Real-Time Architecture Description and Quantitative Analysis using UML

C. André, F. Mallet, M-A. Peraldi-Frati
I3S Laboratory (CNRS/UNSA/INRIA)
Sophia Antipolis (F)

Abstract
In this paper we present our contribution in the domain of real-time system design. This approach is twofold: an UML2.0 and SysML-based architecture description including functional and structural specification, resource allocations and QoS modelling; a quantitative analysis with formal description of behaviour using Time Petri Net. The quantitative analysis explores the solution space and proves the existence of a valid scheduling. A refinement process allows different levels of description of the application. An example illustrates the approach.

1 Introduction

The objective of this work is to represent a model of a real-time distributed application and to prove the existence of a valid deployment for a given physical architecture. Starting with a system-level description of the application, we devise an associated UML 2.0 activity diagram, which describes functions in terms of data and control flows. Activities are then mapped onto *assemblies*, a stereotype of Class from the UML StructuredClasses package defined in SysML (System Modeling Language). SysML [OMG_SysML05] is part of the response submitted by the SysML Partners to the UML for Systems Engineering RFP. Dependencies from activity diagrams (behavioural view) to assembly diagrams (structural view) are expressed by *allocations*, another concept from SysML. We adorn the allocation dependency relationships with Qualities of Service (QoS) characteristics useful for performance evaluation. Our assemblies come from a component library consisting of *processing*, *communication*, and *analysis* hierarchical elements. Each component is associated with a behavioural model: a *hierarchical Time Petri net.*
From the behavioural and structural models we construct a Petri Net model in a systematic way. This net is then exported to Tina [BeRiVe04], a tool dedicated to Time Petri Net analysis. Specific analysis components from our library directly express qualitative and quantitative properties, like deadline satisfaction. These properties are then checked by Tina.
Another specificity of our approach is to rely on the *synchronous hypotheses* [BeCa03] for modelling, and for design as well. This highly simplifies behavioural specifications and their composition.
The different phases of our approach are illustrated by an example presented in Section 2.

2 Application example

This example is a simplified version of a control and signal processing application. Operations stand for complex atomic data processing. This is a TLM (Transaction Level Model [CaGa03]) description where an operation may be an IP (Intellectual Property) such as an FFT, a convolution, a filtering, etc.
This application consists of 4 input signals (from sensors), 3 output signals (to actuators) and 3 operations (oper1 to oper3). From the *functional point of view*, the system may operate in two modes (M1, M2) selected by the input M. A functional specification of the expected behaviour is:

$$\left.\begin{array}{l} W = oper1(C) \\ Z = oper2(C) \end{array}\right\} \text{if} \quad M = M1$$

$$Y = oper3(oper1(A), oper2(B)) \quad \text{otherwise}$$

The execution platform is given: 2 processors (P1 and P2) connected by a channel. Extra functional (also called non-functional) constraints are imposed. The first constraint is a deadline (a period of 38 time units, equal to the deadline). Others concern deployment: some processing elements have fixed location; others have to be mapped onto physical resources so that real-time constraints are met.
With the knowledge of the performances of the platform elements (processors and channels), a constant execution time can be associated with pairs (processing element, processor). An inter processor communication has a fixed duration of 4 time units. In Figure 1, inpX (outX) stands for the acquisition (actuation) processing of signal X.

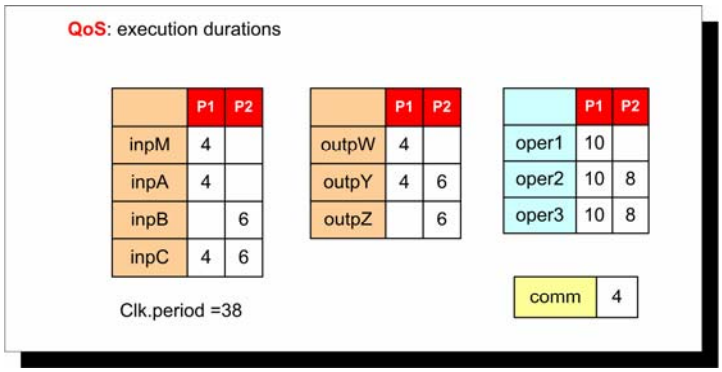

Figure 1 : Execution durations for processing and communication elements.

This example is often used as an illustration of the SynDEx AAA methodology [GrSo03] that focuses on the adequation between algorithm and architecture (timeliness and optimisation). Even though the goal is the same, our approach is different and follows the MDA (Model Driven Architecture) flow suggested by the OMG. It relies on UML 2.0, existing profile (Schedulabitity, Performance, and Time: SPT [OMG_SPT05]) and forthcoming profiles (System Engineering [OMG_SE03], Marte [OMG_Marte05]).

3 From functional description to activities

In UML 1.x activity graphs were just an *informal* specialisation of state machines. Such a representation was not convenient for system engineers. To address this issue, explicit representation of data and control flows has been introduced (influence of the SysML community) through activity diagrams. Now, in UML 2.0, activities are first class concepts with their own diagrams.

3.1 From equations to activities

An activity is a UML behaviour. It specifies a partial ordering of executions of subordinate behaviours, using control and data flow models. Activity diagrams support hierarchical description; subordinate behaviours are individual elements (actions) that can be invocation actions or structured activity nodes. Modes are selected by a decision node. A decision input is a behaviour attached to a decision node, which selects one of its outgoing edges.

Depending of the operating modes, alternative sets of inputs and outputs are used. The corresponding representation for this is the ParameterSet from the UML CompleteActivities package.

Access to information demands special actions, which can be resource and time consuming. We represent explicitly these accesses using input and output actions, named inpX and outpX, where X is the parameter name. This extension, applied to M1_Activity, results in the activity diagrams of the Figure 2.

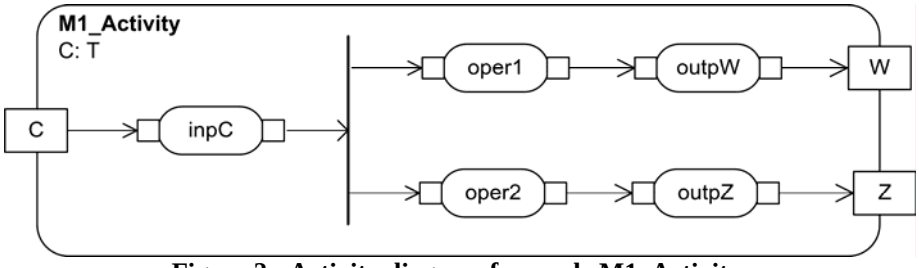

Figure 2 : Activity diagram for mode M1_Activity.

Since activities are behaviours their instances are UML executions, which are, in our case, expected to be deterministic. To meet this objective we use a synchronous semantics such as defined in synchronous languages. A synchronous system evolves in a sequence of non-overlapping reactions in a lock-step manner. A typical synchronous execution scheme, for a reaction, consists of a *read phase* (acquisitions), a *computation phase*, and a *write phase* (actuation). The sequence of these three phases is called a *reaction* and must be performed in isolation. UML activities provide a mustIsolate attribute that models this isolation requirement. Moreover a synchronous execution requires finite execution and respect of causality relations. Details about synchronous execution semantics are beyond the scope of this paper (see [DeAn03]).

3.2 *Taking location constraints into account.*

A partition of the activity diagram is done according to the location constraints. For instance on Figure 3, for M1_Activity, oper1 and outpW must execute on processor P1 (upper swimlane). outpZ must execute on P2 (lower swimlane). The intermediate swimlane contains the not pre-located actions. A first extension to classical activity diagrams is the introduction of «tight» that is a stereotype of the metaclass ActivityEdge. It imposes the same location for the source and the target of the edge. It is often used for linking an input/output to its handler.

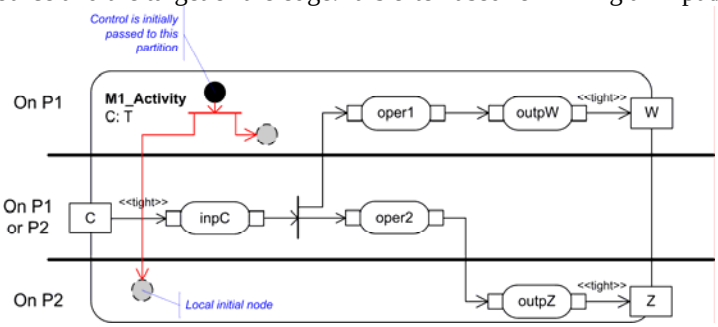

Figure 3 : Partitioned activity diagram

Another extension is related to control flow. Since input M and the associated decision node are located on P1 (location constraint), the control flow is passed to the upper partition, denoted by the initial node (solid circle). A fork node leading to grey circles, which are "virtual" initial nodes, propagates control to the (physical) partitions.

4 Mapping activities onto assemblies

4.1 *Element library*

Our library consists of hierarchical resources (assemblies) that can be interconnected with UML connectors in order to form a directed graph. There are three kinds of resources: processing (PE), communication (CE), and analysis elements (AE). A processing element specializes in a communication element, which processes communication function. Moreover, it is not allowed to connect two CEs directly.
The PE stereotype owns a property: isPhysical: Boolean, stating whether or not the PE represents a physical element. A "non-physical" element must not contain any physical ones.

4.2 *Behaviour of the library components*

The intra-component behaviour is generally specified by UML state machine. UML state machines, which can process only one event at a time, are not suitable for synchronous modelling where simultaneity of events is the rule. Moreover, we would like to compute the behaviour of architectures by composing the behaviour of their components. For these reasons, we have chosen to model the behaviour of our components by a non UML behavioural model: Petri Nets. Even though activity diagrams are widely inspired from Petri Nets, they do not have so well-established mathematical foundations and do not offer analysis capabilities.
Thus, the behaviour of a component is described by a Hierarchical Place/Transition Net (using the Petri Net Markup Language–PNML) making it possible to compose behaviours. Timing information can be attached to transitions (Time Petri Net). The Petri Net analysis tool named Tina supports both timed and untimed nets. However, it does not deal with hierarchical descriptions. We built a tool that allows for graphical composition of Petri Net *modules*. It exports flattened modules into a Tina-compatible format.

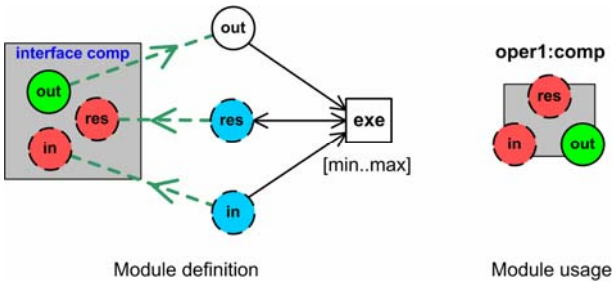

Figure 4 : Generic Petri Net for the «Comp» element.

With each library component, the tool associates a generic Petri Net module. As an example, Figure 4 shows the generic Petri Net for the «Comp» element (a PE specialized in computation). The transition (exe) represents the

computation. This transition can be labelled with an optional time interval standing for the execution time. Places represent resources: res is the availability of the physical resource; in and out are the data input and output resources. In the grey box, the *interface* of the module includes import and export (dashed outline circle) places.

4.3 Mapping rules

After specifying the behaviour we have to model the *structure*. We use SysML assembly (see Figure 5), a stereotype of Class from the UML StructuredClasses package. Each swimlane of the activity diagram is allocated to a processing element assembly (PE). A high-level assembly contains processing and communication elements allocated from activity nodes and activity edges respectively. Physical element assemblies are used only for resources with fixed location. The processing element Anywhere, which is not a physical element, contains all the other resources including the potential communication elements. An activity edge is allocated to either a usual connector when both its source and its target are on the same physical resource or an explicit communication element otherwise. Note that control nodes of the activity diagram appear as explicit processing elements (CtrlFork and DataFork in the figure).

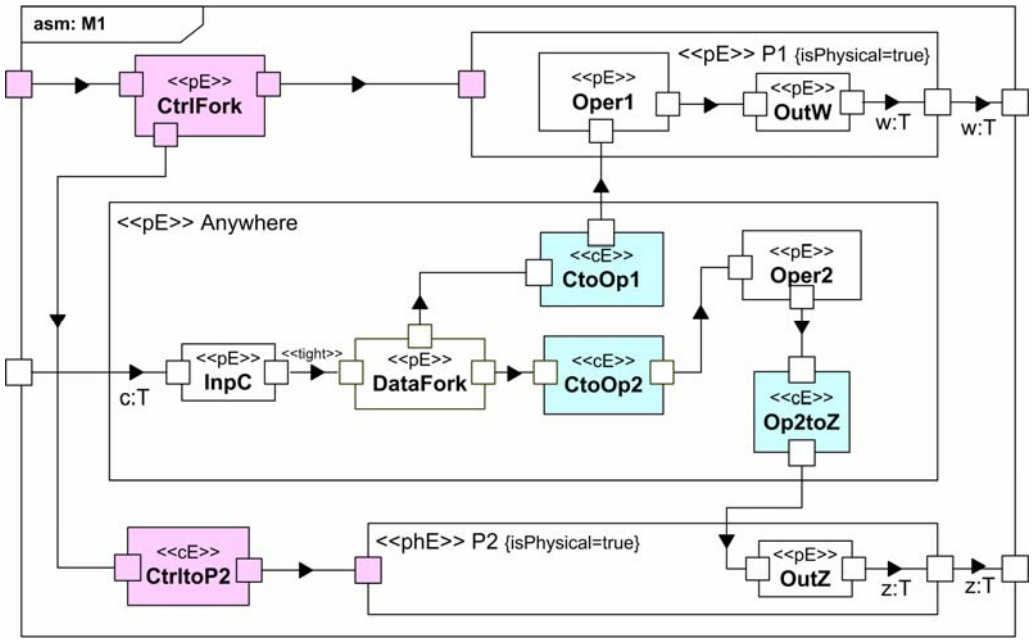

Figure 5 : Assembly for mode M1

The SysML concept of *allocation* provides a way to specify a mapping. It is a UML dependency relationship between a *client* and one or several *suppliers.* Additional extra-functional information, such as deadlines, execution time, power consumption or any other QoS, needs to be attached to this relationship. UML addresses this issue for deployment by introducing DeploymentSpecification. Likewise we introduce AllocationSpecification to be used with SysML allocations.

5 Architecture exploration

5.1 Building the analysis model

Starting with the architecture model above mentioned, we enrich it with extra functional information extracted from quality of service characteristics such as defined by the SPT profile. For instance, in addition to temporal constraints, we can take account of the concurrency degree of physical resources (processors and channels) and of the resource sharing constraints (number of simultaneous accesses).

We generate the hierarchical Petri Net model for the full architecture (see Figure 6). This model preserves the potential allocation freedom expressed by the Anywhere processing element. Consistency rules ensure that communication elements are allocated to the right channel depending on the allocation of processing elements that use this communication. For example, if both InpC and Oper2 are allocated to the same processor, the CtoOp2 communication element is also allocated to this processor and therefore is not time consuming. The respect of consistency rules is imposed by specialized analysis elements (AE). Other AEs are used as observers of required QoS characteristics.

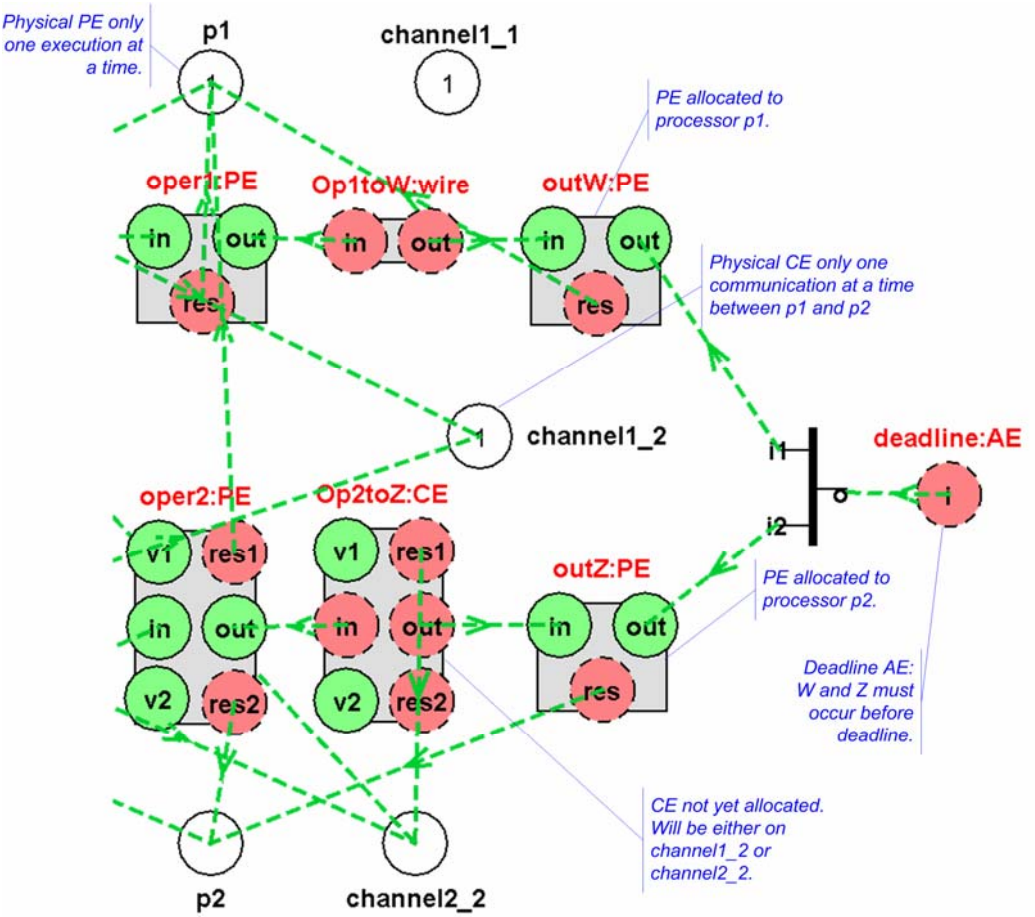

Figure 6 : Part of the Modular Petri Net for mode M1

5.2 Performing analysis

The resulting Time Petri Net model is analysed by Tina. When the reachability set is bounded, Tina builds a behavioural graph whose vertices are called state classes (a marking + a set of firing time intervals), and whose edges are transition firings. The synchronous hypotheses lead to acyclic Petri Nets, far easier to analyse. Tina implements reachability analysis algorithms that detect unboundedness, deadlocks and dead transitions. Dead transitions cannot be fired whatever the execution. The first two properties usually unveil a misconception. We use the existence of dead transition to detect deadline violation (see Figure 7).

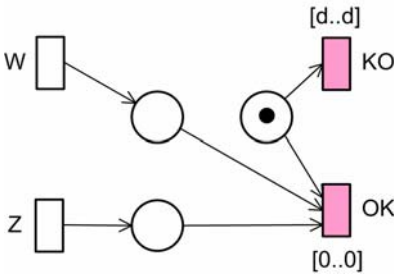

Figure 7: Petri Net model for deadline detection

Figure 7 illustrates how we deal with deadline violation in mode M1. For each possible reaction, we must check whether or not both W and Z occur before a given relative deadline (d time units). The Petri Net contains two conflicting transitions (OK and KO). OK is fired when the deadline is met, KO is fired otherwise.
When the OK transition is dead, there are no allocations satisfying the deadline. When the KO transition is dead, all allowable allocations satisfy the deadline. If another transition is dead, wherever it is, there is a design flaw with dead parts in the architecture. When no transitions at all are dead, there exists (at least) one allocation satisfying the deadline. We need further analysis to characterise the timely solutions.

6 Conclusion

We have shown a way to model a real-time application using UML and SysML. A functional description is captured using activity diagrams. We propose to use swimlanes as an intermediate notation to combine functional and structural descriptions. We provide a library of generic assemblies used to allocate activities to structural information. Mapping rules are given to create the assembly diagrams in a systematic way. Qualitative and quantitative QoS characteristics are associated with activity/assembly pairs using an allocation specification. For analysis purpose, temporal QoSs are used in hierarchical Time Petri nets through the integration of pre-defined analysis elements. Reachability analysis tools of Tina establish the existence of a valid deployment meeting QoS constraints.

Several extensions of this approach are possible. First, more complex properties (expressed using Linear Time Logic–LTL for instance) could be formally analysed by a model checker, Tina supplying the behavioural graph. For instance, with the given parameters, a manual graph analysis has shown that oper2 must necessarily be deployed onto the processor P2 in order to meet the deadline. This leads to a reduction of the possible deployments to be explored. Such a procedure would benefit from being automated. Once the adequate solutions are better characterised, we may export pertinent information extracted from UML and SysML models to other analysis tools. For instance, we could easily export the algorithm and architecture models to SynDEx for further optimisation and generation of the real-time distributed code.
Second, we have demonstrated how to associate time Petri nets with our library elements; we could use other formalisms instead. In the future, to make the best of the underlying synchronous hypotheses, we intend to use the industrial synchronous language Esterel and its validation tools.

7 References

[BeCa03] A. Benveniste, P.Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, R. De Simone *"The synchronous languages 12 Years Later"*, Proc. of the IEEE, Vol.91, n° 1, pp. 64—83, 2003.

[BeRiVe04] B. Berthomieu, P-O. Ribet, F. Vernadat, *"The tool TINA – Construction of abstract state space for Petri net and Time Petri Nets"*, Int. Journal of Production Research, Vol. 42, n° 14, pp. 2741—2756. July 2004. http://www.laas.fr/tina

[CaGa03] Lucai Cai and Daniel Gajski, "Transaction Level Modeling: An Overview," *Proceedings of the International Conference on Hardware/Software Codesign & System Synthesis*, Newport Beach, CA, October 2003.

[DeAn03] R. De Simone, C. André *Towards a "Synchronous Reactive" UML subprofile?*, .I3S Research Report N° RR200326 presented at SVERTS'03 Specification and Validation of UML models for Real Time and Embedded Systems, Octobre, 2003, San Francisco (US).

[GrSo03] T. Grandpierre, Y. Sorel "From Algorithm and Architecture Specifications to Automatic Generation of Distributed Real-Time Executives: a Seamless Flow of Graphs Transformations".*MEMOCODE2003, Formal Methods and Models for Codesign Conference*, Mont Saint-Michel, France, June 2003

[OMG_Marte05] "UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) RFP", *OMG Document realtime/05-02-06*, February 2005, *http://www.omg.org/docs/formal/05-01-02.pdf*

[OMG_SE03] "UML profile for Systems Engineering RFP", *OMG Document ad/03-03-41*, September 2003, *http://www.omg.org/docs/ad/03-03-41.pdf*

[OMG_SPT05] "UML Profile for Schedulability, Performance, and Time, version 1.1". *OMG document formal/2005-01-02*, January 2005, http://www.omg.org/docs/formal/05-01-02.pdf

[OMG_SysML05] "Systems Modeling Language (SysML) Specification. v 0.9". *OMG document ad/05-01-03*. January 2005. http://www.omg.org/docs/ad/05-01-03.pdf