International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:2, 2008

Multi-view Description of Real-Time Systems' Architecture

A. Bessam, and M. T. Kimour

***Abstract*—**Real-time embedded systems should benefit from component-based software engineering to handle complexity and deal with dependability. In these systems, applications should not only be logically correct but also behave within time windows. However, in the current component based software engineering approaches, a few of component models handles time properties in a manner that allows efficient analysis and checking at the architectural level. In this paper, we present a meta-model for component-based software description that integrates timing issues. To achieve a complete functional model of software components, our meta-model focuses on four functional aspects: interface, static behavior, dynamic behavior, and interaction protocol. With each aspect we have explicitly associated a time model. Such a time model can be used to check a component's design against certain properties and to compute the timing properties of component assemblies.

***Keywords*—**Real-time systems, Software architecture, software component, dependability, time properties, ADL, meta-modeling.

I. Introduction

THE rapidly increasing complexity of real-time embedded systems is not the only factor challenging the development. Also, the tasks they handle are manifold, ranging from classical control devices to high-end multimedia applications, automation, and biomedical engineering. With increasing complexity one can observe a shift from electronic and pure control based systems towards software-based systems [1].

In these systems, software has become omnipresent, critical, and complex. Time-to-market of services, which rely on system engineering (operating systems, distributed systems, middleware), is becoming a strategic factor in a competitive market in which operation (deployment, administration) costs are much higher than development costs.

In this context, component-based software architectures have naturally emerged as a central focus in real-time embedded systems. Component-Based Software Engineering (CBSE) is generally recognized as one of the best way to develop, deploy and administrate increasingly complex software with good properties in terms of flexibility, reliability, scalability, lower development cost and faster time-to-market through software reuse and programmers productivity improvements.

A CBSE uses architecture description languages to represent software architecture and its elements, in particular, components, connectors, interfaces, and configuration. An ADL models component structure, their communication patterns, and behavior. It is also used to analyze properties of the system early in CBSD process. Many different architecture description languages (ADLs) have evolved over the years. Some of them targets specific domains or aspects of architecture, others are more special purpose languages.

Architectures are naturally colored by the domain or system family within which they are used, which often gives rise to specific requirements to architecture descriptions. Timing properties are an important aspect of real-time embedded systems. Modeling of time should be a central concern in model-driven engineering for these systems.

Timing analysis at the architecture level allows validating the system early in the development process. To do this, an abstract behavior model of the components should be specified such as durations of computations, which could be derived from a first evaluation of the defined components or from a worst-case-execution-time (WCET) analysis of the pre-existing other ones. Hence, modeling of time should be a central concern in component-based engineering for real time and embedded systems. Time characteristics must be included in different abstraction levels from the architecture of the whole system to the source code level.

In some architecture description languages, time properties are specified in structured or standardized plain English texts, and especially as a contract model. However, working with time properties of software components' assemblies is even more difficult, because loosely defined time concepts do not compose well and they cannot be used to build quality of service contracts. Moreover, in these languages, time properties haven't represented in sufficiently explicit manner [2].

In this paper, we present, a set of common and generic concepts allowing behavioral specification, in meta-modeling level, while integrating timing properties. Besides handling these timing properties, our software architecture description meta-model allows efficiently managing the large explosion of various behavioral concepts and relations among them. This offers to architects a complete and well organized definition of behavioral concepts that can be used to enhance architectural entities and models by specifying their behavior in a generic way. It allows describing and manipulating time as a separate dimension of component-based software architectures in order to improve the modularity at the software architect definition step.

This work is an extension of the ones described in [3], where we have proposed a meta-model, which supports behavior specification of software architecture. The high level definition of the meta-modeling concepts and their classification into four different perspectives, allowed integrating time relative properties in a generic manner. It

could be aligned with the MOF definition [4].

Describing software architecture from multiple perspectives provides more simplicity in using and understanding such description. Also, it offers more advantages for analysis of its functionality. Integrating multiple views representation for software architecture provides high level extensibility of complementary views or concerns of a software system. Indeed, in [5], Medvidovic and Roshandel have shown that a complete description of software architecture should be taken according four viewpoints: 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. It is described, in [5], 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 [5] 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.).

Our meta-modeling approach takes into account those aspects and integrates the time model at each of the four views. In doing so, 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 [6]. 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. This section focuses on the value of specifying behavioral aspects of architecture elements and presents main techniques used by academic and industrial communalities to specify behavior of architectural elements. In a third section we give a brief overview of the general properties in real-time embedded systems with respect to architecture description. Section 4 is reserved to describe the proposed multi-view description of architectural elements behavior. In section 5, we present what are timing properties and how they are used to express real-time embedded system characteristics and how they are integrated in the proposed meta-model. The paper is concluded with a discussion of given results 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 focus on techniques frequently used for specifying behavioral aspects of architecture elements. Specifying behavior is a way to add semantic detail to structural elements and their interactions that have time related characteristics. 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.

Architecture behavior description is supported by some 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 others representations. Connectors' behavior is defined especially in ADLs which model explicitly connectors as first class entities.

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 [7]. 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.

III. CBSE in Real-Time Embedded Systems

CBSE has been introduced to mange the growing complexity of modern software systems. This complexity is issued from the diversity and complexity of both functional and non-functional properties of systems. Mean of evaluation and analysis of these properties before programming phases is given by the description of system architectures using architecture description languages. This requirement is increased with real time systems, for which early simulation and validation steps are critical to assess safety properties. A real-time embedded system is characterized by a specific kind of tasks, called real-time tasks. A real-time task is defined in [8] as "an activity that is scheduled for handling in the entire system. It may be periodic, triggered by a periodic timer, or a periodic, triggered by an external interrupt".

A. Time Properties of Real-Time Embedded Systems

Many real-world computing systems are associated with time constraints [9]. These systems require that their own computations must complete before specific deadlines to ensure a safety execution without any damages. Such systems are called real-time systems. Typical examples are nuclear power plants, the military command and control, automatic manufacturing factories, crisis action management, and air traffic control systems. Real-time system within its timing constraints must accept any stimuli produced by the environment that is an important factor in this kind of systems. Real-time system takes into account timing properties and requires mechanisms to handle this kind of information. So, we must insure, in architectural

level, a well definition and handling of different timing requirements of the system and ensure that the system performance is both correct and timely. As examples of timing properties in real-time systems, we can talk about constraints on the execution platform, periodicity of tasks handling, execution time, etc.

B. ADLs for Real-Time Embedded Systems

Architecture specificities of real-time systems pertain more to non-functional constraints, such as different quality of service properties. Several models in real systems architectures must be improved by adding temporal characteristics. Various ADLs support time concepts in a heterogeneous granularity. They represent time characteristics as second class concepts. Such properties are generally attached, easily, to architectural elements. In real-time embedded systems, time properties must be represented and treated as a first class concept because they are much more difficult to handle them when they are directly linked to architectural entities.

AADL [10] uses hybrid automata to define the temporal constraints of its concepts. It integrates some variables to denote the time in hierarchical finite state machine. The expression of these temporal constraints is done in the level of states and guards over transitions. They express the timing characteristics about discrete transitions occurrence. AADL offers a binding mechanism to link software components to resources components. Time description in AADL is focused on resources models and lacks model elements to describe the application components themselves.

In **SysML** [11], different concerns are separated from each other. This separation is provided by SysML allocation mechanism to represent, in an abstract level, cross-associations among model elements with the broadest meaning. It differentiates three of many possible categories: behavior, flow and structure allocations.

MARTE [12] and the UML profile for scheduling, time and performance, add time and performance dimensions to some model elements. They base on various abstract concepts to specify timing constraints. The abstract concept of Time is a generalization of Instant and Duration concepts. *TimedEvent* is used in MARTE to express an event or behavior bound to time through a clock. So, time here is not a simple notation extension but it changes completely the semantic of the timed model elements. These profiles define concepts for modeling real-time embedded systems without precise semantics.

IV. A Four Views-Based Architecture 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 [13]-[14]-[15]-[16]-[17] 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 the four-view modeling technique [5] to allow a high abstraction level description of functional characteristics of software entities from four perspectives. In the following text of the section, the terms in italics have direct representation in the meta-model.

In order to have a more complete set of behavioral concepts, we have conducted a detailed study of the most notable ADLs and their supporting techniques to specify behavior: Darwin [18], Wright, [19]-[20], MetaH [21], UniCon [22], Rapide [23], and C2 [24]. Hereafter, we present, in Table I, the important concepts used by previous ADLs for specifying behavior of software architecture elements from the four functional viewpoints allowing the definition of views presented previously.

TABLE I
Behavioral Concepts from Four Views

Interface's behavior	Static behavior	Dynamic behavior	Interaction behavior
Event	Event	Guard	Event
Observed event	Post-condition	Transition	Interaction rule
Emitted event	Pre-condition	State	Interaction
Event alternation	Result	Activity	Synchrono-us interaction
Observed event	Process	Transition rule	Asynchro-nous interaction
Function call	Alternative activity	Source state	-
Message passing	Control activity	Targeted state	-
Event sequence	Indeterministic choice activity		-
-	Parallel activity	-	-
-	Sequence activity	-	-
-	Recursion activity	-	-
-	Deterministic activity	-	-
-	Format conversion activity	-	-

A. Interface Behavior View

Interface description is taken into account by several ADLs 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 such as Rapide [23] and Wright [19], describe behavior within interface description.

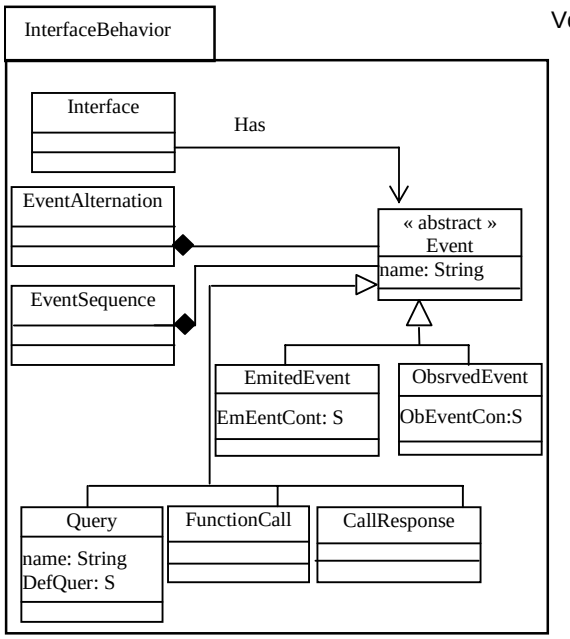

Fig. 1 Package 1: Interface behavior meta-model

The first package, depicted by Fig. 1, of the proposed meta-model, includes 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. An *event* is a class that abstracts all events used or generated by the architectural elements. An interface behavior may be generated by a sequence or an alternation of various events.

B. Static Behavior View

Static behavior view extends interface one with static behavioral semantics [25]-[26]-[27]. This extension is supported by several ADLs to represent behavioral characteristics in specific discrete state 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.

In this level we specify functional properties of an architectural entity in terms of its several *states and* relative operations or *activities.* 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 performed by an architectural entity in the context of software architecture.

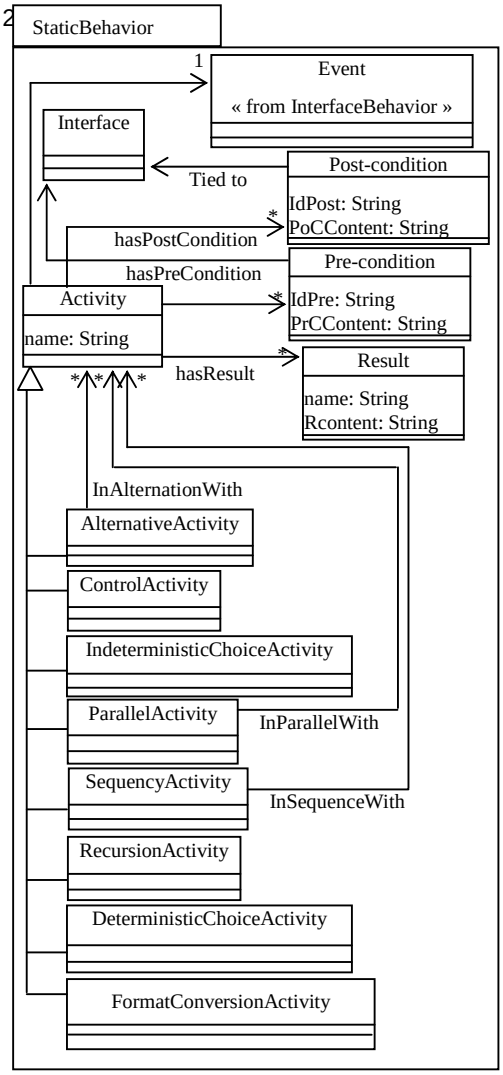

Fig. 2 Package 2: Static behavior meta-model

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. Dynamic behavior concepts can be inspired from UML statechart meta-model.

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.

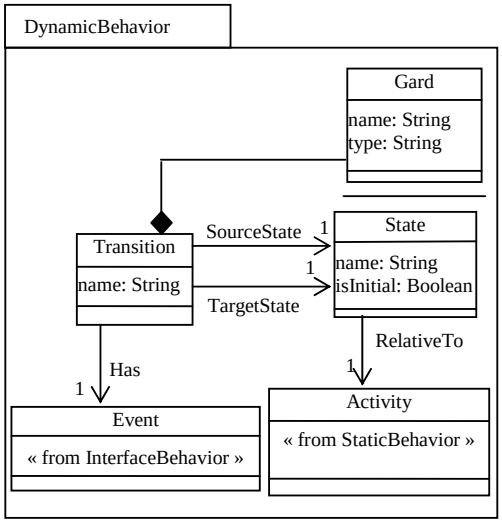

Fig. 3 Package 3: Dynamic behavior meta-model

D. Interaction Behavior View

This view is focused on representing of interaction 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.

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.*

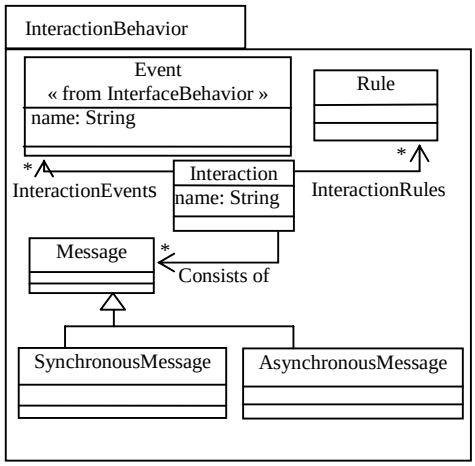

Fig. 4 Package 4: Interaction behavior

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

E. Structural Model of Views 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. The hole schema of the proposed meta-model is defined in Fig. 5, where different packages are interrelated by dependency relationships.

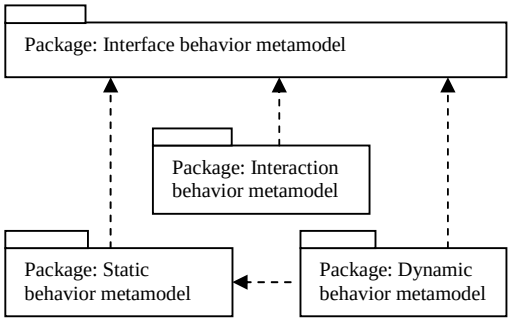

Fig. 5 Structural model of views' dependencies

V. Adding Time Properties

After defining all behavioral properties in the architectural entities in four distinct packages relative to different functional viewpoints, we add some timing properties into it. Time properties are important concept in any architectural aspect description, especially for real-time systems. In this section we show the addition of time attributes expressed in a meta-modeling level from four different views. The principle of this section is summarized in Fig. 6.

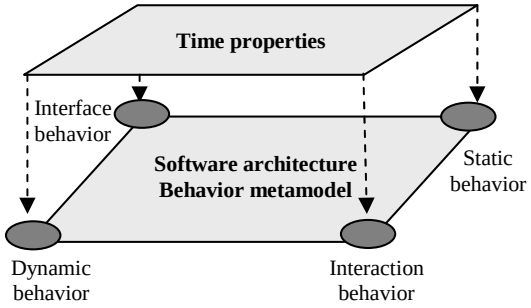

Fig. 6 Time dimension in the ADL metamodel

In order to add these time properties to the meta-model we will classify various timing characteristics by view. This multi-view based separation provides a high level of simplicity and extensibility in time relative properties definition. In elicitation of time constraints and concepts in order to add them to behavior in real time systems architectures, we have based on a set of time patterns extracted from various works such as [28]-[29] on real time systems specification. For example, we have elicited concepts of response time, delay, and period of service call, duration and execution time.

A. At the Dynamic Behavior View

In this level, we have firstly précised a number of basic concepts where we have, in next step, integrated time relative properties in our meta-model. Notice that in this view, as shown in Fig. 7, time is indispensable to express timing constraints on different component states (duration, transition time, transition duration, etc). A transition is represented in various works relative to real time systems engineering by an intermediate state. In our case transition is considered as a first behavioral entity. For which we define time characteristics in independent manner. Plus functional triggering conditions of the transition, we define also a set of time dependent enabling conditions to express at which time points the transition is possible. We define also a set of urgency attributes those are represented by Boolean attributes in the transition class: *isLazy*, *isDelayable*, *isEager* to indicate, respectively, if we have a lazy, a delayable or an eager transition. Urgency attributes allow controlling time progress at the semantic level and they are very adaptable in this architecture meta-modeling level. These variables are used to express next phenomena:

- *Lazy transition can wait forever,*
- *Eager transition has always the high level of priority. It never waits,*
- *Delayable transition can wait, but only until the falling edge of their time dependent enabling condition represented in the guard class.*

For each state we have defined a set of timing properties: *duration* of the state and its *triggering* and *ending* time.

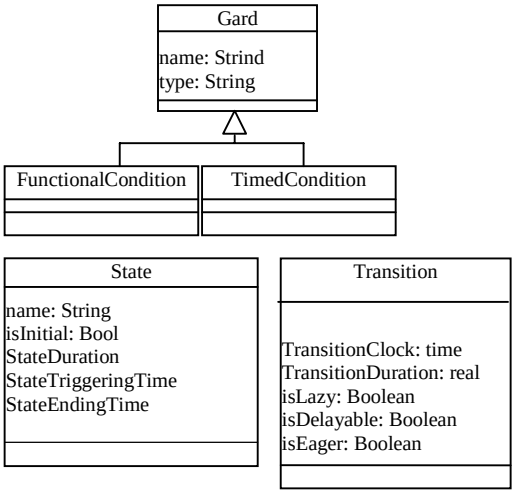

Fig. 7 Adding time properties into dynamic behavior view

B. At static Behavior View

Temporal characteristics are linked strongly to dynamic view of a system behavior. So, in static view we haven't numerous timing properties to define temporal characteristics of architectural elements behavior.

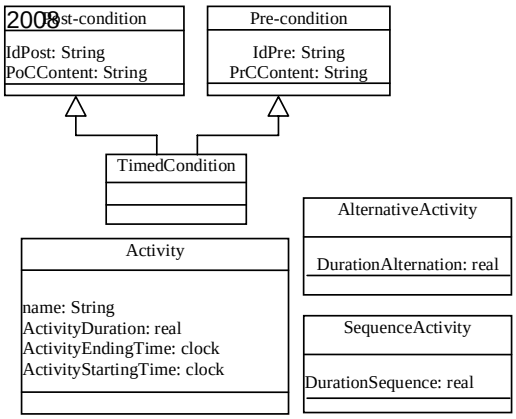

Fig. 8 Adding time properties into static behavior view

In this level we can provide some activities' timing properties such as duration of an activity, starting and ending times of activities' execution. For an alternative activity we define the delay between two alternations of the same activity. For each Sequence activity we define a timing property to express time spending to pass to other activity. Post conditions or preconditions can be timing conditions.

C. At Interface Behavior View

All interface behavior timing properties are event relative ones. In this view we talk about a timed event which is a specialization of a standard event on an architectural entity. A timed event is an event improved by a time value of timing information.

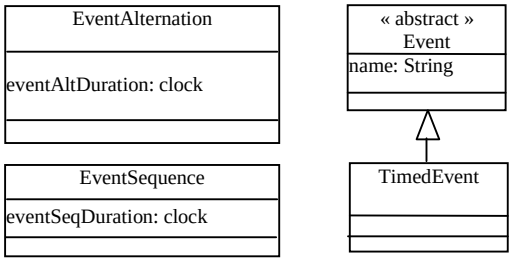

Fig. 9 Adding time properties into interface behavior view

Event alternations and event sequences can be constrained also by time information to express the delay between two event alternations or two events in the same event sequence.

D. At interaction Behavior View

Real time systems require some timing interaction rules to constraint their entities interactions.

In this level, we talk about timed rules those are extended to others interaction rules in various complexity levels of interaction definition.

The main timing information, like are shown in figure 10, are synchronization time used to express waiting times during a synchronous message, and message duration to express the time required for a message to arrive to its target component from its source component. These last attributes are represented as second class properties in level of Interaction and SynchronousMessage classes.

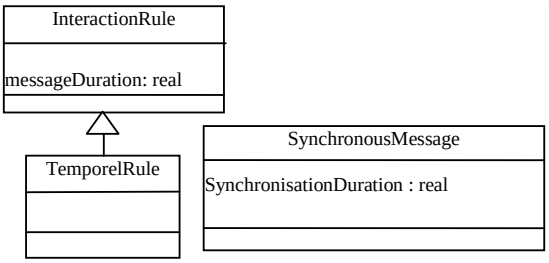

Fig. 10 Adding time properties into interaction behavior view

VI. Conclusion and Perspectives

In this paper, we argued that the software description language in the meta-model level makes the model generic and reduces the semantic gap which exists between different specific domains. Each specific system has its own key properties. Real-time embedded systems key properties are timing ones. These properties control and constrain behavior of these systems. So, we can't talk about real-time system software architecture behavior without specifying its time related characteristics.

Our proposed ADL meta-model incorporates, in the same meta-modeling level, a set of generic time concepts required to specify several timing properties of an architectural element to constraint and control behavior of an architectural entity.

In doing so, we have applied the separation of concerns principle by presenting four architectural views to completely modeling the software architecture in real-time embedded systems, while integrating the time dimension. The multi-view representation is used to modularize in separated categories some specific concerns like Behavioral description, structural description, deployment description, etc. It 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 software architectural entities. The high-level definition of behavioral concepts in the meta-model makes it an extensible, flexible, and opened model on different transformations into other models. This is particularly important for real-time embedded systems, where variability, flexibility and evolution are key success factors.

Currently, we are studying different possibilities to adapt and enrich our meta-model in at least two directions: i) studying and evaluating appropriate languages and techniques to formally specify behavioral and timing attributes of our meta-model, and, ii) extending the MOF meta-model corresponding packages to cope with the behavioral aspects in the ADLs.

References

[1] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in bip. In SEFM '06: Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society.

[2] Sébastien Saudrais, Nöel Plouzeau, and Olivier Barais. Integration of Time Issues into Component-Based Applications.

[3] A. Ibrahim and M. T. Kimour. Integrating Behavioral Aspect into COSA Architecture. Proceedings of SEDE-2007 16th Int'l Conf on Software Engineering and Data Engineering, USA, 2007.

[4] OMG: MOF 2.0 Core, OMG document ptc/06-01-01, Jan 2006.

[5] 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.

[6] Didonet Del Fabro M., Bézivin J., Jouault F., Breton E., Gueltas G.: AMW: a generic model weaver. In: *Proceedings of the 1st day on Model-Based Engineering (MDE)*, 2005.

[7] 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.

[8] Zonghua Gu, Shige Wang, Jeong Chan Kim and Kang G. Shin. (2004-01-02). Integrated Modeling and Analysis of Automotive Embedded Control Systems with Real-Time Scheduling. (Electronic). Accessible: <*http://kabru.eecs.umich.edu/aires/paper/gu_sae04.pdf*> p.3. (2006-06-08).

[9] Bruce Douglass, Gary Cernosek, *Real-time UML: Developing Efficient Objects for Embedded systems*, Addision-Wesley Inc., 1998

[10] SAE: Architecture Analysis and Design Language (AADL). (2006) document number: AS5506/1.

[11] OMG: Systems Modeling Language (SysML) Specification. (2006) OMG document number: ad/2006-03-01.

[12] OMG: UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), Request for proposals, Object Management Group, Inc., 492 Old Connecticut Path, Framing-ham, MA 01701. (2005) OMG document number: realtime/2005-02-06.

[13] Booch G., Jacobson I., Rumbaugh J. "*The Unified Modeling Language User Guide*", Addison-Wesley, Reading, MA.

[14] 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.

[15] 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.

[16] Krutchen, P.B. "The 4+1 View Model of Architecture", *IEEE Software 12*, pp. 42 - 50, 1995.

[17] 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.

[18] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. "Specifying distributed software architectures". *Proc 5th European Software Engineering Conference*, September 1994.

[19] 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.

[20] R. Allen, and D. Garlan, "The Wright architectural specification language", *Technical Report of CMUCS- 96-TBD*, CMU, School of Computer Science, September 1996.

[21] 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.

[22] M. Shaw, "Abstractions for Software Architecture and Tools to Support Them". *IEEE Transactions on Software Engineering*, 1995. 21(4): pp. 314-335.

[23] 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.

[24] 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.

[25] 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.

[26] Liskov B. H., Wing J. M., "A Behavioral Notion of Subtyping", *ACM Transactions on Programming Languages and Systems*, November 1994.

[27] Zaremski A.M., Wing J.M., "Specification Matching of Software Components", *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, 1997.

[28] Bruce Powell Douglass. *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[29] O. Florescu, J. Voeten, H. Corporaal Modelling Patterns for Analysis and Design of Real-Time Systems. *Technical Report ESR-2006-05, ISSN 1574-9517.*

A. 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. His email address is bessamamar@yahoo.fr.

M. T. 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.