Journal of Computer Science 1 (4): 454-460, 2005
ISSN 1549-3636

Meta Architecting: Towered a New Generation of Architecture Description Languages

Adel Smeda, Tahar Khammaci and Mourad Oussalah
LINA, Université de Nantes, 2 Rue de la Houssinière, BP 92208, 44322 Nantes Cedex 03, France

Abstract: The techniques of meta-modeling and meta-levels have become a mature concept and have been largely used to solve real problems in programming languages, distributed environments, knowledge representation, or data bases. In this article it is shown how the same techniques can be applied in component-based software architecture. It also shown the need to propose mechanisms of reflexivity within the domain of software architecture meta-modeling. The outcome of this is a meta-meta-architecture with a minimal core whose finality is to define meta-components, meta-connectors and meta-architectures. Call this meta-meta-architecture MADL (Meta Architecture Description Language).

Key words: Software architecture, component-based systems, architecture description languages, metamodeling

INTRODUCTION

In the domain of Knowledge Representation[1] one speaks about meta-knowledge to evoke knowledge relating to knowledge, or meta-model for a model representing a model, etc. In the context of software architecture[2] meta-modeling is an act of modeling that is applied to an architecture. The result of such an act of modeling, i.e. the use of an architecture A to establish an abstraction of a system S, is called the Architecture (A) of the system S. Similarly, the meta-architecture of an architecture is itself an architecture, which does not model a final application but an architecture. When the act of modeling is applied to software architectures, it is called meta-architecting. A meta-architecture is thus a formal or semiformal Architecture Description Language (ADL)[3] that permits to describe particular systems, called architectures. Meta-Architecture (MA) itself is an architecture and thus in a more general way a system, which can be described. One then can define the architecture of a meta-architecture by the meta-meta-architecture (M^2A). As in any recurring modeling, it is advisable to stop by a reflexive architecture, i.e. auto-described architecture. The number of levels imports little, but it seems that three levels of modeling are sufficient within the domain of software architecture engineering where the M^2A serves as an auto-modeler and as a modeler of ADLs.

In object-oriented modeling, this self-modeling is generally implemented by the concept of meta-class which guarantees the homogeneity of the concepts and the extensibility of the system. It is a question of conceiving an architecture by itself or using an architecture to define or conceive another architecture.

Meta-architecting of an architecture can be a tool to define, comment, document, compare architectures, in particular semiformal architectures. It is about describing an architecture by its conceptual diagram, resulting of a step of specification using a meta-architecture that most of the time itself is a semiformal. This diagram then constitutes a document of explanation and/or documentation of the architecture.

Meta-architecting can also be a means of formalization for semiformal architectures. Formal ADLs are based on mathematical theories ensuring obtaining exact specifications[4]. However the formalisms of these ADLs can discourage the designers. In additional, formal ADLs are not easily comprehensible for non trained designers, thus it is very useful to transfer a semiformal specification to a formal specification. The approach consists of specifying, once and for all, the concepts of semiformal architecture in a formal specification. One then obtains a meta-architecture of the semiformal architecture which can then be directly used.

For brevity, it can say that meta-architecting is a good way to:

- Standardize: Architectures are based on well-defined semantics. These semantics are provided by means of meta-architectures. Each architecture must conform to a meta-architecture, which specifies a specific way to define architectures. Describing different architectures using the same meta-architecture confers on the meta-architecture a role of standardization for at least the architecture that it describes.

Corresponding Author: Adel Smeda LINA, Université de Nantes 2, Rue de la Houssinière, BP 92208, 44322 Nantes Cedex 03, France, Tel: +332 51125963 Fax: +332 51125812

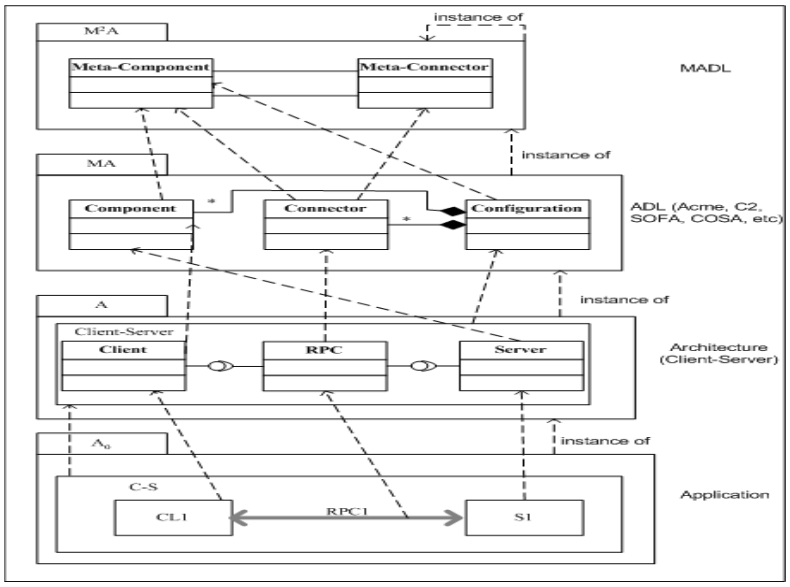

Fig. 1: Applying the four layers of OMG in software architecture

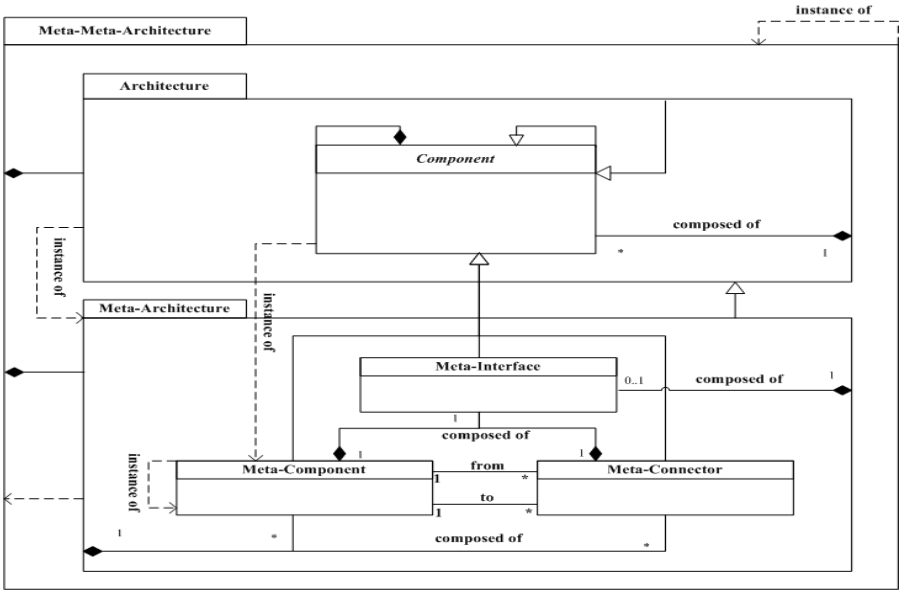

Fig. 2 : The MADL model

- Compare: Meta-architecting is a good tool to compare various architectures. Indeed, describing various architectures with the same formalism, facilitates their comparison and analysis.
- Define and integrate several architectures, therefore, supports and makes the interchange of architectures among ADLs easier.
- Map ADLs into other modeling techniques, e.g. UML[5].

MADL: Meta Architecture Description Language : In object-oriented modeling, self-modeling is generally implemented by the concept of meta-class which guarantees the homogeneity of the concepts and the extensibility of the system. It is a question of conceiving an architecture by itself or using an architecture to define or conceive another architecture. Meta-architecting of an architecture is a technique to define, comment, document, compare architectures, in particular semiformal architectures. It is about describing an architecture by its conceptual diagram,

resulting of a step of specification using a meta-architecture that most of the time itself is a semiformal. MADL is intended to be a meta-mate-architecture that defines meta-architecture. It is based on the Y architecture, hence we have three main elements in MADL: meta-component, meta-connector and meta-configuration (or meta-architecture).

The four abstraction layers of software architecture : The four metamodeling layers of OMG (application layer, model layer, meta-model layer and meta-meta-model layer[7]) can be applied in software architecture and the outcome of this is an architecture with four levels of abstraction represents the different architecture levels, starting from the definition of the meta-meta-architecture to the application level. Consequently, one can see four abstraction levels in software architecture: meta-meta-architecture level, meta-architecture level, architecture level and application level (Fig. 1)

Meta-meta-Architecture level (M^2A): Provides the minimal elements of architectural modeling. It is represented by three basic elements: Meta-component, Meta-connector and Meta-architecture. These three elements are the base for defining different meta-architectures. A meta-meta-architecture conforms to itself (instance of itself). The basic concepts of a meta ADL are represented in this level.

Meta-Architecture level (MA): Provides the basic modeling elements for an Architecture Description Language (ADL): Component, Connector, Architecture, Ports, Roles, etc. These elements are the base for defining different architectures. Meta-architectures conform to meta-meta-architectures. In the scope of a conformity relation, each element of MA is associated with an element of M^2A. For example, in Fig. 1 component is associated with meta-component.

Architecture level (A): In this level, various types of components, connectors and architectures are described. Architectures conform to meta-architectures (ADLs), therefore, each element of A is associated with an element of MA. For example, in Fig. 1, Client and Server are components, RPC is a connector and Client-Server is a configuration.

Application level (A_0): Allows us to describe applications. An application is seen as an assembly of instances of types of components, connectors and architectures. Applications conform to architectures. Each element of A_0 is associated with an element of A. For example in Fig. 1, CL1 is an instance of client, S1 is an instance of server, RPC1 is an instance of RPC and C-S is an instance of client-server.

The structure of MADL: The meta-meta-architecture must be a minimal generic core whose finality is to define meta-meta-architecture elements, which define type elements for meta-architectures. It introduces the concepts: meta-components, meta-connectors and meta-architectures needed to manipulate and to define architectural concepts (structural and behavioral). It is organized these meta-concepts in a meta-meta-architecture called MADL (Fig. 2).

MADL is organized in three packages: Meta-Meta-Architecture package, Meta-Architecture package and Architecture package.

Meta-meta-architecture package : To define an architecture we need a meta-architecture and to define a meta-architecture we need a meta-meta-architecture, therefore, Meta-meta-architecture package composed of Meta-architecture package and Architecture package. Meta-meta-architecture package holds all the concepts needed to define meta-architectures and architectures. Meta-meta-architecture does not conform to another architecture, but acts as its own meta-architecture. Similarly, each element of Meta-meta-architecture has to be associated with another element of Meta-meta-architecture, to respect the auto-conformity relation. For brevity, Meta-meta-architecture is an instance of itself.

Meta-architecture package : To define architectures we need a meta-architecture, so Meta-architecture classifies and defines architectures. Architectures contains components and connectors, therefore, Meta-component and Meta-connector are parts of Meta-architecture (hence, each component and connector of level MA must be a part of an architecture of level MA). Meta-architecture is an Architecture, that is why Meta-architecture inherits Architecture. To permits architectures of level MA to have interfaces Meta-architecture is composed of Meta-interface. Meta-architecture conforms to the definition of Meta-meta-architecture, i.e. it is an instance of Meta-meta-architecture. Meta-architecture is composed of the following meta-elements:

- Meta-component. It is a meta architectural element that classifies and defines constructs of computation and state for level MA. Meta-component is nothing but a component, so it inherits component. Meta-component is a part of (has a composite relation with) the package Meta-architecture. To respect the reflexivity principle, which MADL is based on, Meta-component is an instance of itself.
- Meta-connector. It is a meta architectural element that classifies and defines constructs of interactions among components of level MA. Meta-connector is a part of (has a composite relation with) the package Meta-architecture.

- Meta-interface. It is a meta architecture element that classifies and defines interfaces. It is a part of (has a composite relation with) the package Meta-component, Meta-connector and Meta-architecture, therefore defines interfaces for component, connectors and architectures of architectures of level MA. It is assigned an interface to Meta-architecture in order to make it feasible for architectures for level MA to interact with each other, to have a composition relation with each other, or even to inherit each other.

Architecture package : In order to respect software architecture definitions including components are parts of architectures, MADL component is a part of MADL Architecture (i.e. Component is part of architecture). The main principle of MADL, which says everything is a component is applied to Architecture, so Architecture inherits Component. Consequently, architectures for level MA behave like components, i.e. can interact with each other, have a composition relation with each other and inherit each other. Architecture is an instance of Meta-architecture.

Component. It is an abstract class that classifies and defines all MADL elements and entities. As a result, all elements of MADL inherit Component, either directly or indirectly (the principle of everything is a component). Components and connectors of level MA can be generalizable and specializable elements, they can also be composed of other elements, this justifies the inheritance relation and the composition relation between component and itself (to allow elements of level MA to engage in an inheritance relation and composition relation). Component is a part of architecture, therefore, each Component and connector of level MA must be part of an architecture. Component is an instance of Meta-component.

We can also count four types of relation among elements in MADL, instance of, association, composed of and inheritance:

- Instance of, when an element conforms to the definition of another element. It defines the association between an element (or an architecture) and its meta. For example, component is an instance of Meta-component, Architecture is an instance of Meta-architecture. To respect the reflexive principle, on which MADL is based, architecture is an instance of meta-architecture, component is an instance of meta-component and meta-component is an instance of itself, so all the instantiation relations ends in meta-component. The principle is also applied to meta-meta-architecture which is an instance of itself.
- Association, when an element has an association with another element. For example, the association between Meta-component and Meta-connector, the multiplicity is set to 1..*, so components of level MA can engage in more than one association at the same time.
- Composed of, when an element is composed of another element. For example, Meta-architecture is composed of one or more Meta-component, one or more Meta-connector and zero or one Meta-interface.
- Inheritance, when an element inherits another element. For example, Meta-component, Meta-connector and Meta-interface inherit component.

All these relations are nothing but instances of Meta-Connector, i.e. are implemented using specific types of connectors, in order for the model to utilize merely architectural elements. For example, inheritance is achieved by a specific connector that implements the inheritance relation between two elements, the roles of this connector (the interfaces of the connector) connect the two parties (the super element and the sub-element) and the glue, which defines the behavior of the connector, insures that the sub-element is identical (from the same type) to the super element.

Defining MAs using MADL : To define a new meta-architecture (new ADL) we instantiate MADL and a new model conforms to the definition of MADL is obtained. Each meta-architecture element is an instance of a MADL element, elements and notations related to computation are instances of Meta-component, elements and notations related to interaction and communication are instances of either Meta-connector or Meta-component depending on their role and definition (are they intended to be explicitly or implicitly defined). For example, components, configurations and properties are instances of Meta-component, meanwhile, constraints, bindings and attachments are instances of Meta-connector. In this section we give two examples of instantiating ADLs from MADL, COSA[7] and Acme[8].

COSA: COSA is a meta architecture that respects the definitions and the regulations imposed by MADL. It is a component-object based modeling notations based on separating components from their interactions. The architectural model of a system provides a high level model of the system in terms of components that do the

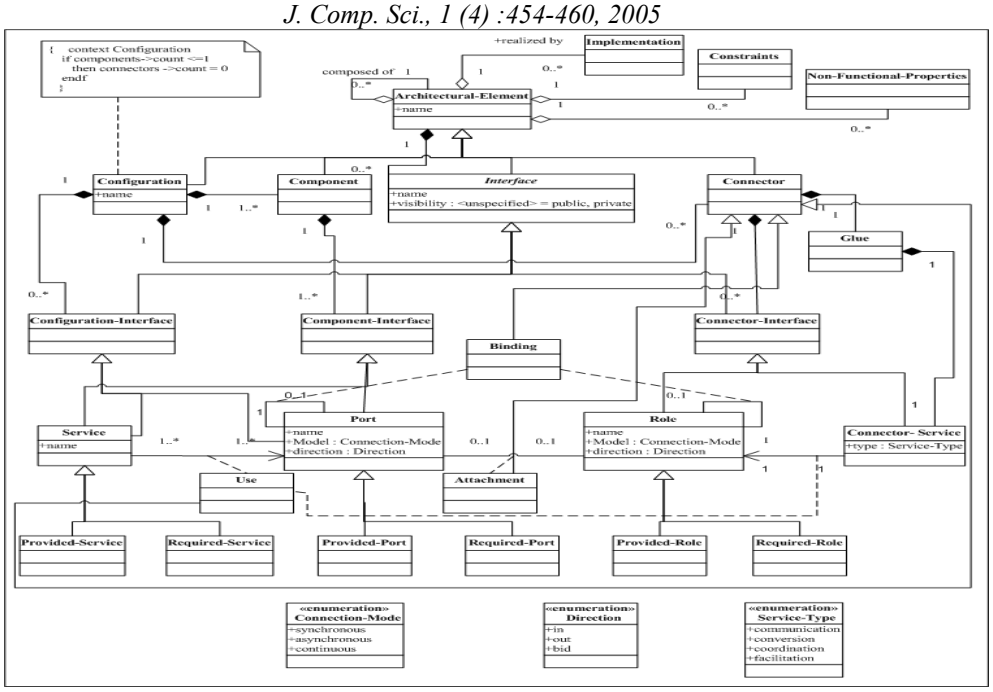

Fig. 3 : The COSA model

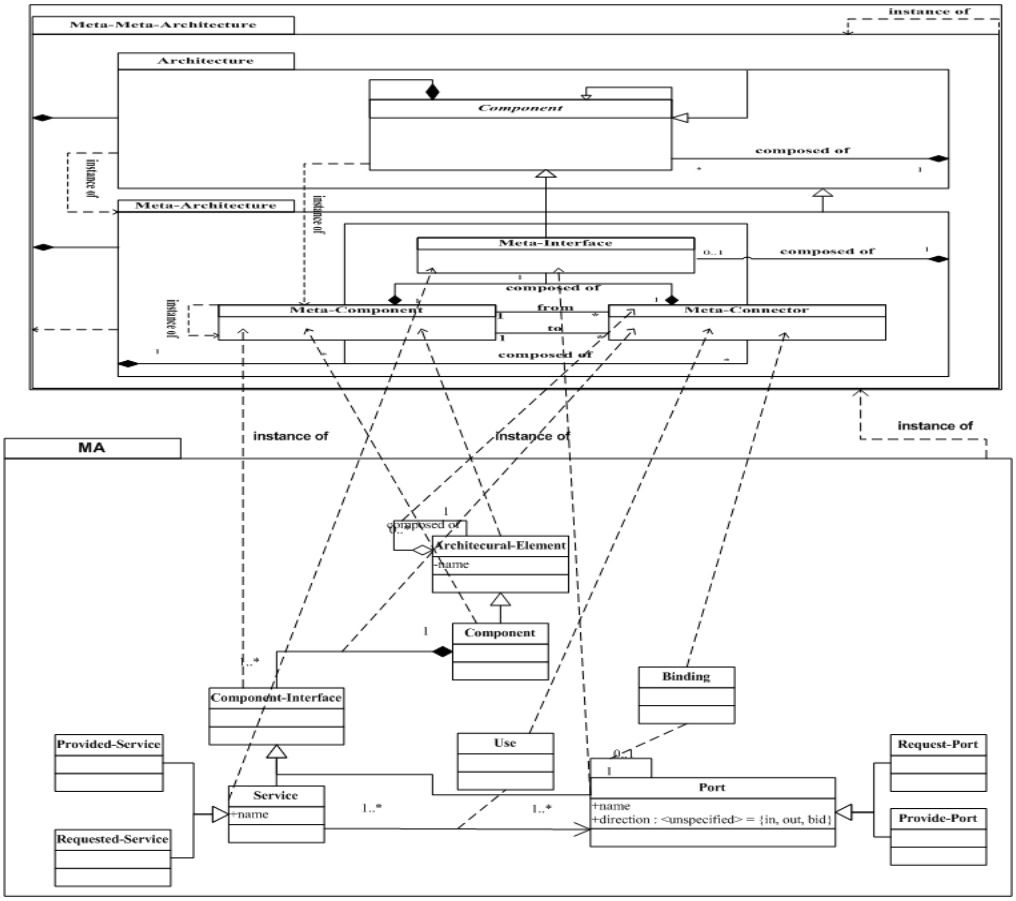

Fig. 4 : Instantiating the component part of COSA from MADL

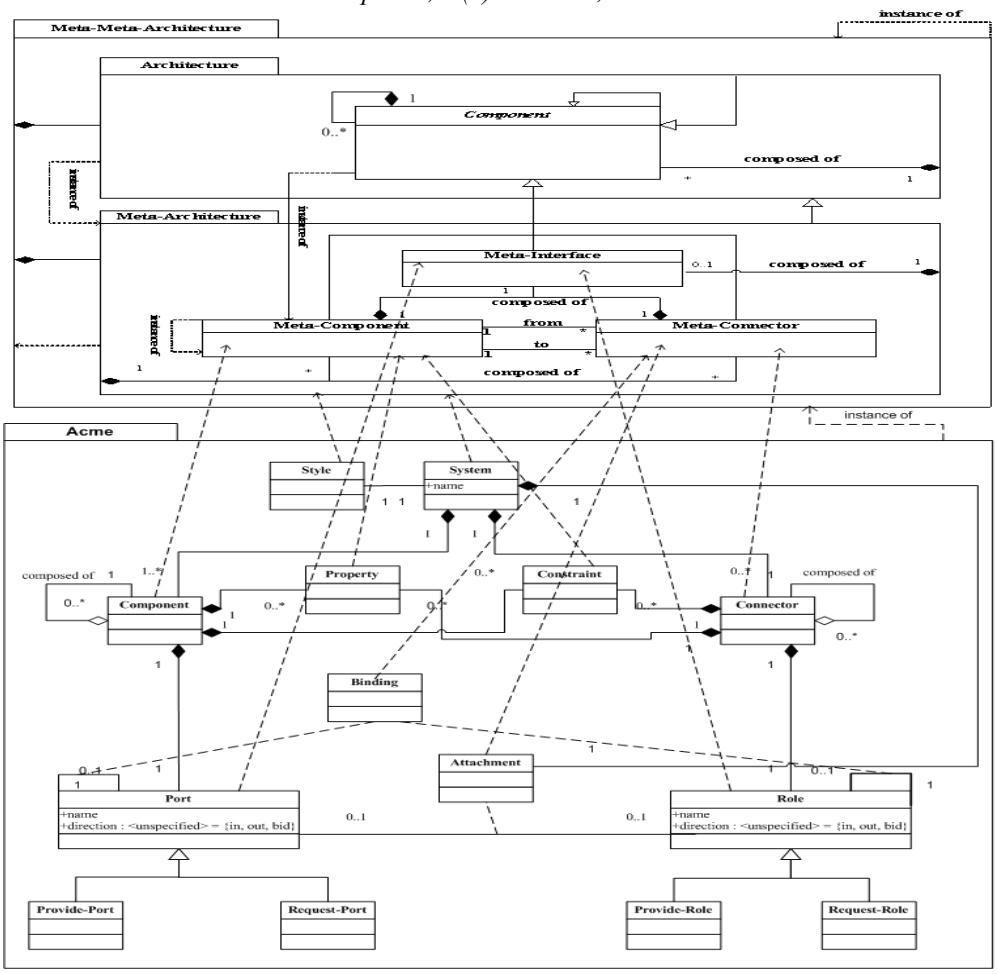

Fig. 5: Instantiating Acme from MADL

computation and connectors that causally connect the components. Figure 3 presents a meta-model of COSA, the key entities of the COSA are:

- Components. Encapsulate computation.

Connectors. Encapsulate interactions and communications among the components.

- Configuration. Define the topological structure of the architecture.
- Ports. Are components interfaces.
- Roles. Are connector interfaces.
- Services. Present the functionality of the entities (components, connectors).

The key associations are:

- Attachments. Link a port to a role.
- Bindings. Link two ports or tow roles together.
- Uses. Link a services (or services) to port/role (or ports/roles).

Instantiating COSA from MADL : Figure 4 shows how COSA is instantiated from MADL. The figure shows only the part related to components. As can be concluded from the figure, each MA notation is an instance of MADL notations. For example components and configurations are instances of Meta-component, connector, binding, attachment, are instances of Meta-connector, while interfaces such as ports and roles are instances of the Meta-interface. The meta-architecture itself is an instance of the met-meta-architecture (M^2A).

Acme : Other models can be easily instantiated from MADL. As an example, we chose Acme, because it considers most of architecture description concepts and notations. Acme has resulted from a careful consideration of issues in and notations for modeling architectures. As such, it can be viewed as the starting point for studying existing ADLs and developing new ones. However, Acme represents the least common denominator of existing ADLs rather than a definition

of an ADL. Architectural structure is defined in Acme using seven types of entities: components, connectors, systems, ports, roles, representations and rep-maps.

Instantiating Acme from MADL : As it can be concluded from Fig. 5, all notations of the model Acme are just instances of MADL notations. For instance, components are instances of Meta-component, while connectors, bindings and attachments are instances of Meta-component. Moreover, systems and styles (styles are just instances of systems) are instances of Meta-architecture.

CONCLUSIONS

In this article we have shown how techniques of meta-modeling can be applied in software architecture. We have also shown the need to propose mechanisms of reflexivity within the domain of software architecture meta-modeling. Next we have presented a meta-meta-architecture dedicated to software architecture, then we have shown how to use it to instantiate meta-architectures. Our approach shows that meta-meta-architecting can be a good tool of documenting, analyzing, comparing and unifying ADLs. Present proposal is directed to the context of the recent research work, in which software architecture is oriented more and more toward the OMG group works[6]. Thus, the introduction of software architecture concepts in UML 2.0[9] is considered as a sign of this orientation and it gives an indication for the possibility to define (component based) software architectures in a way that refinement to adapted executive platforms is much easer.

REFERENCES

1. Oussalah, M. Component-oriented KBS. Proceedings of 14th Intl. Conf. Software Eng. and Knowledge Eng. (SEKE'02) Ischia, Italy, pp: 73-76.
2. Dewayne, E. P. and L. W. Alexander, 1992. Foundations for the study of software architecture. ACM SIGSOFT Software Eng., Notes, 17: 40-52.
3. Medvidovic, N. and R. N. Taylor, 2000. A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Transactions on Software Eng., 26: 70-39.
4. Allen, R.J., R. Douence and D. Garlan, 1998. Specifying and analyzing dynamic software architectures. Proc. 1998 Conf. Fundamental Approaches to Software Eng., Lisbon, Portugal.
5. Ivers, J., P. Clements, D. Garlan, R. Nord, B. Schmerl and J.R. Silva, 2004. Documenting Component and Connector Views with UML 2.0, Technical Report CMU/SEI2004-TR-008.
6. OMG-MOF, 2004. Meta Object Facility (MOF) Specification, Version 1.4, February 2004, www.omg.org/docs/pas/04-02-01.pdf
7. Smeda, A., M. Oussalah and T. Khammaci, 2004. A Multi-paradigm approach to describe complex software system. WSEAS Trans. on Computers, 4: 936-941.
8. Garlan D., R. Monroe and D. Wile., 2000. Acme: Architectural Description of Component-Based Systems. In Foundations of Component-Based Systems (L. Gary and S. Murali), Cambridge University Press: 47-68.
9. Object Management Group. UML 2.0 Superstructures Specification: Final Adopted Specification. http://www.omg.org/docs/ptc/03-08-02.pdf, August 2003.