World Academy of Science, Engineering and Technology
International Journal of Computer and Systems Engineering
Vol:3, No:1, 2009

Cosastudio: A Software Architecture Modeling Tool

Adel Smeda, Adel Alti, Mourad Oussalah, and Abdallah Boukerram

***Abstract*—**A key aspect of the design of any software system is its architecture. An architecture description provides a formal model of the architecture in terms of components and connectors and how they are composed together. COSA (Component-Object based Software Structures), is based on object-oriented modeling and component-based modeling. The model improves the reusability by increasing extensibility, evolvability, and compositionality of the software systems. This paper presents the COSA modelling tool which help architects the possibility to verify the structural coherence of a given system and to validate its semantics with COSA approach.

***Keywords*—**Software Architecture, Architecture Description Languages, UML, Components, Connectors.

I. Introduction

THERE are at least two different techniques of describing the architecture of a software system, either by using object-oriented notations (e.g UML) [1], [4], [5] or by using special notations for software architecture (e.g. Architecture Description Languages ADL) [2], [8]. The two techniques are successively called Object-Based Software Architecture (OBSA) and Component-Based Software Architecture (CBSA).

Actually, UML becomes a standard language for specifying, visualizing, constructing and documenting architectural description concepts. However, with the introduction of UML 2.0 [3] new notations have been constructed and existing ones have been modified to answer software architecture description demands. UML 2.0 provides a suitable base to define UML profiles for software architecture.

In this article, we are interested with building a COSA modelling tool; which is an approach for software architecture based on object oriented modeling and component based modeling [3]. Recently, concepts of COSA are mapped into UML 2.0 [6]. Using the capacities of UML profiles and models technological space (MTS), also known as MDA technological space [10], we define a plug-In called *COSAStudio* for software architectures modelling. The main objective of this plug-In is to show the ability for modelling complex applications. The plug-In offers to the architects the possibility to verify the structural coherence of a given system and to validate its semantics with COSA approach.

Adel Smeda is with University of Al-Jabel Al-Gharbi, P.O. Box 64200, Gharian, Libya (e-mail: Adel.Smeda@univ-nantes.fr).

Adel Alti is with Computer Science Department, Engineering Faculty, Ferhat ABBAS University of Sétif, 19000 Sétif, Algeria.

Mourad Oussalah is with LINA, University of Nantes, 2, Rue de la Houssinière, BP 92208, 44322 Nantes, France.

Abdallah Boukerram is with Department of Computer Science, University F. Abbas, 19000 Sétif, Algérie.

II. COSA: Component-Object based Software Architecture

Like it is approved in several previous works on software architecture description, it is possible to represent software architectures using specific architecture description languages (ADLs) which are component-based languages (Acme [11], Rapide [12], etc.) or using object-based languages (UML [3]).

COSA (Component-Object based Software Architecture) is hybrid model, based on both object and component modeling to describe software systems [1]. The basic principal of this model is to base on architectural description languages formalism extended with object-oriented concepts and mechanisms to specify software architectures. A major advantage of COSA is that, it defines and manipulates connectors as first class entities by explicitly define them. In COSA, components, connectors and configurations are defined as classes which can be instantiated to define different architectures.

In addition to instantiation mechanism, basic elements of COSA can be beneficiated also of others object concepts and mechanisms, such as encapsulation, composition, reuse and specialisation. COSA architectures description approach is not based on any particular notation or language, but it is considered as a metamodel which describe a concept set of vocabulary and modelling elements used to express a software architecture description. This allows more simplicity, extensibility, and genericity in software architecture description.

Basic concepts of the architecture COSA are components, connectors, configurations, interfaces, constraints and functional (and non-functional) properties as shown in Fig. 1 [3].

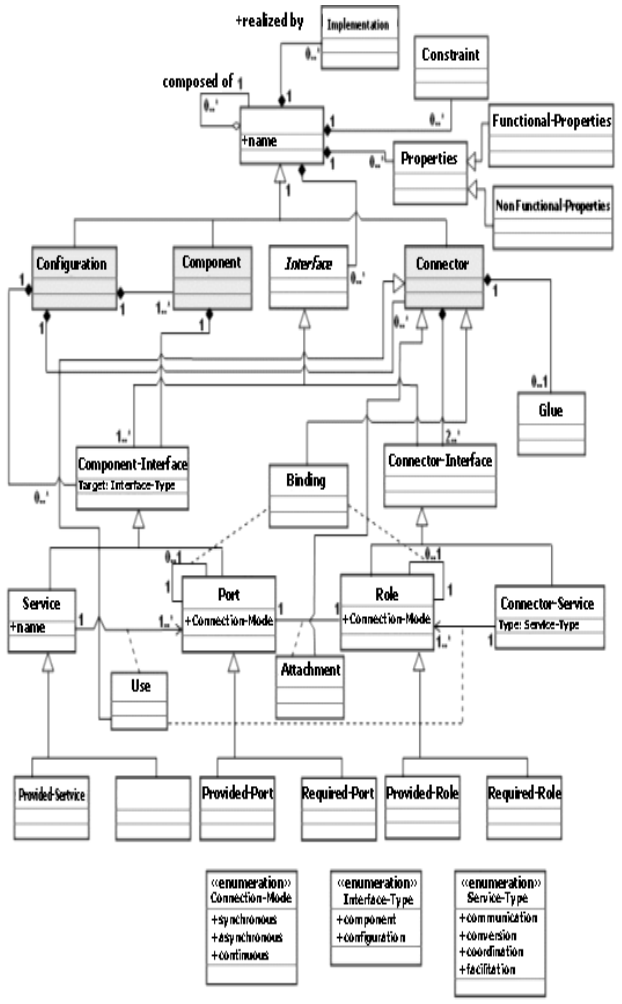

Fig. 1 Meta-model of the COSA approach

The key role of *configurations* in COSA is to abstract the details of different components and connectors. A configuration has a name and defined by interfaces (ports and services), A configuration has a name and defined by an interface (ports and services), which are the visible parts of the configuration and support the interactions among configurations and between a configuration and its components.

Components represent the computational elements and data stores of a system. Each component may have an interface with multiple ports and multiple services. The interface consists of a set of points of interactions between the component and the external world that allow the invocation of the services. A component may have several implementations. A component can be primitive or composite [3].

Connectors represent interactions among components; they provide the link for architectural designs. A COSA connector is mainly represented by an interface and a glue specification [3] [13]. In principle, the interface shows the necessary information about the connector, including the roles, service type that a connector provides (communication, conversion, coordination, facilitations). Connectors can be composite or primitive.

Interfaces in COSA are first-class entities. They provide connection points among architecture elements. Likewise, they define how the communication between these elements can take place. A component/configuration interface's connection point is called *port* and a connector interface's connection point is called *role*. In addition to ports and roles interfaces have services that express the semantics of the element with which they are associated.

Properties represent additional information (beyond structure) about the parts of an architectural description. Typically they are used to represent anticipated or required extra functional aspects of an architectural design. There are two types of properties: functional properties and non-functional properties. Functions that relate to the semantics of a system and represent the requirements are called functional properties. Meanwhile non-functional properties represent additional requirements, such as safety, security, performance, and portability.

Constraints are specific properties, they define certain rules and regulations that should be met in order to ensure adherence to intended component and connector uses.

III. COSAStudio: A Software Architecture Modeling Tool

This section presents the development of the model COSA in Rational Software Modeler (RSM) for Eclipse [14]. For this, we chose to use the mechanisms of creating profiles of RSM. Next we focus on what tooling is needed to verify the structural coherence of a given system and to validate its semantics with COSA approach. After that we present an example from the tool and we end up with a comparison of the tool with other existing tools.

A. Mapping Cosa Model into UML 2.0

Mapping architectural elements: The architectural element is a basic concept that defines all COSA architectural concepts. This concept is not defined explicitly in UML. The UML profile must include a «COSAArchitecturalElement» stereotyped class to represent COSA architectural element. This class may have properties and constraints and can be implemented by another class.

Mapping components, connectors and configurations: Components and connectors are treated differently in COSA. Components are abstractions that include mechanisms of computation and connectors are abstractions that include mechanisms of communication. Meanwhile configurations are graphs of components' and connectors' types. Our choice is based on using UML components to represent COSA components and configurations and each one is associated with a stereotype. COSA connectors are represented by a stereotype corresponds to UML class.

A UML 2.0 component is as expressive as a UML class and provides services through ports, these services must belong to an interface. COSA component types correspond to UML 2.0 component types, and COSA component instances correspond to UML component instances. The UML Class defines and specifies connectors in COSA. A class can contain ports as points of interaction. COSA Connector must have at least a port stereotyped by «ConnectorInterface» and contains single

Glue. A COSA connector defines the behavior of each of the interacted parties. How these behaviors are combined to form a communication is described by the glue. In UML the AssociationClass concept is relative to the COSA glue concept. A UML port, which has at least two interfaces (provided and required), matches COSA connector roles. An important aspect of COSA architecture is to offer a graph of components and connectors types called configurations. Since a UML component can contain subcomponents and subclasses, the configurations of COSA are mapped into UML components.

Mapping ports and roles: The class *Port* of UML represents COSA components' interface and COSA connectors' interface in the UML metamodel 2.0, but they remain well distinguished by stereotypes assigned to each one of them.

Mapping specific connectors: A UML delegation connector corresponds to the COSA concept Binding, which is used to bind an external interface into an internal interface. A UML assembly connector corresponds to the COSA concept Attachment. Attachments define the link between a provided port (or a required port) and a required role (or a provided role).

B. Implementing the Modeling Tool

Once we have the COSA Meta-model mapped into an UML model, we can take advantage of the tools developed around Rational Software Modeler. The UML 2.0 metamodel for COSA is implemented in IBM Rational Software Modeler for Eclipse 3.1 [14]. This visual modeling tool supports creating and managing UML 2.0 models for software applications, independent of their programming language, and provides a common language for describing formal semantics with OCL language and have been used successfully to define profiles and to valid models of complex systems.

The Plug-In is developed with three levels of abstraction. In the high level, the meta-model of COSA with all tagged values and its OCL 2.0 constraints is defined by the UML 2.0 profile. This diagram plays an important role in the second level when it is used by to model of software architecture. Once we ensure that the given model complies to the semantic constraints defined by the profile, a set of instances for the types are defined and evaluated in this level.

The main objective of this plug-In is to show the ability to apply the profile for complex applications. The plug-In offers to the architects the possibility to verify the structural coherence of a given system and to validate its semantics with COSA approach. First we create a components diagram in UML 2.0 for the described system and then we add the needed OCL constraints. After that, the model is evaluated by the profile.

COSA is defined in UML 2.0 by using the mechanisms of creating profiles of RSM. Fig. 2 shows the profile with its stereotypes, all tagged values and OCL 2.0 constraints expressed in the meta-model UML 2.0 -EMF (Eclipse Modeling Framework).

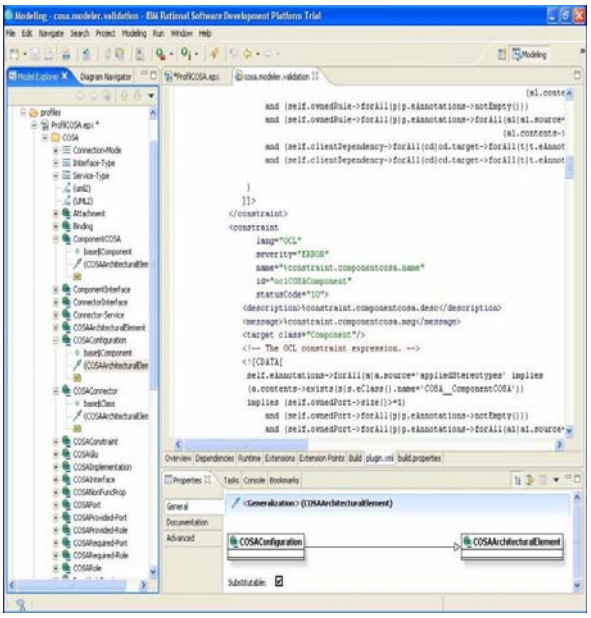

Fig. 2. The COSA-UML profile in RSM for Eclipse 3.1

C. Final Evolution Results

For well known client server architecture, we elaborated the system by a components diagram and OCL constraints. Once, COSA profile is applied from the Select Profile dialog, shown in Fig. 3, all its stereotypes will be available, applied, and contributed by the tagged-values. The model then checks to remove any constraints violation.

Fig. 3 Selecting the COSA Profile for the Client-Server system

The model is tested and validated with the semantic constraints defined by the profile, a set of instances (ex: arch-1) for the types are defined and also evaluated for the final mapped system as shown in Fig. 9.

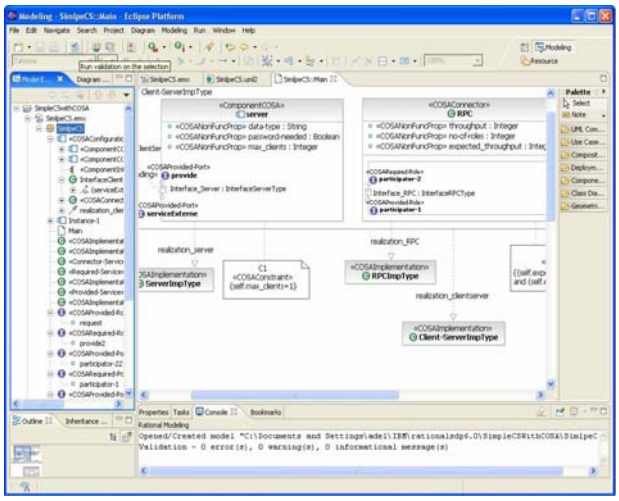

Fig. 4 Validating Client-Server system in UML 2.0 with RSM

D. Comparison and Remarks

Many, if not most of, architectural design and analysis tools require a representation for describing, storing, and manipulating architectural designs. Unfortunately, developing good architectural representations is difficult, time consuming, and costly. COSA can mitigate the cost and difficulty of building architectural tools by providing a standard UML language and toolkit to use as a foundation for building tools. COSA provides a solid, extensible foundation and infrastructure that allows tool builders to avoid needlessly rebuilding standard tooling infrastructure. Further, COSA's origin as a generic language allows tools developed using COSA as their native architectural representation to be compatible with a broad variety of existing architecture description languages and toolsets with little or no additional developer effort. Finally, COSAStudio provides an easy way to describe complex software architectures.

IV. Conclusion and Perspectives

In this article we have presented a COSA software architecture which describes software architectures in an abstract manner. Components and connectors in COSA have the same level of abstraction and defined explicitly. We think the presented approach seems very interesting particularly when considering the increasing use of UML and model driven development. We have also shown how this model can be implemented as a plug-in for Rational Software Modeler [14]. For this, we have created an UML 2.0 meta-model. This meta-model allows us to model any architecture that conforms to COSA language specification. It opens the door to other tools that can take advantage of architectural models in order to conduct architectural analysis, transformations, etc. Another useful feature is the extensibility of the COSA meta-model to include new connectors' types [15] (Attachment Connector, Expansion-Compression Connector, Composition-Decomposition Connector, Service-Connector).

References

[1] G. Booch, J. Rumbaugh., I. Jacobson, The Unified Modeling Language User Guide. Addison-Wesley Professional, Reading, Massachusetts, (1998).

[2] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, , R. Nord, J. Stafford, Documenting Software Architectures: Views and Beyond. Boston, MA, Addison-Wesley, (2002)

[3] M. Oussalah, A. Smeda, T. Khammaci, An explicit definition of connectors for component based software architecture. In: Proceedings of the 11th IEEE Conference Engineering of Computer Based Systems, Czech Republic (May 2004)

[4] I. Jacobson, Object-Oriented Software Engineering: A Use Case Driven Approach. Addison Wesley Professional. (1992).

[5] OMG, Unified Modeling Language Specification V.1.4. http://www.omg.org/docs/formal/01-09-67.pdf , Sept 2001.

[6] Alti A., Khammaci T., Smeda A., Representing and Formally Modeling COSA software architecture with UML 2.0 profile. IRECOS Review, 2007, 2(1): 30-37.

[7] Garlan D., Monroe R.T., and While D., Acme: Architectural Description of Component-Based Systems. G.T. Leavens and M. Sitaraman, Eds, Cambridge University, 2000.

[8] Medvidovic, N., Taylor, R.N.: A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Transactions on Software Engineering, Vol. 26. N°. 1. 2–57, 2000.

[9] Amirat A., Oussalah M., "Enhanced Connectors to Support Hierarchical Dependencies in Software Architecture", 5th NOTERE'08 International Conference on New Technologies in Distributed Systems, Lyon, France, Voluome.1, pp. 252-261, June 23-27, 2008.

[10] Moore B., Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework, I. Redbooks, 2004.

[11] Garlan D., Monroe R.T., and While D., Acme: Architectural Description of Component-Based Systems. G.T. Leavens and M. Sitaraman, Eds, Cambridge University, 2000.

[12] Luckham D.C., Augustin L.M., "Specification and Analysis of System architecture using Rapide," IEEE Transactions on Software Engineering, 1995, 21(1): pp. 336 – 355.

[13] Smeda A., Oussalah M., and Khammaci T., "A Multi-Paradigm Approach to Describe Complex Software System", WSEAS Transactions on Computers, Issue 4, Vol., 3, pp. 936-941, October 2004.

[14] Rational Software Modeler, http://www-128.ibm.com/developerworks/downloads/r/rswm

[15] Amirat A., Oussalah M., "Enhanced Connectors to Support Hierarchical Dependencies in Software Architecture", 5th NOTERE'08 International Conference on New Technologies in Distributed Systems, Lyon, France, Voluome.1, pp. 252-261, June 23-27, 2008.