

Multi-view Meta-modeling of Software Architecture Behavior

Ammar Bessam

LAMEL laboratory, Department of Computer Science, University of Jijel, Algeria
bessamamar@yahoo.fr

Mohamed Tahar Kimour

LRI laboratory, Department of Computer Science, University of Annaba, Algeria
mtkimour@hotmail.fr

Abstract—Component based development is recognized now as a powerful tool to manage actual systems' technological complexity. The success key factor of this discipline is the high level abstracting of systems' structural and behavioral constituents. On the other hand, enhancing software architectures simplicity and clarity by separating several concerns is a useful technique to manage complexity. In order to have a complete system specification, a rigorous behavior description is needed. Behavioral concepts and their use in architectural specification are in a fast evolution and have become so numerous, so it becomes difficult to elicit and manage them. For these purposes, we present in this paper, a generalized meta-model of behavioral aspects, that indexes the various architectural behavior concepts in classes, in a generic way. To enable more sophisticated and consistent analysis of architecture behavior we have separated behavioral concepts into packages basing on four functional perspectives: interface, static behavior, dynamic behavior, and interaction protocols. We show that our proposed meta-model allows having a general, a unified and an adaptable view of behavioral concepts required in software architecture description from all functional viewpoints.

Index terms—Multi-view behavior meta-modeling, architecture description languages, Software architecture functional views, component-based software architecture, extensibility.

I. INTRODUCTION

Complex software systems require expressive notations for representing their software architecture. In this field, Component Based Software Engineering has now emerged as a discipline for systems development. Since 1990 software engineering community has developed techniques centered on systems' architectures description [1]. These have been elaborated in order to improve the complex systems comprehension and design, favorite their evolution and their reuse, proceed to diverse analysis, make easy construction and deployment, or also assist the configuration management. These techniques are concretized by specific languages, the ADLs (Architecture Description Language), and associated tools [2]. Today, ADLs are the principal research area in software architecture, especially amelioration of architectural representation in terms of flexibility and extensibility.

Many ADLs have been developed mainly focusing either on behavioral aspects (ex. Rapide [3], Darwin [4]) or on system's structural aspects (ex. Aesop [5], COSA (component and object-based Software Architecture) [6]). Many formal and informal techniques are used to allow behavioral specification in software architectures, such as, state chart, CSP, Pi calculus, etc. Behavioral concepts and their use in architectural specification are in a fast evolution. They, also, become so numerous, so it becomes more difficult to represent and to handle or manage them. This diversity of behavioral concepts is generated by the large spread of software architecture in several fields. Specifying behavior of system architecture is necessary to have a complete software architecture description. And necessity of meta-modeling to supervise behavioral concepts in all domain specific languages has increased.

In [6], the authors have developed a meta-model for COSA, only describing structural entities, without offering any specific mechanism to handle the behavioral concepts. After that, authors of [7] have developed a meta-model to specify behavior in COSA architecture, by adding behavioral concepts in the previous meta-model. An extension of this meta-model is done to support behavior specification of software architecture in meta-modeling level [8]. From a first look to this meta-model we can observe, also, a difficulty of its understanding and eventual ambiguities and redundancies in its use. One way to make it clearer and more expressive is dividing it into distinct packages. To do this decomposition we are based on a multi-view description of the architecture behavior.

Describing software architecture from multiple perspectives provides more simplicity in using and understanding such description. It allows also, more advantages for analysis of its functionality. Integrating multiple views representation for software architecture provides high level extensibility of complementary views of a software system. Functional characteristics of a component in various component based languages are usually described from one or more of four aspects [9]: interface's behavior, static behavior, dynamic behavior, and interaction behavior. Interface behavior captures, in particular, how a component behaves with other architectural entities in the interface level. Static behavior

is the discrete functionality of architectural elements. Medvidovic N. and Roshandel R. have described static behavior in [10] by a set of properties: {a set of state variables, an invariant, and a set of operations}. Dynamic behavior is the continuous state changes of architectural elements during their execution. In [10] dynamic behavior is defined by a set of properties {an initial state, a set of states, and a set of transitions}. Interaction or connection behavior is the specialization of interaction protocols through an external view of the architectural entity (interaction's ordering, interaction's dependencies, etc.).

In order to efficiently manage the large explosion of behavioral concepts and relations among them, we present in this paper, a set of common and generic concepts allowing behavioral specification, in a multi-view meta-modeling level. Our proposed meta-model offers to architects a complete and well organized and categorized definition of behavioral concepts which can be used to enhance architectural entities and models by specifying their behavior in a generic way.

In doing so, we targeted important aspects of software, by the identification of the general concepts and their relations. We should have a distinction between what is modeled and what is the framework where the model and its entities live, to make it possible to apply meta-modeling to real software development. This will cope with the context of Model Driven Engineering, where meta-modeling is presented as «a convenient way for isolating concerns of a system [11]. Meta-model specifies the set of concerns that should be taken into account while creating a model.

The remainder of this paper is organized as follow: in section 2, we provide an overview on behavior specification in software architectures description. In this section we present main techniques used by academic and industrial communalities to specify behavior of architectural elements. Section 3 is reserved to describe the functional views that we have agreed in different packages' definition. In section 4 we have describe the behavior specifications in some existing ADLs focusing on the four viewpoints. In section 5, we present the proposed meta-model and description of its basic entities. The paper is concluded with a discussion of our approach and some remarks on our objectives for future work in Section 6.

II. BEHAVIOR MODELING IN SOFTWARE ARCHITECTURES

In this section we focus on the value of specifying behavioral aspects of architecture elements. Also, we explore behavioral aspect in important existing ADLs.

A. Specifying Behavior in ADLs

Specifying behavior is a way to add semantic detail to structural elements and their interactions that have time related characteristics. Behavior specification, in software architecture, prepares rules for executing the actions and provides the basis for executing the architecture. To ensure that the specified architecture is well-formed, both static and dynamic behavior needs to be verified.

Architects describe behavior to specify how an element behaves when stimulated in a particular way, or to specify how a set of elements react with each other. Specification of architecture elements behavior is used for system analysis, for constraints enforcing, and for consistent matching of architectures from one level of abstraction to another. It is indispensable to reason about and to explore, in architectural level, the completeness, correctness, and quality attributes of the final product resulting of architecture. In most ADLs, behavioral aspect specification is reduced to components behavior.

Component behavior description is supported by all existing ADLs, although to varying degrees from expressing behavioral information in component property lists of UniCon to model of dynamic behavior in Wright and Rapide. Along this spectrum we find other representations. Connectors' behavior is defined especially in ADLs which -model explicitly connectors as first class entities. While we find some ADLs using different behavioral models for their connectors than for components, most of them tend to use a unique language to model both components and connectors behavior.

Different concerns are described in different viewpoints of software system architecture, including those of the syntactic dependency among components through interfaces, static behavior, dynamic behavior and interactions of components. Behavior concern is specified in [9] from four different viewpoints on what we will base to represent behavior specification of architectural elements in meta-modeling level.

1. Interface view specifies, statically, the interaction points of a component.
2. Static behavior view captures the functionality of a component in a discrete manner, during the system's execution. It represents several states of architectural elements during the system execution. Its description is focused generally on pre- and post conditions specification.
3. Dynamic behavior view provides, in contrary of static one, a continuous view of how an architectural entity arrives at different states presented in static view throughout its execution. Its description is based, usually, on state machines.
4. Interaction behavior view provides the external view of the architectural entities and how they interact with each others in the system.

These four views will be more detailed in the next section.

B. Used techniques in behavior modeling

In this section we focus on techniques frequently used for, specifying behavioral aspects of architecture elements. In software architecture, to specify architectural elements behavior, many techniques have evolved over time, scaling from a documentation written in plain English attached to each element to some sophisticated formal methods [12]. One of the oldest forms of component behavior specification is based on enhancing component interfaces with pre- and post-conditions. A more convenient approach to specifying component behavior is employing various process algebras. These

formal methods became domains of interest for a majority of component models originating in the academic area.

Among others, the modified Hoare's CSP notation [13] used in Wright, Pi-calculus employed in Darwin [4], and also the behavioral protocols proposed by SOFA [14] are named in particular. On the other hand, in the area of the software industry, the Unified Modeling Language (UML) is becoming increasingly popular as a widely accepted standard for specifying software architectures.

Formal methods are mathematically-based techniques for specification, development and verification of software systems. There are a few formal methods [13], which can be suitable for specifying a component based systems' desired behavioral and structural properties. The suitable are formal methods such as temporal logics, automata-based methods and process calculi.

III. FUNCTIONAL VIEWS OF SOFTWARE ARCHITECTURE BEHAVIORAL DESCRIPTION

An efficient architectural description should provide a multi view representation of architectural elements and their relative properties. The complicated aspect in architecture description is architecture dynamicity. So, it is indispensable to give more detailed views of architecture behavior. Authors of [16, 17, 18, 19, 20] have recognized that modeling from multiple perspectives is an effective way to capture several properties of component-based software systems. For example, UML, in its last version, has employed thirteen views to model requirements from several system aspects.

To have a consistent specification of software architecture behavior, we focus on multiple functional modeling aspects of software components. We adapt a four-view modeling technique to allow a high abstraction level description of functional characteristics of software entities from four perspectives.

A. Interface behavior view:

Interface description is taken in a count by several languages in different abstraction levels from programming languages to general purpose modeling notations such as UML. Behavior of software architecture elements is focused, generally, on the level of architectural entities interfaces. Various ADLs COSA+ [8], Rapide [3], Wright [21] describe behavior within interface description. The first package, depicted by figure 1, of the proposed meta-model include all behavior concepts allowing description of the interface behavior. Both, components, connectors and the entire system architecture are handled as first class entities in this level of description. So, when we talk about component interfaces, connector interfaces and the whole architecture interface we use respectively, port, role and architecture interface. The principle behavior unit in the interface behavior level is the *Event* concept.

B. Static Behavior view:

Static behavior view extends interface one with static behavioral semantics [22, 23, 24], to describe the internal behavior of the elements. This extension is supported by several ADLs to represent behavioral characteristics in specific discrete states during the system's execution. Static behavioral specification is used to describe several states of a component during specific points of time, without expressing the manner how the component arrives at a specific state. It focuses on what a component does handle while it is in a given state. The static behavior can be validated by analyzing several static properties such as, connectedness, data-transfer paths and completeness using, especially, graph based models.

C. Dynamic Behavior view:

Dynamic component behavior gives more detailed specification of the component behavior by adding information about the manner how it arrives at certain states during its execution. It is used to express the continuous state change of architectural elements. This view gives more detailed internal information about each architectural entity. It is most commonly known as runtime evolution. Most dynamic behavior concepts can be inspired, generally, from UML stateChart metamodel.

D. Interaction behavior view:

This view is focused on representing of interactions among architectural entities. It adds information about the manner how architectural entities communicate during their execution in time. Interaction behavior view presents detailed external information about each architectural element. It is used to present continuous states changes of an architectural entity according to its information interchange with its environment and basing on some interacting rules.

IV. BEHAVIOR MODELING IN SOME EXISTING ADLs FROM FOUR VIEWPOINTS

Software architecture in SOFA [14] is described as a hierarchy of nested components. Behavior of SOFA components are captured formally via behavior protocols. Runtime structure of a component is composed of a control part, and a functional part, which in the case of a primitive component consists of code of the component. Behavior description in SOFA is based on interface behavior by presenting different frames of each component. Static behavior, in SOFA, is represented by different implementations or architectures of each frame during executing time. Interaction view is in the kernel of SOFA behavior description basing on formal interaction protocols.

In a *Wright*, [21, 25, 26] specification, the behavior of the system entities is described as processes and the interactions between entities thereby become the interactions of communicating processes. Wright uses a subset of CSP [13] to provide a formal basis for specifying the behavior of components and connectors. Wright is concerned with the static structure of software

components as well as their interactions. Dynamic behavior description is limited in Wright because of the restrictions of the CSP process algebra. So, Wright focuses on the description of the static interface and internal view of architectural behavior. Wright supports also the formal specification and analysis of interactions between architectural components.

C2 [27] does not introduce any formal method to describe the communication behavior of the system. Component behavior is expressed in terms of causal relationships between input and output messages in its interface. The basic view in behavior description using C2 is the interface one, which is done using an event-based style; Interaction view is reduced to a set of asynchronous message passing.

Aesop [28, 29] does not provide any language mechanisms for specifying elements behavior. However, for each architectural style defined in Aesop, it allows the use of style-specific languages for modeling behavior.

Darwin [4] and **LEDA** have a precise operational semantics defined in π -calculus. The behavioral view employs Finite State Processes (FSP) to specify the behavior of components. Darwin makes a point of structure being dynamic rather than static. Interface behavior is ensured by communication objects. Darwin supports the analysis of distributed message-passing systems;

ACME [30, 31] allows semantic information to be specified in components' property lists. ACME places no restrictions on the specification language; however, from its point of view, properties are not interpreted, so that, strictly speaking, component behavior is outside the scope of the language. ACME is enhanced by support of a communication description based on events.

Rapide [3] allows modeling of component interfaces and their externally visible behavior. In Rapide, each component specification has an associated *behavior*, which is defined via partially ordered sets of events (posets). Rapide introduces a unique mechanism for expressing both a component's behavior and its interaction with other components.

The view supported by **UniCon** [32, 33] language is primarily the structural view. UniCon's main focus is on non-functional properties of components, it allows specification of event traces in property lists to describe component behavior. UniCon uses interaction protocols to represent several interactions among components.

In the **MetaH** [34] ADL behavioral aspects may be specified using *paths* for sequencing behavior, *events* for triggering purposes, and a number of entity attributes for other aspects (e.g. time between process dispatches, computation deadline, etc.).

UML [35, 36] supports several diagram types for a behavior description of software system – collaborations, sequence and statechart diagrams. UML 2.0 has been used largely in the industrial community for software architecting, however it still lacks basic architectural concepts such as “layer” or a faithful notion of “connector”. Also, it can't analyze interactions among views or make strong connections to code, and it too

easily mixes design concepts with implementation directives. Lately, there exist many works, but solutions offered by them are incomplete, especially in recovery and enforcement of runtime views and dynamic architectural rules.

A. Synthesizing information

In this comparative table (table 02) we present at what level can main existing ADLs support the four views in describing behavior of software architectures. To use this table we have four levels of grey to traduce four levels of usefulness of each ADL in describing the corresponding behavioral view.

TABLE I.
SYMBOLS DEFINITION

Symbol	Signification	Description
○	None	Doesn't support the view completely.
◐	Low	Supports insufficiently the view.
◑	Average	The ADL supports sufficiently the view.
●	Fort	The ADL focused on the corresponding view.

TABLE II.

SYNTHESIZING TABLE OF THE RELATION ADL/BEHAVIOR VIEWS

ADLs	Interface's behavior	Static behavior	Dynamic behavior	Interaction behavior
Wright	◑	●	◑	◐
C2	◐	◐	◐	◐
Rapide	◐	◐	●	◐
SOFA	◑	◑	◐	●
Darwin	◐	◑	◐	◑
Aesop	◐	◑	◐	◐
ACME	◐	◐	○	◐
UniCon	◐	◐	◐	◑
MetaH	●	◐	◑	◐
UML	◐	◑	◑	◐

In order to have a more complete and main set of behavioral concepts, we have done, especially, a more detailed study of next existing ADLs and their supporting techniques to specify behavior: Wright, [21, 25, 26], Rapide [3], and C2 [27]. Hereafter, we present, in table 03, the important concepts used by these ADLs for specifying behavior of software architectures elements,

from the four functional viewpoints allowing definition of views presented previously.

TABLE III.
IMPORTANT BEHAVIORAL CONCEPTS USED BY C2, RAPIDE AND WRIGHT FROM FOUR PERSPECTIVES

<i>Interface's behavior</i>	<i>Static behavior</i>	<i>Dynamic behavior</i>	<i>Interaction behavior</i>
Event	Event	Gard	Event
Observed event	Post-condition	Transition	Interaction rule
Emitted event	Pre-condition	State	Interaction
Event alternation	Result	Activity	Synchronous interaction
Observed event	Process	Transition rule	Asynchr-onous interaction
Function call	Alternative activity	Source state	Message passing
Event sequence	Control activity	Targeted state	-
-	Indeterministic choice activity	-	-
-	Parallel activity	-	-
-	Sequence activity	-	-
-	Recursion activity	-	-
-	Deterministic activity	-	-
-	Format conversion activity	-	-

V. MULTI-VIEW BEHAVIORS META-MODELING

Many works and papers have mentioned that we can use both object technology and component technology to practically describe software architectures [35], [12]. In [35] and others works of Medvidovic are focused on using UML to describe software architecture. In our approach we suggest to explicitly present behavioral concepts issues from behavior specification in existing architecture description languages. This allows to completely separating structural from behavioral aspects in software architecture representation. In order to have a more complete set of behavioral concepts required by architects to express architectures behavior, we have done a large study of techniques making basis for behavior definition in existing ADLs, especially, those mainly focusing on behavioral specification, such as Darwin, Rapide, Wright and C2. To specify behavior and coordination of architectural entities, our meta-model offers a rich set of concepts. Each concept can be implemented in an appropriate language. In some cases, it is better to specify all behavioral entities by the same language.

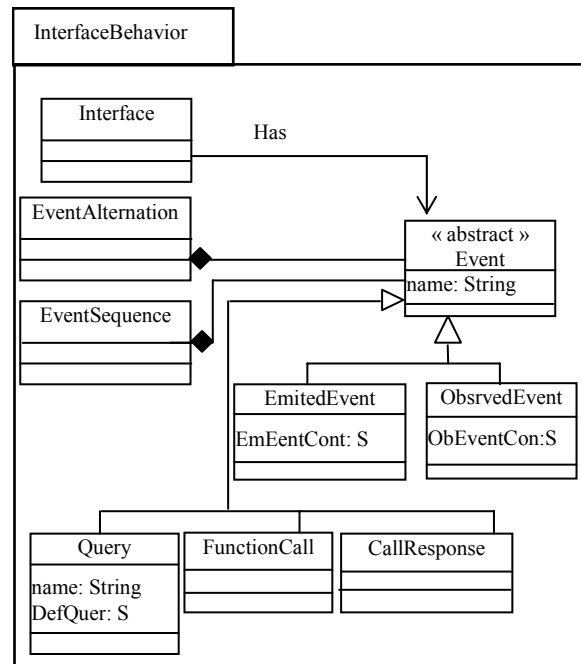


Figure 1. Package 1: Interface behavior meta-model.

In the previous section we have seen that most of existing ADLs represent behavioral specification, only, in component level. In our case, all architectural entities are represented explicitly, so we represent connector behavior and the entire system behavior independently from component behavior.

Defining architecture behavior of complex software systems from multiple perspectives is essential for capturing separated and clarified information about interfaces' behavior, static behavior, dynamic behavior and interaction properties of the system under development. Separating good views of complex software systems in terms of their appropriate components is an important step in realizing the goals of architecture-based software development. Different packages are specified from the following four modeling perspectives: *Interface behavior*, *Static Behavior*, *Dynamic Behavior*, and *Interaction behavior*. In the following text of the section, the terms in italics have direct representation in the meta-model.

A. *Interface behavior meta-model*

Interface behavior package is considered as the core of the whole meta-model packages through dependency relations. So, for each package we can find, in addition to all considered structural concepts, the interface concept. Here, we focus on behavioral characteristics of the interface without taking in a count its structural description like its type, interface elements, and its set of parameters and their types.

The main behavioral concept in interface level is the *event* one. An *event* is a class that abstracts all events used or generated by the architectural elements. All its instances can be one of these types:

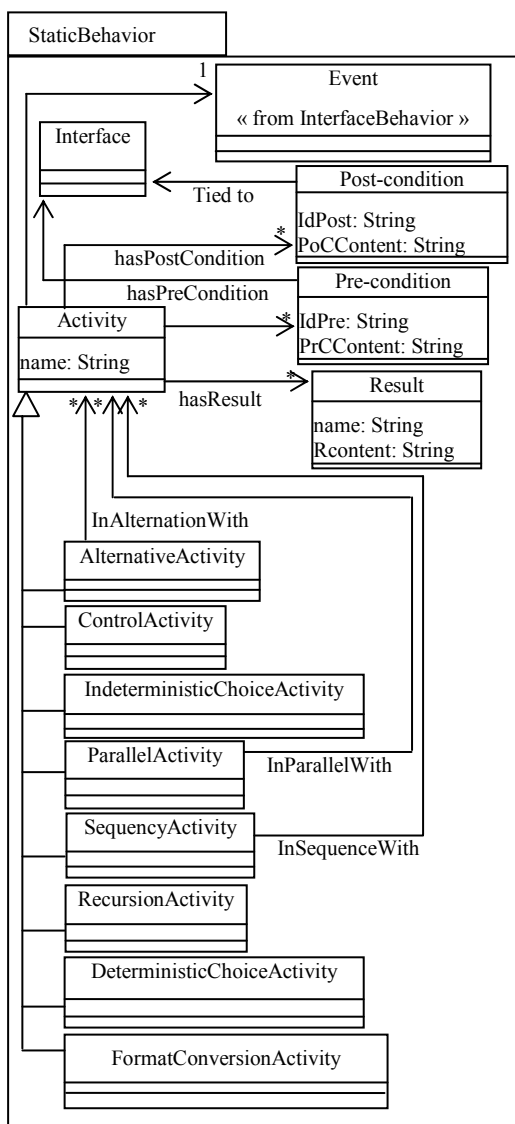


Figure 2. Package 2: Static behavior meta-model

- *Emitted events*: events generated by an architectural element.
- *Observed events*: events received and handled by an architectural element.
- *Query*, *FunctionalCall*, and *CallResponse* classes are used to represent different natures of an event in software architectures.

An interface behavior may be generated by a sequence or an alternation of various events.

B. Static Behavior meta-model

In this level we specify functional properties of an architectural entity in terms of its several *states* and relative operations or *activities*. This package assembles the following set of concepts:

Instances of a *state* represent different states of an architectural element during its execution.

To combine and handle events, we find the *Activity* class. An *activity* is the abstraction of all processes

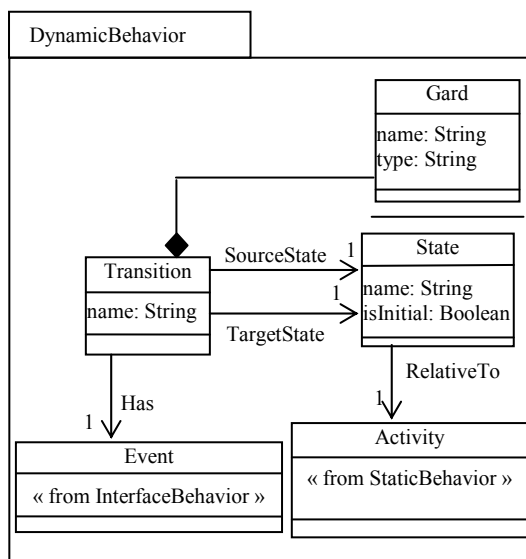


Figure 3. Package 3: Dynamic behavior meta-model

performed by an architectural entity in the context of software architecture.

FormatConversionActivity is an important function of Glue (concept used in several ADLs to represent the connection type and the functionality of a connector) in case of heterogeneous interactions.

An *Activity* can be elementary or composed by a combination of other activities in forms of *parallelism*, *sequence*, and *alternation*. The proposed meta-model distinguishes between simple activities like *Format conversion activity*, *control activity*, *recursion activity* and complex activities like *Parallel activity*, *Sequence activity*, *alternative activity* and *Deterministic and indeterministic choice activity*.

An *activity*, to be executed, may have a *precondition* and a *postcondition*. Execution of an *activity* requires and generates several types of *events*, and provides a set of results.

C. Dynamic Behavior meta-model

The dynamic behavior meta-model will provide a set of concepts and relations among them to describe, as we have seen previously, the continuous state changes of an architectural entity. To express completely this view, we have offered the possibility to describe all details about architectural entities state changes through corresponding transitions. This package offers all internal execution details representation. For this purpose, we define next concepts and relationships:

The main information depicted from this view is a set of states and a sequence of guarded transitions from a source to a targeted state.

Each *transition* has a source *state* and an arrival or a target *state*. A *transition* can be composed of a *guard* and, usually, an *event*. In a *state*, a structural element can perform a set of *activities*. So, the activity class, in this view, is used to describe the internal operations of the architectural entity.

The architectural entity *state* and *transitions* between them are the most important concepts in this package, because they are in basis of the dynamic behavior specification in any systems' description models.

A state, in the proposed meta-model is specified by their name and set of incoming and outgoing transitions. It can be an initial state, a final state or a regular state.

D. Interaction behavior meta-model

This portion of the entire meta-model of behavioral concepts required to specify the behavior of software architecture is denied to specify the valid sequences of operations' invocations among architectural entities.

Specification of invocations sequences is done independently from internal state and operation's pre-conditions, because interaction behavior is reserved for external view of the architectural element. The set of *interaction rules* forms protocols of interactions. And one *interaction* is based on a set of *events*.

An *interaction* is an entity allowing definition, through its instantiation, of different interactions among others structural entities of the system. To make the metamodel more flexible, an interaction consists usually of several messages, some of which can be synchronous, some others can be asynchronous. The set of *interaction rules* forms protocols of interactions.

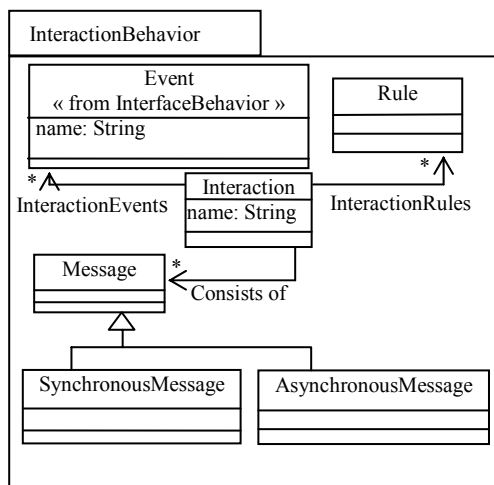


Figure 4. Package 4: Interaction behavior meta-model

E. Structural model of views dependencies

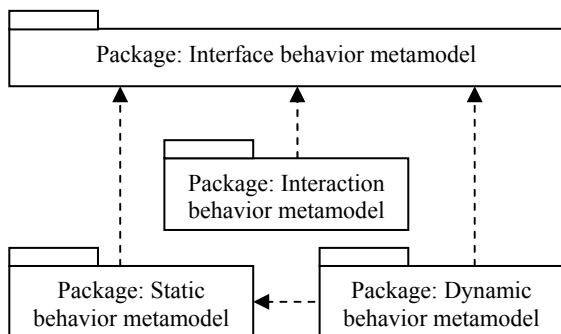


Figure 5. Structural model of view's dependencies.

The interface is in the core of all behavior views. Because static, dynamic and interaction behaviors are expressed in relation with architectural entities interface. At static behavior meta-modeling level, the pre- and post-conditions are linked to the specific interface used for accessing to the corresponding operation.

VI. CONCLUSION AND FURTHER WORKS

The separation of concerns using various views facilitates considerably the task of architect. It makes the architecture easier; improve the testability and the software maintainability. The multi view representation is used to modularize in separated categories some specific concerns like Behavioral description, structural description, deployment description, etc. In this paper we have defined a four-view meta-modeling approach to describe and manipulate behavioral concepts in software architecture as a separate dimension of component-based software design in order to improve the modularity when the architect defines his architecture.

This multi view description of software architecture behavior allows increasing the efficiency and completeness of behavior specification in architecture description languages. It can be used to comprehensively specify interface properties, static and dynamic behavior, and interaction properties of a software architectural entity. The high level definition of behavioral concepts in the meta-model makes it an extensible and opened model on different transformations into other models and formalisms. Moreover, we allow integrating of different architectural relative aspects. At this stage, the meta-model is programming language independent model, as it doesn't refer to any specific language to define architecture behavior. It offers to architects a set of common and generic architectural elements to specify behavior of software system architecture. Architectural elements behavior is described through meta-model entities instantiation. Representing elements of architecture as first class entities permits an independent and explicit behavior definition of each architectural element. Moreover, we believe that the software description in meta-model level makes generic, so reduce the semantic gap which exists between different application domains by abstracting various specificities.

The ideas presented in this paper is a continuation of our previous work, in which we have integrated a whole meta-model to describe behavior in COSA architecture. Currently, we are studying different possibilities to adapt and enrich our meta-model in at least three directions: i) analyzing appropriate languages and techniques to formally specify behavioral entities of our meta-model, and, ii) extending the MOF meta-model corresponding packages to cope with the behavioral aspects in the ADLs. Our long-term goal is to use this meta-model to support behavior definition in several specific domains, such as real time systems, embedded systems, ubiquitous systems, etc. This will be achieved by offering a set of extension basis, iii) defining, through instantiation mechanism, different adaptation of the

proposed meta-model to existing ADLs those support behavior specification.

REFERENCES

- [1] IEEE Architecture Working Group: "Recommended Practice for Architectural description of Software-Intensive Systems", *IEEE Std 1471- 2000*, IEEE, 2000.
- [2] N. Medvidovic and R. N. Taylor: "A Classification and Comparison Framework for Software Architecture Description Languages", *IEEE Transactions on Software Engineering*, vol. 26, no. 1 (January 2000).
- [3] D.C. Luckham, L.M. Augustin, J.J. Kenny, J. Veera, D. Bryan, W. Mann. "Specification and analysis of system architecture using Rapide", *IEEE Tr on Software Engineering* vol. 21 no 4, April 1995, pp 336-355.
- [4] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. "Specifying distributed software architectures". *Proc 5th European Software Engineering Conference*, September 1994.
- [5] D. Garlan, "An Introduction to the Aesop System", D. Garlan, <http://www.cs.cmu.edu/afs/cs/project/able/www/aesop/html/aesopoverview.ps>, juillet 1995.
- [6] A. Smeda, and T. Khammaci, M. Oussalah, "Multi-Paradigm Approach to Describe Software Systems", In *Proceedings of 3rd WSEAS Int. Conf. On Software Engineering, Parallel and Distributed Systems*, Salzburg, Austria, 2004.
- [7] A. Bessam and M. T. Kimour. Integrating Behavioral Aspect into COSA Architecture. *Proceedings of SEDE-2007 16th Int'l Conf on Software Engineering and Data Engineering*, 2007.
- [8] A. Bessam, M. T. Kimour and A. Melit. Behavioral Metamodeling for Software Architecture Description. To be published in November 2007 issue of IRE.CO.S international review, 2007
- [9] R. Roshandel, N. Medvidovic, "Modeling Multiple Aspects of Software Components", in *Proceeding of Workshop on Specification and Verification of Component-Based Systems, ESEC-FSE03*, Helsinki, Finland, September 2003.
- [10] Roshandel R., Medvidovic N., Multi-View Software Component Modeling for Dependability, in *Architecting Dependable Systems II*, LNCS, 2004.
- [11] Didonet Del Fabro M., Bézevin J., Jouault F., Breton E., Gueltas G.: AMW: a generic model weaver. In: *Proceedings of the 1st day on Model-Based Engineering (MDE)*, 2005.
- [12] D. Garlan, S. W. Cheng, A. Kompanek, "Reconciling the Needs of Architectural Description with Object-Modeling Notations", *Science of Computer Programming* 44 (2002) 32-49.
- [13] C.A.R. Hoare, "Communicating Sequential Processes.", *Prantice Hall, Englewood Cliffs NJ*, 1985.
- [14] F. Plasil, D. Balek, R. Janecek, "SOFA/DCUP: Architecture for Component Trading and Dynamic Updating", *Proceedings of ICCDS 1998*, USA, 1998.
- [15] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, Upper Saddle River, NJ, USA, 1998.
- [16] Booch G., Jacobson I., Rumbaugh J. "*The Unified Modeling Language User Guide*", Addison-Wesley, Reading, MA.
- [17] Dias M., Vieira M., "Software Architecture Analysis based on Statechart Semantics", in *Proceedings of the 10th International Workshop on Software Specification and Design, FSE-8*, San Diego, USA, November 2000.
- [18] Hofmeister C., Nord R.L., and Soni D., "Describing Software Architecture with UML" In *Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, San Antonio, TX, February 22-24, 1999.
- [19] Krutchen, P.B. "The 4+1 View Model of Architecture", *IEEE Software* 12, pp. 42 - 50, 1995.
- [20] Nuseibeh B., Kramer J., and Finkelstein A., "Expressing the Relationships Between Multiple Views in Requirements Specification", in *Proceedings of the 15th International Conference on Software Engineering (ICSE-15)*, Baltimore, Maryland, USA, 1993.
- [21] R. Allen, and D. Garlan, "A Formal Basis for Architectural Connection", *ACM Transitions on Software Engineering and Methodology* vol.6, No. 3, p. 213-249, 1997.
- [22] Aguirre N., Maibaum T.S.E., "A Temporal Logic Approach to Component Based System Specification and Reasoning", in *Proceedings of the 5th ICSE Workshop on Component-Based Software Engineering*, Orlando, FL, 2002.
- [23] Liskov B. H., Wing J. M., "A Behavioral Notion of Subtyping", *ACM Transactions on Programming Languages and Systems*, November 1994.
- [24] Zaremski A.M., Wing J.M., "Specification Matching of Software Components", *ACM Transactions on Software Engineering and Methodology*, 6(4):333-369, 1997.
- [25] R. Allen, and D. Garlan, "The Wright architectural specification language", *Technical Report of CMUCS-96-TBD*, CMU, School of Computer Science, September 1996.
- [26] M. Graiet, M.T. Bhiri, F. Dammak, and J. Giraudin, "Adaptation d'UML 2,0 à l'ADL Wright", *LSR-IMAG Laboratory*, 2001.
- [27] R.N. Taylor, and N. Medvidovic, "A Component and Message-based Architectural Style for GHI Software.", *IEEE Transactions on Software Engineering*, vol. 22, No. 6, June 1996.
- [28] D. Garlan, R. Allen, and J. Ockerbloom, "Exploiting Style in Architectural Design Environment", *Proceeding of SIGSOFT'94 Symposium on the Foundations of Software Engineering*, December 1994.
- [29] D. Garlan, R. Monroe, and D. Wile: "Acme: An Architecture Description Interchange Language" In *Proceedings of CASCON 97, Toronto, Ontario, November 1997*, pp. 169-183.
- [30] D.Garlan, R.T.Monroe and D.Wile: "Acme: Architectural Description of Component-Based Systems", *G.T.Leavens and M.Sitaraman, Eds: Cambridge University Press*, 2000.
- [31] M. Shaw, "Abstractions for Software Architecture and Tools to Support Them". *IEEE Transactions on Software Engineering*, 1995. 21(4): pp. 314-335.
- [32] G. Zelesnik: "The UniCon language Reference Manual", Carnegie Mellon University, Pittsburgh, Pennsylvania, 1996.
- [33] P. Binns, M. Englehart, M. Jackson, and S. Vestal, 1995, "Domain-Specific Software Architectures for Guidance, Navigation and Control," to appear in *International Journal of Software Engineering and Knowledge Engineering Honeywell Technology Center, Minneapolis MN*, January 1994, revised February 1995.
- [34] N. Medvidovic, D.S. Rosenblum, D.F. Redmiles, J.E. Robbins, "Modeling Software Architectures in the Unified Modeling Language", *ACM Transactions on Software Engineering and Methodology*, Vol. 11, N0 .1, January 2002.

- [35] M. Kande, A. Strohmeier: "Towards an UML Profile for Software Architecture Descriptions". UML'2000 - The Unified Modeling Language: Advancing the Standard, *Third International Conference*, York, UK, October 2-6, 2000, Kent, S., Evans, A., Selic, B.(Ed.), LNCS (Lecture Notes in Computer Science).

Ammar Bessam is born in Algeria on Mai 28 1975. He is a teacher in the Computer Science Department in University of Jijel, Algeria. He has obtained his Magister Diploma in 2003 at University of Jijel. Now, he is inscribed to obtain his doctorate thesis in software architectures and data and knowledge engineering. Ammar's research interest is software architecture modeling and alignment with UML, and MOF metamodels and MDE. In this subject he has published a paper in proceeding of SEDE'2007 international conference. Mr. Bessam is a membership in Decision Support Systems Modeling team of

Modeling in Electro-technique Laboratory (LAMEL). He teaches courses in information systems, decision support systems, organizational systems and design methods.

Mohamed Tahar Kimour is born in Algeria on January 03 1960. He is an associated professor in Computer Science Department at University of Annaba, Algeria. He has obtained his Doctorate Diploma in 2005 at University of Annaba. Mohamed Tahar's research interest is software model driven engineering and real time and embedded systems modeling. He has published many papers on real time and embedded systems modeling, and lately a paper on software architecture description in proceeding of SEDE'2007 international conference. Dr. Kimour is a head of project research on embedded systems engineering in Research Laboratory of Computer Science (LRI). He teaches courses in information systems, software engineering and organizational systems. His email address is mtkimour@hotmail.fr.