An Environment for Describing Software Systems

ADEL SMEDA[*], ADEL ALTI[**], ABBDELLAH BOUKERRAM[**]

[*]University of Al-Jabel Al-Gharbi, P.O. Box 6473, Gharian, Libya
Adel.Smeda@univ-nantes.fr

[**]Computer Science Department, Faculty of, Engineering, Ferhat ABBAS University of Sétif, 19000 Sétif, Algeria
altiadel2002@yahoo.fr , aboukerram@univ-setif.dz

Abstract. Describing the architecture of complex software systems need a comprehensive models and complete tools. The description of software systems can be achieved by using an architecture description language (ADL) or an object oriented modeling language. In this article, we show how we can build a hybrid model to describe the architecture of software systems. This model is based on the two approaches. First we define a metamodel for software architecture, next based on this metamodel we implement an environment for describing the architecture of software systems.

Key-words: Software Architecture, COSA, Architecture Description Languages, UML 2.0 Modeling Language, Component, Connector

1 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], [17]. 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 [5] 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 of building a modeling tool for software architecture based on object oriented modeling and component based modeling called COSA [13][3]. Recently, the 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 a 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.

2 COSA : Component-Object based Software Architecture

COSA (Component-Object based Software Architecture) is hybrid model, based on both object and component modeling to describe software systems. 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 simplicity, extensibility, and genericity in software

architecture description. Figure 1 presents the model of the COSA software architecture.

3 The metamodel COSA (Architecture modeling)

COSA supports number of architectural elements including configurations, components, connectors, interfaces, properties and constraints [3]. These architectural elements are types that can be instantiated to construct several architectures.

3.1 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. Each interaction point of a component is called a *port*. Ports are named and typed. We distinguish between required and provided ports. Each port can be used by one or more services. Components semantics is modeled to enable evolution, analysis, enforcement of constraints and consistent mappings of architectures from one level of abstraction to another. The structure of component is the specification of its required and provided ports. The behavior of a component is the specification of its required and provided services exchanged with its environment. A component may have several implementations. A component can be primitive or composite [3].

3.2 Connectors

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 [4]. In order to enable proper connectivity of components and their communication, a connector should export as its interface those services it exports.

COSA refers to connector interactions points as *roles.* Explicit connection (attachments) of components ports and connector roles is required in an architecture configuration. Roles are named and typed and are in many ways similar to component ports. In principle, the interface shows the necessary information about the connector, including the roles, service type that a connector provides (communication, conversion, coordination, facilitations). Connector services are described inside the glue code [4]. Therefore, a connector's interface is a set of interactions points between it and components/configurations attached to it.

3.3 Configuration

Architecture configuration has a name and defined by interfaces (ports and services), which are the visible parts of the configuration and support the interactions among configurations and between a configuration and its components. Configuration is connected graph of components and connectors that describe architectural structure. This information is needed to determine whether appropriate components are connected, their interfaces mach, connectors enables proper communication, and their combined semantics results in desired behavior. The key role of *configurations* in COSA is to abstract the details of different components and connectors. They depict the system at a high level that can be potentially be understood by actors with various levels of technical expertise and familiarity with the problem at hand.

For more clarity, in COSA model each component or connector is perceived and handled from the outside as primitive element. But their inside can be real primitive elements, or composite with a configuration which encapsulates all the internal elements of this composite. These configurations are first-class entities. A configuration may have ports similar to components ports, and each ports is perceived like a bridge (*binding*) between the internal environment of the configuration and the external one. In COSA this binding is realised using connectors. Generally configurations can be hierarchical where the internal components and connectors can represent sub-configurations with their proper internal architectures.

3.4 Interface

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. To establish connections between elements we use required/provided ports for component and configuration elements and required/provided roles for connector elements and we assign the

services to each port and role with the necessary set of constraints to be respected during the connections. From conceptual view ports, roles and services are concrete classes inherited from the interface abstract class as shown in Figure 1. Also, at modeling level we use the cardinality to describe the multiplicity of each connection between architectural elements. This cardinality express the number of ports associated with components and configurations and the number of roles associated with connectors. Each port or role is considered as a channel to carry in/out required/provided services exchanged with elements of the environment.

3.5 Properties

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 [11]. 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.

3.6 Constraints

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.

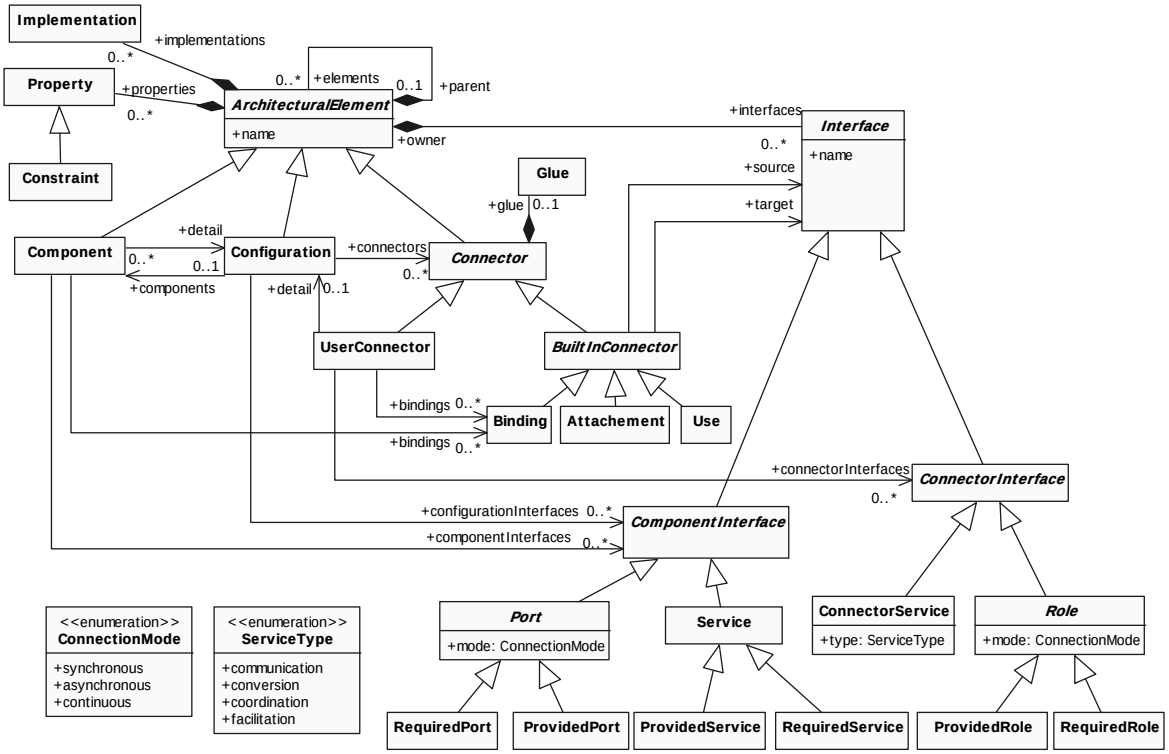


Figure 1. COSA Meta-model

3.7 The metamodel of COSA Instances (application modeling)

In the real word application is an instance of the architecture model. The architect has the possibility to select and instantiate COSA architectural elements as many times as he needs to describe a complete application. Instances are created from types that are defined in COSA metamodel. Elements are created and assembled with respect to the different constraints defined at COSA architecture models. Figure 2 shows the COSA instance meta-model.

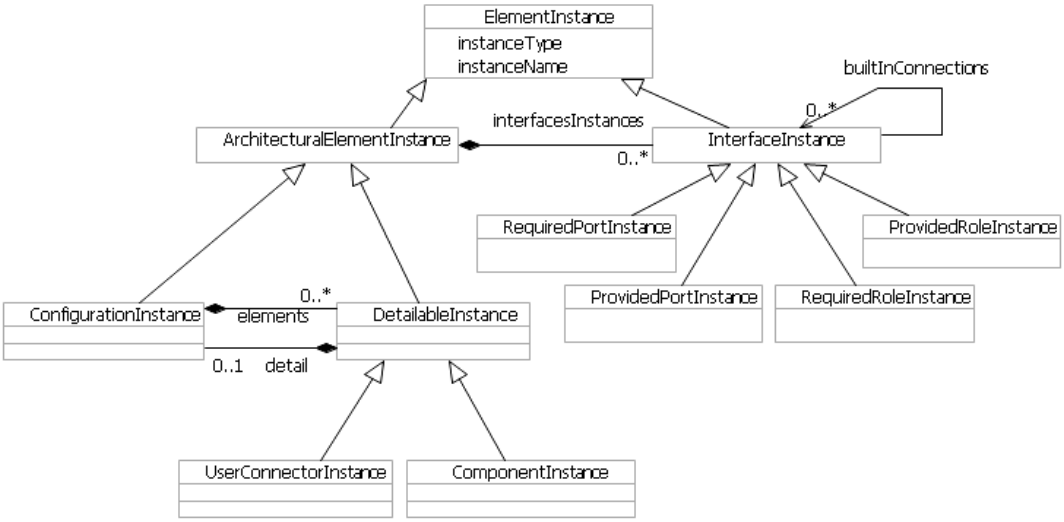


Figure 2. COSA Instance Meta-mode

4 Development of COSA Modeling and Instantiating Tool

This section presents the development of the model COSA in Eclipse. For this, we chose to adopt a modeling point of view as described in [5, 6], since the issue is a language and a modeling problem rather than an architectural point of view. First we explain how we implement this approach to convert the COSA architectural description language into a modeling language that can be processed by tools. Next we focus on what tool is needed to realize this, after that we present an example.

4.1 Definitions

Our job is to provide a tool that allows the production of architectural models, in accordance with the COSA meta-model. These activities (i.e. modeling and meta-modeling) are part of Model Driven Engineering concerns. Generally, Model Driven Engineering community considers four different levels of modeling [6, 7]: "reality" level (M_0), models level (M_1), meta-models level (M_2), meta-meta-models level (M_3).

Each level uses a syntax (a language) that is defined at its higher level, except the level M_3 which is defined by itself. The relations between elements of two levels are called "meta", and they are strictly "syntactical" links. Semantics of levels are absolutely not correlated (i.e. there is no semantic correlation between a UML diagram and the UML meta-model for instance[1]).

Each level deals with the modeling of its lower level, except M_0 which is considered as the "reality" and therefore can not represent something else. Note that "reality" is a concept that is adapted according to the general domain and/or technological space on which we focus. For the rest of this article, we call this approach *models technological space (MTS)*, also known as MDA technological space [7].

Architecture meta modeling presented in [8] describes another technological space based on the initial works of OMG (see Figure 3). It defines an architecture with four levels:

- M²A (Meta-meta-architecture) level: correspond to meta-meta-architecture level. It is an architecture of all architectural concepts. At this level, Smeda et al. [8] introduced MADL a language for Meta Architectural Description. MADL is similar to MOF but it is a component oriented.
- MA (meta-architecture) level: contains architectural meta-elements such as Component, Connector, allowing the description of Architectures. COSA is defined at this level. All operations undertaken at this level are always in conformance with the top level of pyramid.
- A (architecture) level: contains architectural elements describing architectures. These elements are instances of meta elements.
- A_0 (application) level: contains instances architectural types. At this level the developer has the possibility to select and instantiate elements any times as he needs to describe a complete application. Elements are created and assembled with respect to the different constraints defined at architecture level.

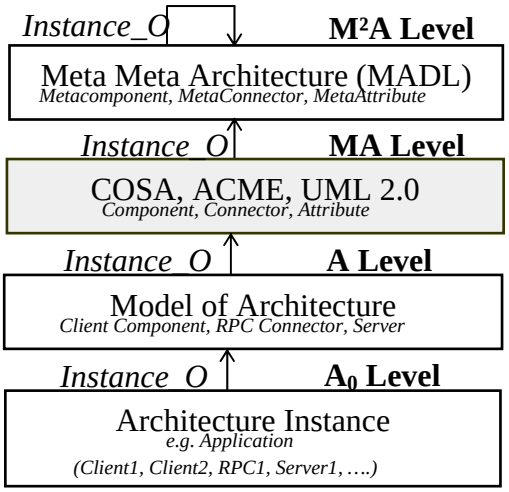


Figure 3. Architectural technological space.

For the rest of this article, we call this approach *architectural technological space (ATS)*. In ATS, the passage from one level to its higher level is achieved by an « instanceOf » relation. Contrary to the previous approach, this relation has an important semantic role. Due to this fact, we should distinguish between these two technological spaces. MTS is appropriate for models tooling developers based on model processing (such as MDA approach, models transformations, Domain Specific Languages, etc.). ATS is adapted for anyone interested of architectural abstractions, and wants to have a strict distinction between an architecture (M_1 level) and an application (M_0 level).

[1] Defining a UML model for UML meta-model is just a special case.

Since our intention is to create a tool to implement the meta-model COSA, we adopt the "language" point of view for this purpose. In the tooling development point of view, we don't focus on the semantics of ATS, but we consider it as the "reality" that we want to model.

Given this organization, we focus on the different stakeholders (actors) that will act on the architecture process as given in [8]. At M_0 we find only concerns. We consider that every work is always some kind of a model and therefore situated at M_1. Figure 5 introduces the different actors that intervene in the process (Architecture Language Designer, Architecture Language Developer, and Architect Developer).

As an architecture language developer, our work is to provide an eCore compliant meta-model for COSA from the COSA meta-model that is created by the Architecture Language Designer. This highlights the following remarks:

- Since eCore is an implementation of MOF, it is not as rich as UML. COSA meta-model uses number of UML specific constructions that are not available in eCore.
- COSA meta-model has number of constraints that are applied to architectural elements. eCore does not support this directly. The solutions to this problem are discussed in the next section.

4.2 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).

4.3 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 with 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 a 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.

4.4 Final evaluation results

We have implemented a full modeling application tool based on COSA metamodel. It has two options to generate the graphical editor: from an existing COSA-Ecore model as XMI file or from a new COSA architecture as a COSA file. The instance diagram is then elaborated using the JET (*Java Emitter Templates*) as the language for the parameterization of the interfaces, components, connectors and configurations which allows the use of Java code in the parameterization process. Once we have selected correctly the architectures as an Ecore file, the COSA instance editor is generated and opened as the model instance contains graphical information. Figure 5 presents an overview of the graphical editor of the Client-server architecture with the different views: COSA instance editor with its palette, the Outline, and the Properties view. If we look at the *"palette"* and we will discover that all the elements of the Client-Server architecture model are presented, so we have just to drag and drop the appropriate element into the area that contains the Client-Server application diagram to describe a complete application. Figure 5 shows the eCore model of the Client-Server example.

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 figure 6.

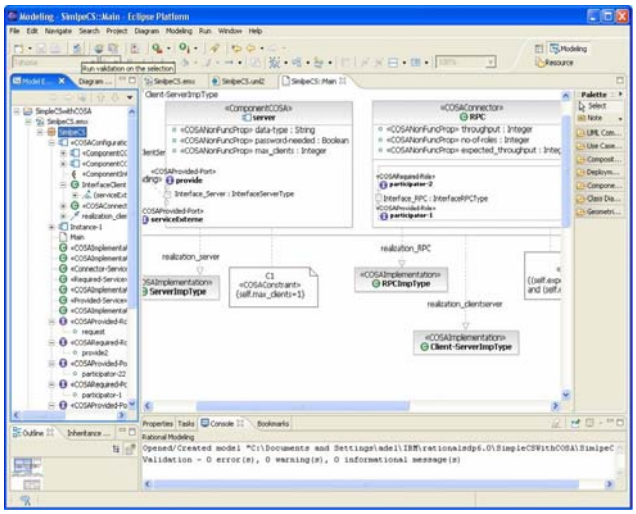
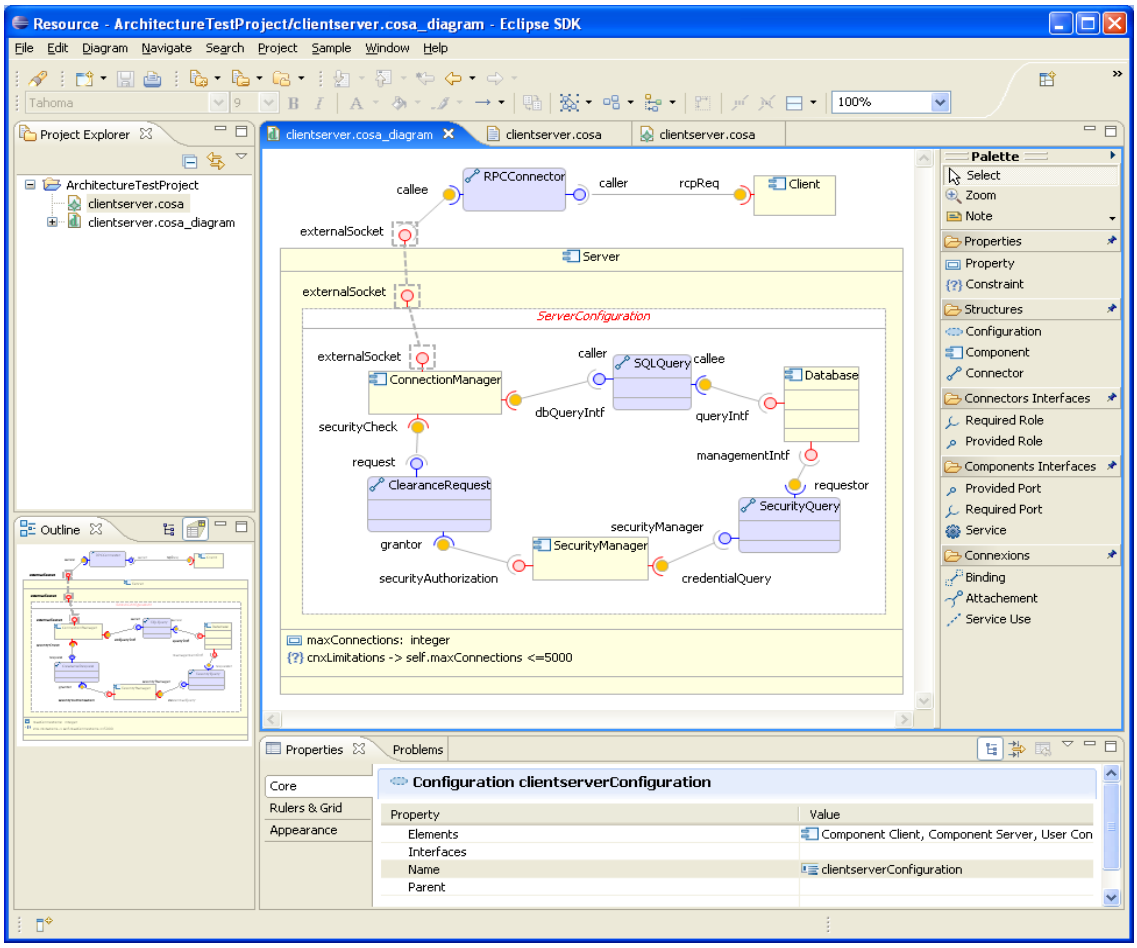


Figure 6. Validating Client-Server system in UML 2.0 with RSM



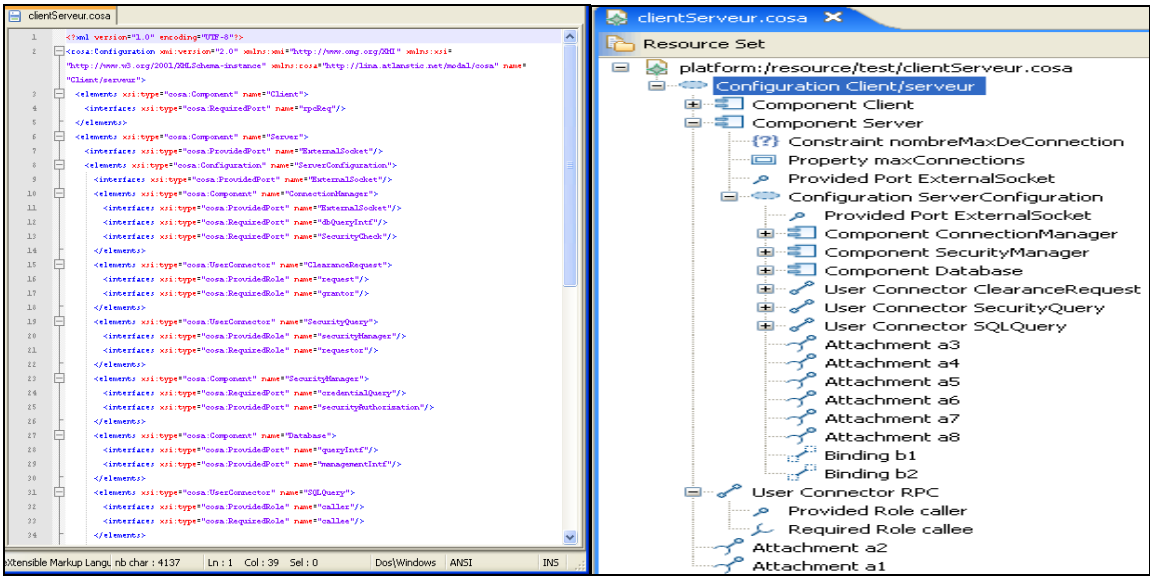


Figure 4: Different views of the same COSA Client/Server Architecture.

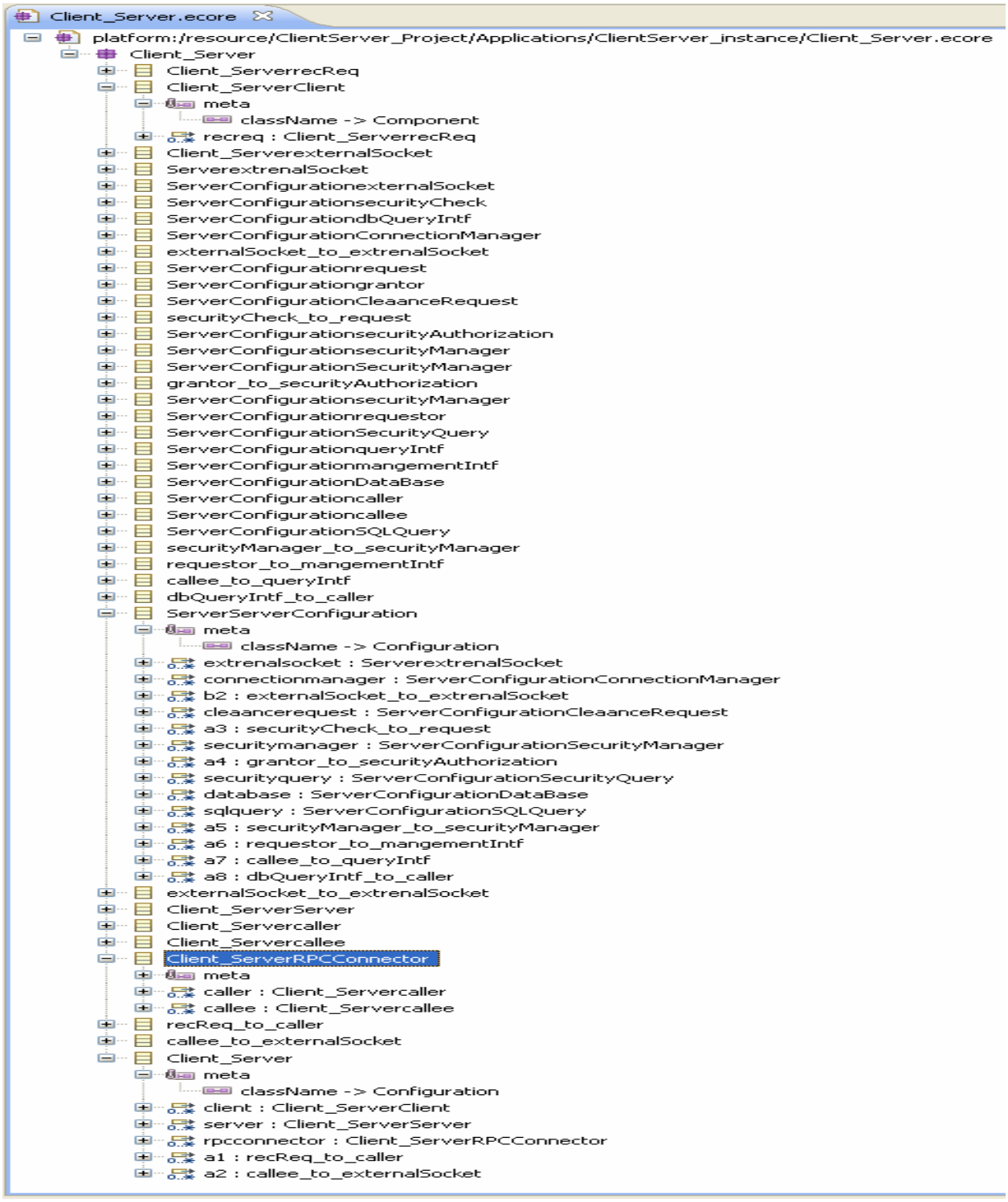


Figure 5. The eCORE model of the Client-Server system

5 Conclusion and perspectives

In this article we have presented a multi-paradigm approach for software architecture based on object-oriented modeling and architectural description (COSA: Component-Object based Software Architecture). It describes systems as a collection of components that interact with each other using connectors. In COSA, components and connectors are defined in configurations which describe the topology of the system. We have also shown how this model can be implemented as a plug-in for Eclipse. For this, we have created an eCore meta-model from the original UML COSA 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.

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] Z. PANIAN, "*Recent Advances in Data Engineering and Management*", in Proceedings of the 8th WSEAS International Conference on Telecommunications and Informatics (TELE-INFO'09), May 30- June 1, 2009, Istanbul, Turkey.

[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.

[16] Aldrich J., Chambers C., Notkin D., "ArchJava: Connecting Software Architecture to Implementation", Proceedings of the 24th International Conference on Software Engineering (ICSE'02), Orlando, USA, 2002.

[17] M. Oussalah, N. Sadou, D. Tamzalit, SAEV, *"a Model to ensure a Static and Dynamic Software-Architecture Evolution*", WSEAS TRANSACTIONS on SYSTEMS, Issue 5, Volume 4, May 2005