces (sometimes called *ports*) that specify how the component interacts with its environment by either offering or requiring operations.
- **Connector:** This models an interaction protocol among components. Connectors range from simple interaction protocols (e.g., message passing) to complex ones (e.g., a middleware architecture). A connector may define interfaces (sometimes called *role*) that specify the protocol's participants.
- **Configuration:** This defines a system model at the architectural level through the specification of embedded component and connector types, and the assembly of component instances *via* connector instances at their respective interfacing points. Although not supported by all the ADLs, the definition of configurations may be hierarchical in that both components and connectors may be refined into configurations as the system development progresses. We qualify as *composite* such components and connectors.

Conventional ADLs differ in the notations they offer to specify the above architectural elements [6]. They may be roughly subdivided into two categories depending on whether they aim at assisting the system's analysis or construction. In the former case, the notations allow specifying behavioral views of the system regarding either functional or extra-functional properties of the system, while in the latter case, the notations are oriented towards generating gluing code. Notice further that there is no clear consensus on whether connectors should be first class elements or not since system modeling regarding interaction patterns may be addressed at the components' interfaces, and the realization of complex connectors may be specified using components. We consider that both approaches are relevant, which we capture through the notions of *abstract* and *concrete* connectors.

3 BASE UML PROFILE

There are two basic approaches to defining base architectural elements in UML: (i) mapping the elements onto UML elements, (ii) defining corresponding extensions to the UML meta-model. We undertake the second approach because architectural notions are distinct from object-oriented ones. This further enables sound transition from architecture-based design to object-oriented design at an elaborated stage of the software development[1]. The issue that must be addressed is then identifying for each base architectural element, the UML modeling element that best matches it and that need be extended.

Component: As discussed in the literature, various UML modeling elements may be extended to characterize an architecture component (i.e., Class, Component, Package, Subsystem). We define the ADLComponent stereotype as being an extension of the Subsystem element. A subsystem is defined as "*a grouping of model elements, of which some constitute a specification of the behavior offered by the other contained elements*" [8], which in particular adds specification elements to the definition of the Package element. The UML Component element was not considered as it specifically corresponds to an executable software module. The Class element is often considered as the basis for defining architectural components. However, this is a less flexible solution in that it does not directly support the specification of composite architectural elements, which would require providing both the class definition and associated class diagram, and hence a package integrating them. Considering further that the definition of architectural components is to come along with behavioral specification, this naturally leads to adopt a stereotype based on the Subsystem element. We get the following definition for the ADLComponent stereotype whose distinctive feature relates to the specification of a number of provided and required interfaces:

```
ADLComponent:
  self.baseClass = Subsystem and
  self.extendedElement.Instantiable = true and
  self.extendedElement.requirement->collect(d:Dependency |
     d.client = self and d.supplier.oclIsKindOf(Interface)
  )->size >= 0 and
  self.extendedElement.Interface >= 0
```

Abstract Connector: An abstract connector is simply a connection element among architectural components. Hence, it is defined as a stereotype based on the Association element:

```
ADLConnector:
  self.baseClass = Association and
  self.extendedElement.allConnection->forall(
    ae, ae':AssociationEnd |
      ae.taggedValue->exists(tv:TaggedValue |
        tv.name = "role" and
        tv.value.oclIsKindOf(Set(Interface)) and
        tv.value->size >= 2 and
        tv.value->forall(i: Interface |
          ae.type.requirement->exists(
            d: Dependency | tv.value->includes(d.supplier)) or
          ae.type.interface->exists(
            i: Interface | tv.value->includes(i)))
      ) and
      ae'.taggedValue->exists(tv: TaggedValue |
         tv.name = "role" and
         tv.value.isOclKindOf(Interface))
  )
```

[1] For instance, an architectural component abstracts the realization of a software element that may be detailed using object modeling.

The distinctive feature of the above stereotype lies in the definition of *roles* that are bound by the connector. In addition, associations are constrained so that their ends bind required interfaces with provided ones.

Concrete Connector: In general, connectors abstract away complex interaction protocols that are built using a number of software elements. A typical example is the use of a CORBA-compliant middleware that offers a number of extra-functional properties and hence combines the CORBA ORB and some common object services. It is thus crucial to support connector refinement as for architectural components. Hence, the ADLConcreteConnector stereotype is defined through specialization of the ADLComponent and ADLConnector stereotypes (see Figure 1).

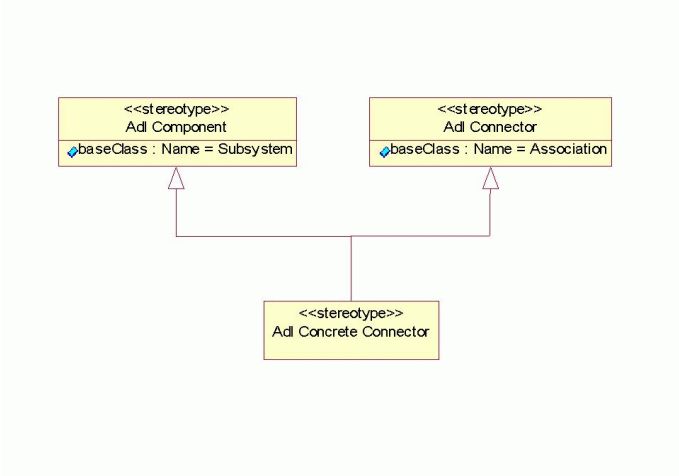

Figure 1: The ADLConcreteConnector stereotype

Configuration: A configuration is defined through extension of the Subsystem stereotype whose instances are realized by assembling only instances of ADLComponent via instances of ADLConnector (including their specializations). We get:

```
ADLConfiguration:
  self.baseClass = Subsystem
  self.extendedElement.isInstantiable = true
  self.extendedElement.contents->forall(c |
    c.oclIsKindOf(ADLComponent) or
    c.oclIsKindOf(ADLConnector) or
    c.oclIsKindOf(ADLConcreteConnector))
```

For illustration, Figure 2 gives the diagrammatic specification of a base client-server architecture using our profile.

4 EXTENDING THE PROFILE FOR THE DEFINITION OF ARCHITECTURAL VIEWS

The base UML profile for software architecture description enables describing the system's structure and exploiting UML-based support for the system development. However, thorough architecture-oriented design of the software system

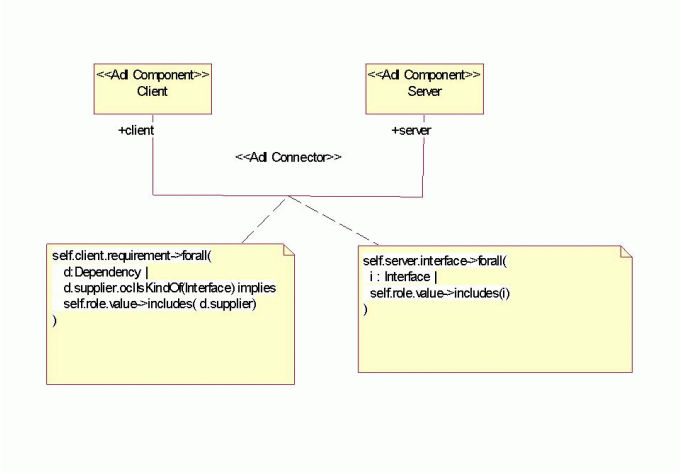

Figure 2: A Client-Server Architecture

is better supported if it is possible to define system views [4] as supported by conventional ADLs, so as to exploit associated tools for system development. This is realized by defining various extensions of the base profile where each extension guides the specification of architectural elements according to a given conventional ADL. Basically, conventional ADLs are distinguishable with respect to: (i) the notations used for specifying the behavior of architectural elements, which are often based on previously defined formal notations (e.g., Wright uses CSP), and (ii) the constraints associated with the definition of the architectural elements. These issues are respectively addressed through the addition of adequate tagged values and the specification of constraints in OCL.

For exemplification, let us outline how to extend our base profile to specify Wright-like architectures. Basically, a Wright specification amounts to specifying each architectural element as a number of CSP processes (i.e., a process for each port and role and a process specifying the overall behavior of the architectural element). Then, system analysis is mechanized through the use of the FDR tool. Considering the base UML profile introduced in the previous section, what we need to add is the specification of CSPmodel tagged values that encode CSP processes, within the definition of architectural elements, including their embedded interfaces. The extension for Wright thus consists in: (i) defining the WrightInterface stereotype that is an extension of the UML Interface element with a CSPmodel tagged value, (ii) extending the architectural elements stereotypes through inheritance with a CSPmodel tagged value and requiring their embedded interfaces to be an instance of the WrightInterface stereotype, and (iii) constraining the definition of the architectural elements so as to enforce the consistency rules as defined by Wright[2]. System architectures described using

[2]For instance, the behavior of a component interface bound to a role must be compatible with the role's behavior according to the CSP refinement

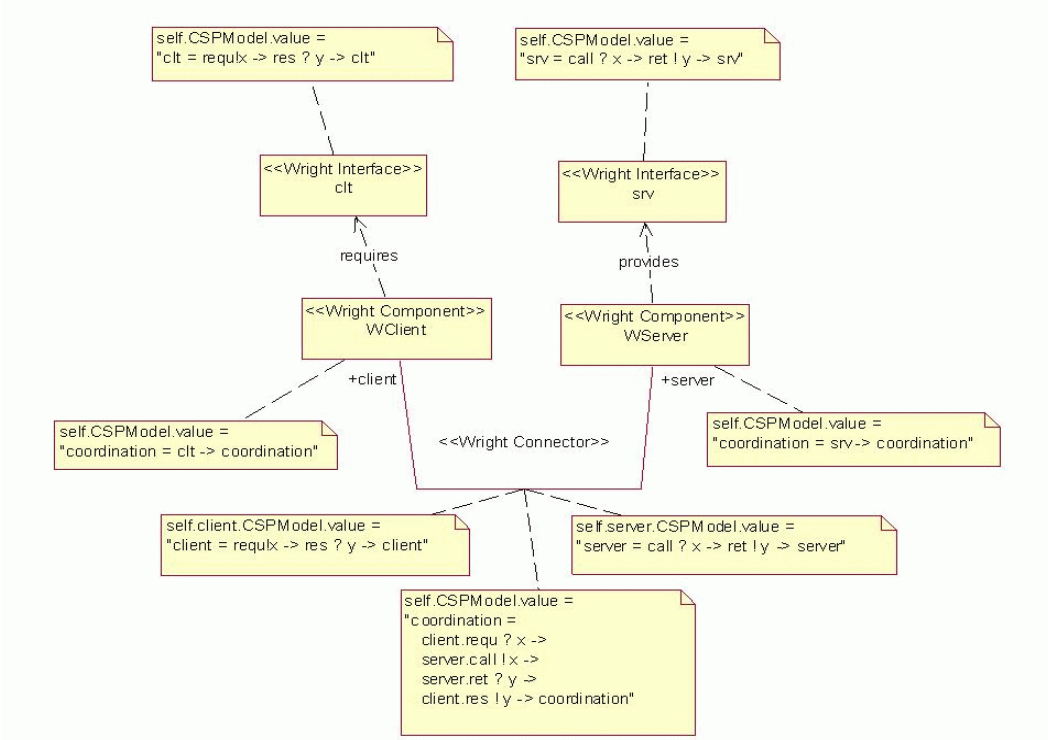

Figure 3: Wright-based specification of the Client-Server Architecture

the above elements may then be processed to generate a system model that is analyzable by the FDR tool so as to check system properties as supported by Wright. For illustration, Figure 3 gives the Wright-base diagrammatic specification of the client-server architecture that was introduced in the previous section.

5 EXTENDING THE PROFILE FOR THE DEFINITION OF ARCHITECTURAL STYLES

Architecture-oriented development of software systems comes along with the definition of architectural styles that define classes of systems, which promotes software and design reuse, and enables reasoning about system evolution. The definition of a style is addressed in our framework by extending the definition of the architectural elements according to the style's specifics. As an example, let us take the specification of the CORBA architectural style. The corresponding extension of the ADLConcreteConnector element embeds the specification of the proxy and skeleton architectural components. The extended connector may then be refined for the case where the middleware architecture offers enhanced non-functional properties (e.g., transactions) by combining relevant CORBA services[3]. Figure 4 gives the base diagrammatic specification of the client-server architecture that was introduced in the previous section according to the CORBA style.

relationship.

[3]Notice that the specialization may exploit the UML profile for CORBA Specification [9], for the definition of relevant elements.

Ideally, the profile extension appertained to a given architectural style should be provided for the base profile and extensions for defining architectural views discussed in the previous section. However, notice that a given view may actually apply to a number of related styles (e.g., the Wright-based specification that is given for the Client-Server architecture applies to the CORBA-based style).

6 CONCLUSION

This paper has presented a base UML profile for architecture description, which amounts to defining stereotypes characterizing the three key architectural elements, i.e., component, connector and configuration. Extension of the profile has further been discussed from the standpoint of defining both various views and styles of system architectures. Our work resembles the effort on defining an architecture meta-language (in particular ACME [3]) so as to integrate in a single environment the rich toolset for assisting architecture-oriented system development. However, it is different in that it specifically targets UML-based modeling.

We are currently extending the proposed UML profile for architecture description, for the specification of system views and styles oriented towards assessing and improving the system's quality (e.g., performance, reliability). We will then integrate the resulting extensions in the ROSE commercial tool so as to experiment with the use of our solution. We will further examine the coupling of object-oriented and architecture-oriented tools for assisting the development of

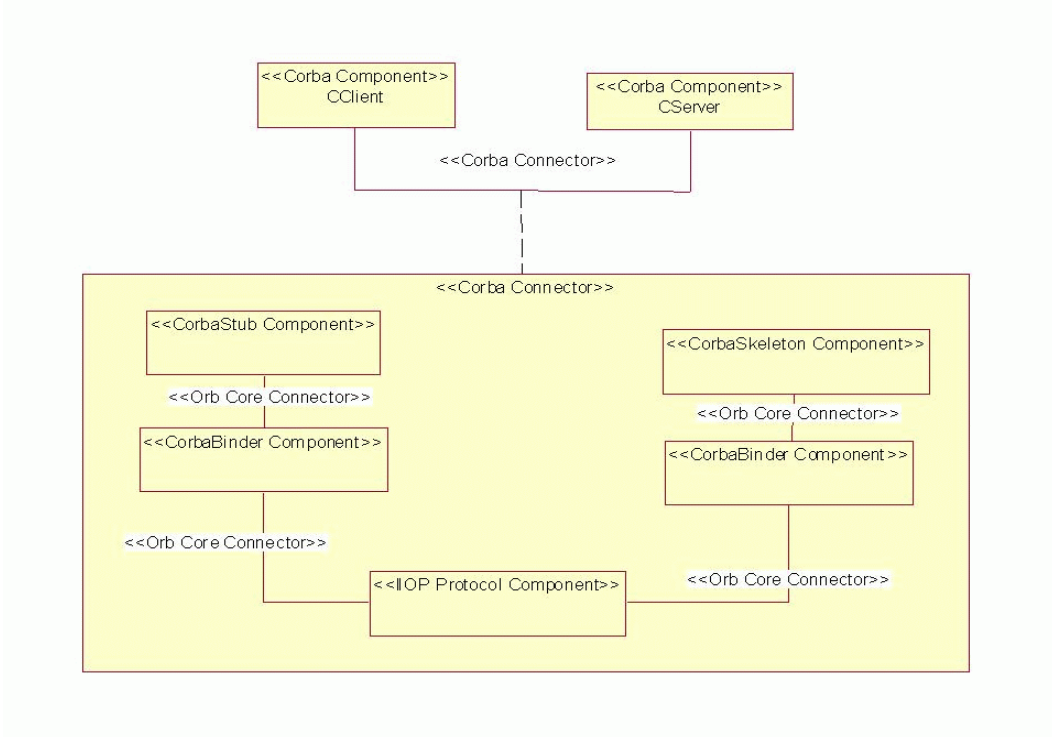

Figure 4: Specification of the CORBA-based Client-Server Architecture

software systems.

REFERENCES

[1] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, 1997.

[2] D. Garlan, A. J. Kompanek, and P. Pinto. Reconciling the needs of architectural description with object-modeling notations. In *Proceedings of the 3rd International Conference on the Unified Modeling Language (UML'2000)*, 2000.

[3] D. Garlan, R. Monroe, and D. Wile. ACME: An architecture interchange language. Technical report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, USA, 1997. http://www.cs.cmu.edu/afs/cs/project/able/www/-papers.html.

[4] IEEE Architecture Working Group. *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems (IEEE Std 1471)*. IEEE, 2000.

[5] N. Medvidovic and D. S. Rosenblum. Assessing the suitability of a standard design method for modeling software architecture. In P. Donohe, editor, *Software Architecture (Proc. of WICSA'1)*, pages 161–182. Kluwer Academic Publishers, 1999.

[6] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[7] R. T. Monroe, D. Kompanek, R. Melton, and D. Garlan. Architectural style, design patterns, and objects. *IEEE Software*, January 1997.

[8] OMG. OMG Unified Modeling Language Specification, Version 1.3. Technical report, OMG Document, 2000. http://http.omg.org.

[9] OMG. UML Profile for CORBA Specification, V1.0. Technical report, OMG Document, 2001. http://http.omg.org.

[10] J. E. Robbins, N. Medvidovic, D. F. Redmiles, and D. S. Rosenblum. Integrating architecture description languages with a standard design method. In *Proceedings of the 20th International Conference on Software Engineering (ICSE'98)*, pages 209–218, 1998.

[11] R. N. Taylor, N. Medvidovic, K. Anderson, E. J. Whitehead, K. A. Nies, P. Oreizy, and D. Dubrow. A component and message-based architectural style for GUI software. *IEEE Transactions on Software Engineering*, 22(6):390–406, 1996.

[12] D. Wile. AML: an architecture meta-language. In *Proceedings of the IEEE International Conference on Automated Software Engineering (ASE)*, 1999.