Towards an UML Profile for the Description of Software Architecture

Abdelkrim Amirat, Mourad Oussalah

Towards an UML Profile for the Description of Software Architecture

Abdelkrim Amirat[1,2] and Mourad Oussalah[1]
[1]*Laboratoire LINA, CNRS UMR 6241, Université de Nantes, France*
[2]*Centre Universitaire de Souk Ahras, Algérie*
{abdelkrim.amirat ; mourad.oussalah}@univ-nantes.fr

Abstract

Existing ADLs (architecture description languages) have an advantage of formally specifying the architecture of component-based systems. But ADLs have not come into extensive use in industries since ADL users should learn a distinct notation specific to architecture, and ADLs do not address all stakes of development process that is becoming diversified everyday. On the other hand, UML is a de facto standard general modeling language for software developments as UML provides a consistent notation and various supporting tools during the whole software development cycle. A number of researches on architecture modeling based on UML have been progressed. In particular, many research results have been introduced that specialize UML by its extension mechanism in order to explicitly represent core architecture concepts that UML does not fully support. In this paper, we examine architecture modeling elements that can be represented in UML2.0 and discuss how to extend and specialize UML2.0 in order to make it more suitable for representing architectures.

Keywords: *Software Architecture Modelling, UML 2.0, OCL, Profile and Metamodel.*

1. Introduction

Software architecture has emerged as an important subdiscipline of software engineering. A key aspect of the design of any software system is its architecture, i.e. the fundamental organization of the system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution [10].

Architecture can be modeled according to different viewpoints. From a run time perspective, two viewpoints are frequently used in software architecture: the structural viewpoint and the behavioural viewpoint [10]. In this work we are interested by the structural viewpoint which can be specified in terms of Components, Connectors and Configurations (C3 model). Thereby, from this viewpoint, an architecture description should provide a formal model of the architecture in terms of components and connectors and how they are composed together.

The Unified Modeling Language (UML) [5] [6] [7] is a family of design notation that is rapidly becoming a de facto standard for representing the software artifacts obtained in the various activities (like requirement acquisition, requirement analysis, system design, or system deployment) of a software development process. For this reason, there have been attempts to use this language to represent the software architecture of systems as well. However, the language is not designed to represent syntactically and semantically the elements of software architecture [2].

The attempts to instantiate the constructors defined in the UML meta model or to extend UML by using stereotypes to represent these elements has driven to the same representations (boxes and lines) that have been widely criticized by the software architecture community. Consequently, the only solution is to extend the UML meta model. However, the extension of the UML meta model implies the modification of the language, which means a deviation from the standard. This has been one of the reasons used in the literature to extend UML with stereotypes or by specifying profiles for the area of interest [11].

A question that arises at this point is why not using Architecture Description Languages (ADLs) to describe the application software architecture, therefore avoiding the change to the UML meta model. Indeed, the currently available architectural description languages (ADLs) have not spread in industry mainly because they are not generic enough, are not standardized and are poorly supported by tools. UML is a standard, but its current semantics fails to meet the criteria stated above: it is weak at describing interfaces, the abstractions it provides are not univocal

and it provides little support for modeling architecturally significant information [3]. Additionally, the ADLs are not integrated in any development process (like the Unified Software Development Process [4]), while UML is. Hence, representing the application architecture with UML allows the integration of this representation with the rest of software artifacts. In this paper, we propose UML 2.0 profile for explicit components, connectors and configurations defined in previous work [8] [9].

The remainder of the paper is organized as follows. In Section 2 we describe the main elements that appear in the description of the C3 architectural elements. Section 3 describes the UML extension profile as specified by the Object Management Group (OMG). In Section 4 we present several attempts to extend UML for representing software architecture. In Section 5 we characterize C3 elements as UML meta classes by defining UML Profile. Finally, Section 6 presents conclusions and future lines of research.

2. Basic Architecture Elements of C3 Model

The C3 model supports description of software architectures from a structural viewpoint. In C3, architecture is described in terms of components, connectors, and their composition (configuration). Figure 1 depicts its main constituents.

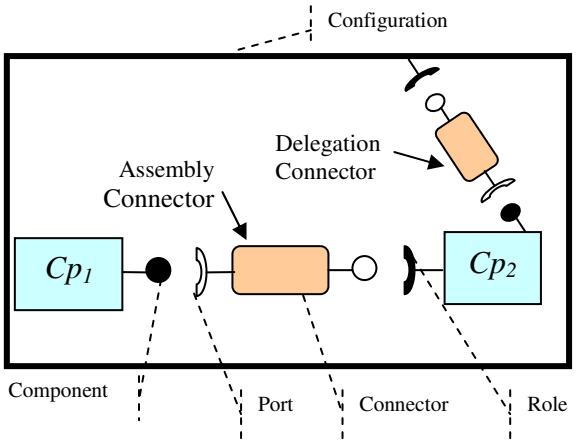

Figure 1. Architectural Concepts

Components are described in terms of external interfaces and an internal behaviour. Their architecture role is to specify computational elements of a software system. Interfaces are described in terms of ports and services. Ports are described in terms of connections between a component and its environment. The figure 2 defines the metamodel of component concept in C3 from the structural point view.

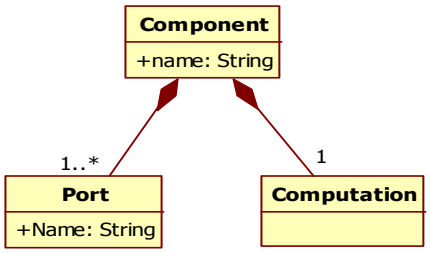

Figure 2. Component Meta Model in C3

Connectors are special-purpose components. They are described as component in terms of external interfaces and internal behaviour. However, their architectural role is to connect together components. They specify interactions among components. The internal behaviour is described by the glue protocol. Interfaces are described in terms of roles and services. Attachments describe the different possible connection of roles with the external environment. Figure 3 depicts the main constituent of connectors.

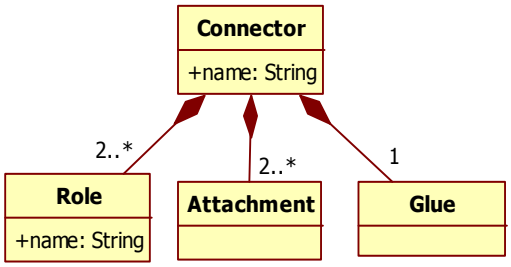

Figure 3. Connector Meta Model in C3

In order to attach a port to role, the interfaces of the two elements must be compatible, i.e. the type of the component must be defined in interface of the connector. So, the provided port will be connected with required role and required port will be connected with provided role. Thereby, attached port/role can transport values (that can be data, connections, or even architectural elements. From a black-box perspective, only port of components and roles of connectors and values passing through connections are observable.

Components and connectors can be composed to construct *configuration* (composite elements), which themselves will become components. Configurations can be decomposed and recomposed in different ways or with different components in order to construct different compositions. The visible parts of

configurations are their interfaces which are defined in terms of ports and services. Ports are described in terms of connections between the configuration and its internals from one side and from the other side between the configuration and its environment. The figure 4 defines the meta model of configuration concept in C3 from the structural point view.

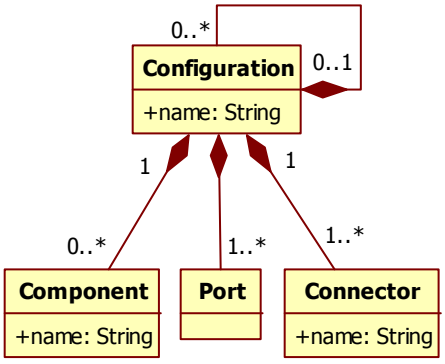

Figure 4. Configuration Meta Model in C3

3. UML 2.0 Profile

UML provides a number of extension mechanisms that allow designers to customize and extend the semantics of model elements:

Constraints place added semantics restrictions on model elements. The possibilities for constraints are numerous and include type constraints on class attribute values, constraints on the construction of associations between classes, and so on.

Tagged values: Allow new attributes to be added to particular elements of the model. The stereotype defines a number of tagged values. Each tagged value is typed with a data type number, string, boolean, or user-defined enumeration.

Stereotypes allow groups of constraints and tagged values to be given descriptive names (with the same specified in double angle brackets), and applied to model elements, effectively creating a new yet restricted form of a meta class for constructing models. The semantic effect is as if the constraints and tagged values were attached directly to those elements.

UML Profiles combine the concepts of stereotypes, tagged values, and constraints to provide a coherent and concise dialect of UML for specific family of applications.

4. UML Extension Mechanisms

UML 2.0 has become an industry standard for modeling, design and construction of software systems as well as more generalized business and scientific processes. In UML 2.0 there is no specific diagram for modeling architectures. In fact, constructs for architecture description are not directly provided but architecture description is supported and can be expressed as a combination of different views, e.g. 4+1 views.

UML 2.0 provides a major improvement in its support to architecture description with a major enhancement in the Component Diagram and the introduction of a new diagram Composite Structure Diagram. So, in UML 2.0 components have been generalised, and are considered as higher-level than classes.

The definition of UML Profiles for modelling software architecture is not new; [1] identifies three possible strategies for modeling software architectures using UML. The four-layer meta modelling architecture of UML suggests three possible strategies for modeling software architectures using UML.

- Using UML “as is”
- Constrain the UML meta model using UML’s built-in extension mechanisms (e.g. UML Profile).
- Extend the UML meta model to directly support the needed architectural concepts.

Each strategy has certain potential advantages and disadvantages. This section presents a brief discussion and preliminary evaluation of the strategies. In order to reap the benefits of standardization we require that any resulting notation adhere to the syntax and semantics of UML.

4.1 Using UML “As Is”

Using UML 2.0 “As Is” is not the good choice of strategy [1]. The modeling capabilities provided by UML 2.0 “As Is” do not fully satisfy the structural and behavioural requirements for describing software architectures, because UML 2.0 does not provide specialized constructs for modeling software architectures, in particular for modeling software architecture from a runtime perspective. For example, although they are different architectural elements with very different responsibilities, components and connectors must be modeled in UML 2.0 using the same mechanism. Hence, describing software architecture in UML 2.0 is an error-prone approach.

4.2 Constraining UML

This strategy uses profiles, also some times called *lightweight* built-in extension mechanisms. The most important profile element is the stereotype. Stereotyping is a pure extension mechanism. The model elements marked with a stereotype have the same structure (attributes, associations, operations) defined by the meta model element that describes them, plus the constraints and tagged values added by the stereotype to that meta model element. This is accomplished via the extension mechanisms described in section 3. However, with stereotypes we can not change the semantics of the meta model elements (at most, we can refine it), change its structure, nor create new elements of that meta model. So, the architecture specified in this manner would still be manipulated by standard UML tools and would understandable to UML users.

4.3 Augmenting UML

This strategy is a *heavyweight* extensibility mechanism as defined by the specification of Meta Object Facility (MOF) [5][11]. In this strategy the goal is to extend the UML meta model by explicitly adding new meta classes and other meta constructors. The potential benefit of such an extension is that it could fully capture every desired feature of every ADL and provide "native" support for software architectures in UML. However, the challenge of standardization is finding a language that is general enough to capture needed concepts without adding too much complexity, while such a modification would result in a notation that is overly complex. More importantly, the notation would not conform to the UML standard and could become incompatible with UML compliant-tools.

In this work we have experimented with the second strategy. Indeed, the use of UML Profile as an extension mechanism provides the best compromise to at the same time remain compliant with UML and specialise UML with precise semantics.

5. UML 2.0 Profile for C3

First of all we identify the target meta classes of UML 2.0 meta model which allow to stereotype the structural concepts as well as behavioral ones. The C3 structural concepts component, connector and the configuration are considered as types. Furthermore, those concepts are treated as entities having the same level of abstraction (first class entities). Finally, the external vision of component and configuration concepts is based on a set of ports and the external vision of connector concept is based on a set of roles. Although both component and class concepts of UML 2.0 have the same expressive power, they are used as base for stereotyping respectively the component and connector concepts of C3. The concept state machine of UML 2.0 is used as base for stereotyping the behavioral aspects of the C3 elements. A C3 interfaces is described by a stereotype of UML 2.0 interface «C3Interface».

5.1 Components

UML 2.0 component is the closest concept to the C3 component. So, the former concept will be used as base for stereotyping the later one. Invariant 1 assures that those components have only interfaces through C3 ports and properties. There are no required or provided interfaces which are associated to C3Component. All ports associated with C3component are C3Ports and have port type. A C3 component is described by a stereotype of UML 2.0 component «C3Component» as depicted by Figures 5 and 6.

```
Context Component inv: -- invariant 1
  self.isC3Component () implies
  self.provided → isEmpty and
  self.required → isEmpty
  self.ownedPort →
     forAll (p | p.stereotype = C3Port
     and p.C3PortType = # port)
   self.realisation → isEmpty
   self.sateMachine → size() = 1
```

Figure 5. OCL description for a component

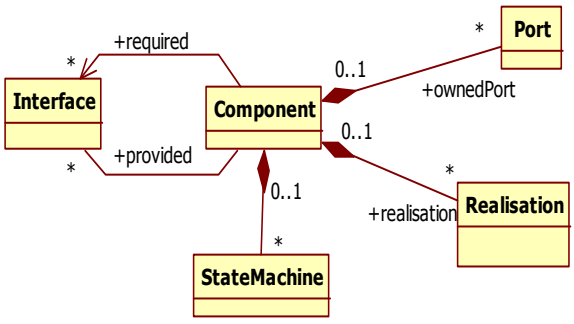

Figure 6. Component Meta Class in UML 2.0 Meta Model

5.2 Ports

Ports identify points of interaction between a component and its environment. UML ports are features of classifiers that specify distinct points of interaction between the classifier and its environment. UML ports have required and provided interfaces. We use a combination of UML port and corresponding required and provided interfaces to express C3's port concept as illustrated by figure 7. Ports can only be used with components and they have only one provided and one required interface.

```
Context  Port  inv:  -- invariant 2
  self.isC3Port () implies
    self.owner.isC3Component () and
   (  self.required  → size() =1 and
     self.provided → size() = 1)
```

Figure 7. OCL constraints for a Port

5.3 Connectors

Representing connectors using UML's assembly connector would be visually appealing, but we would loose expressiveness because C3 connectors may be much more complex than a simple interfaces' match. They can be, for example, a protocol, or a SQL link between two components (a client and a database). Moreover, when reusing components built by different teams it is normal that their interfaces do not match exactly. The connector may provide the required glue between the components and this must be made explicit in the design. In order to represent the concept of connector, which has no semantic equivalent in UML, we use a stereotype of UML class named <<C3Connector>> and that it has no other interfaces than the ones defined through its roles and properties as depicted by Figures 8 and 9.

```
Context  Connector  inv:  -- invariant 3
  self.isC3Connector () implies
      self.ownedAttribute → isEmpty() and
      self.ownedOperation → isEmpty()
  self.provided →  isEmpty  and
      self.required → isEmpty
  self.ownedPort  →
      forAll (p | p.stereotype = C3Port and
      p.C3PortType = # role)
  self.sateMachine → size() = 1
  self.end  →  size() = 2
```

Figure 8. OCL description for a component

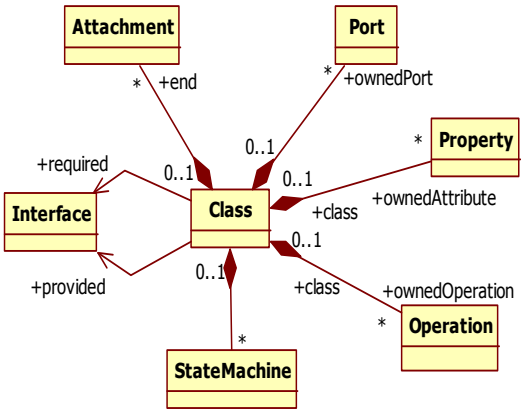

Figure 9. The Meta class Class in UML 2.0 Meta Model

Attachments are represented by stereotype attribute in C3Connector <<C3Attachment>>. An attachment can connect only two C3 elements. Those two elements can only be a connector with a component or a component with configuration. All elements connected by an attachment are C3Port. A C3Attachment can bind one C3Port of type port with one C3Port of type role.

5.4 Roles

In C3, roles are related to connectors the same way as ports are related to components. Thus, it makes sense to represent C3 roles as constrained UML ports, through the use of the <<C3Role>> stereotype as illustrated by Figure 10.

```
Context  Port  inv:  -- invariant 4
   self.isC3Role ()  implies
       self.owner.isC3Connector ()  and
        (self.required  → size() =1 and
         self.provided → size() = 1)
```

Figure 10. OCL constraints for a Port

5.5 Configurations

We introduce stereotypes for modeling the attachments of components to connectors and for C3 configurations.

Stereotype C3Attachment for instances of metaclass association:

- C3 attachments are associations between two elements.

  self.ocltype.end → size() = 2

- One end of the association must be a C3 component.

  Let ed = self.ocltype.end
  ed[1].multiplicity =”1..1” and
  ed[1].class.stereotype = C3Component

- The other end of the association must be a C3 connector

  ed[2].multiplicity =”1..1” and
  ed[2].class.stereotype = C3Connector

Stereotype C3Configuration: A C3Configuration is made up of only C3 model elements.

self.ocltype.elements → forAll (e |
e.stereotype = C3Component or
e.stereotype = C4Connector or)

6. Related Work

Different UML Profiles dedicated for the description of software architecture have been proposed in the literature. For instance, the SAE Architecture Analysis and Design Language (AADL [13]) standard includes UML 1.4 and UML 2.0 Profiles that add the real-time and embedded systems semantics of AADL to UML [14].

In [16] the authors establish an UML 2.0 profile for the ADL ACME. The authors of [15] indicate some weaknesses of this work specially related to the proposed representation of ADL connector in UML2.0 and propose a generic ADL in the form of a UML2.0 profile. In this work, authors use the concept of collaborations provided by UML2.0 to represent ADL connectors.

Oquendo in his paper [12] presents the UML 2.0 Profile for π-ADL, a novel ADL that has been designed in the ArchWare European Project. he presents π-ADL and its UML 2.0 Profile which formally modelling software architectures.

It is expected that multiple profiles for different domains will be defined as specialization of UML 2.0 in the future.

7. Conclusion

C3 introduces the notion of architecture abstractions, which can be components, connectors, and configuration from structural viewpoint. All abstractions are first-class citizens. The UML 2.0 Profile for C3 architecture elements briefly presented in this paper provides a UML-compatible notation for modeling software architecture. This UML 2.0 Profile provides an easy to learn and low cost entry point for describing software architectures.

However, while a connector is regarded as first class design element by architecture community, it has no direct mapping in UML 2.0. Our proposal is to promote connectors to first class architectural element, by representing them as stereotyped components. This seems to be good option, considering that the evolution of Component Based System should provide us with an increasing number of off-the-shelf components. Representing connectors as stereotyped components gives us the extra flexibility to meet this challenge.

The availability in UML 2.0 of components with ports typed by provided and required interfaces has proved to be a step forward in bridging the gap between architectural and design information. This improves the traceability between architectural description and its implementation, using the design as a middle layer between them. This traceability is relevant for keeping the consistency between the architecture, design and implementation of a software system. Our ongoing works in this field are: 1- Implementation of this C3 Profile in UML 2.0 environment with OCL support. 2- Extension of this profile to support advanced concepts like behavioral aspects of C3 elements, nested configurations, architectural styles.

References

[1] Medvidovic N., Rosenblum D.S., Redmiles D.F. and Robbins J.E., “Modeling Software Architectures in the Unified Modeling Language”, *ACM Transactions on Software Engineering and Methodology*, Volume 11, Issue 1, January 2002.

[2] Medvidovic, N. and Taylor, R.N. 2000. “A Classification and Comparison Framework for Software Architecture Description Languages”, *IEEE Transactions on Software Engineering*, Volume 26, Issue 1, Pages 70–93, January 2000.

[3] Riva, C., Xu, J. and Maccari, A. "Architecting and Reverse Architecting in UML", *In Proceedings of the Workshop on Describing Software Architecture with UML, 23rd International Conference on Software Engineering*, Toronto, Canada, 2001.

[4] Jacobson, I., Booch, G. and Rumbaugh, J., *The Unified Software Development Process*, Massachusetts, Addison Wesley, 1999.

[5] OMG: *Unified Modeling Superstructure*, [Electronic Version] from http://www.omg.org/docs/ptc/06-04-02.pdf, 2006.

[6] OMG: *Unified Modeling Language: Infrastructure*, [Electronic Version] from http://www.omg.org/docs/formal/07-02-06.pdf, 2007.

[7] OMG: *Object Constraint Language*, Proposal, OMG Document, (http://www.uml.org/), May, 2006.

[8] Oussalah, M., Amirat, A., and Khammaci, T., "Software Architecture Based Connection Manager", *In Proceedings of SEDE'07*, Las Vegas, Nevada, USA, Pages 194-199, July 2007.

[9] Amirat, A., Oussalah, M., and Khammaci, T., "Towards an Approach for Building Reliable Architectures", *In Proceeding of IEEE IRI'07*, Las Vegas, Nevada, USA, Pages 467-472, August 2007.

[10] Szyperski, C., *Component Software: Beyond Object-Oriented Programming*, 2nd Edition, Addison Wesley, January 2002.

[11] Enrique, J., and Martínez, P., "Heavyweight Extensions to the UML Metamodel to Describe the C3 Architectural Style", *ACM SIGSOFT Software Engineering Notes*, Volume 28, Issue 3, May 2003.

[12] Oquendo, F., "Formally Modelling Software Architectures with the UML 2.0 Profile for π-ADL", *ACM SIGSOFT Software Engineering Notes*, Volume 31, Issue 1, Pages 1-13, January 2006.

[13] SEA *Standard AS5506: Architecture Analysis & Design Language (AADL)*, Embedded Computing Systems Committee, November 2004.

[14] Feiler, H., Lewis, B., and Vestal, S., "The SAE Avionics Architecture Description Language (AADL) Standard: A Basis for Model-Based Architecture-Driven Embedded Systems Engineering", *In RTAS 2003 Workshop on Model-Driven Embedded Systems*, May 2003.

[15] Roh S., Kim K. et Jeon T., "Architecture Modeling Language based on UML2.0", Software Engineering Conference, APSEC'04, *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, 2004.

[16] Goulão M., Abreu F.B., "Bridging the gap between Acme and UML 2.0 for CBD", Workshop at **ESEC**/FSE 2003-Septembre 1-2, 2003.