A SysML profile for mechatronics integrating Bond Graphs

SKANDER TURKI THIERRY SORIANO ADEL SGHAIER
LISMMA (EA 2336)
Supmeca
Maison des technologies, Toulon 83000
FRANCE

Abstract: – In this paper we present a SysML [1] profile for mechatronic-systems design. It is an extension to the activity diagram of UML 2.0 so that it maps to the Bond Graph formalism which is a useful tool when dealing with mechatronic systems. This extension is a package that contains a set of stereotypes and tagged values which are the extension mechanisms that SysML inherits from UML2.0 [3]. We establish mapping between bond graph elements and these extensions. We illustrate this SysML profile by an example.

Index Terms: – Mechatronics, Bond Graphs, SysML, UML 2.0, Activity diagram, Systems Engineering, Profile.

1 Introduction

Mechatronic systems are the result of the integration of mechanical, electronic and information technologies. The complexity of such systems needs to apply a systems approach. The systems approach is a global multidisciplinary methodological approach that aims at master the practices of systems engineering. To support such an approach SysML has been introduced. SysML is an essay to provide systems engineers with a standard language that covers the specification, analysis, design, verification and validation phases.

For mechatronic-systems engineers Bond Graphs is a very useful tool. Bond Graphs is a formalism used to depict the energetic transfers between subsystems of different natures (mechanical, electrical, etc). It is today used in numerous important projects inside companies such as PSA, Renault, General Motors…[5]

In this work we built a profile extending SysML to integrate Bond Graphs. We start introducing SysML. Then we present the Bond Graph formalism. In the third section is a detailed presentation of our contribution that is composed of two sub-profiles. The first one is for Bond Graphs, the second is for block diagrams as they are usually combined to express both the physical sub-system and the control sub-system. Then we give an example. Finally we give some conclusions and perspectives of this proposal.

2 SysML : A new modeling language

SysML is an extension to UML 2.0. It is the Systems Modelling Language built as a response to the Object Management Group's UML for systems engineering Request For Proposal [2]. This RFP claims that systems engineers need a standard language easy to integrate both in the engineering teams and in the existing tools. Other UML-based approaches for systems engineering have already been introduced like the UML/PNO approach [12], but such approaches are not rich enough to give to systems engineers all the expression possibilities they may find in SysML. The OMG's RFP asserts that the language must help heterogeneous teams (software, electronics, mechanics…) to work together and communicate. This is why SysML is based on the minimal subset of UML2.0 that satisfies systems engineers needs. It is intended to be minimal to be easily accepted by the systems engineering community. In addition to that SysML is an extension to UML2.0 and tries to bring the minimum change to the UML meta-model to facilitate its implementation to tool vendors. It benefits from the UML extension mechanisms (stereotypes, tagged values). In the "UML 2.0 infrastructure specification" [4] page 174 a stereotype "defines how an existing meta-class may be extended, and enables the use of platform or domain specific terminology". These mechanisms will be used to specialise SysML to specific domains such as aeronautics, automobile etc. SysML is also aligned with other standards such as ISO AP-233 [8] for data interchange to support tool interoperability. SysML also inherits the XMI (XML model interchange) from UML2.0. But this basic SysML will need to bring to these systems engineers their usual tools by creating extension packages that can be added or removed from their design environments just like the practices of the software engineers with the profiles of C++, CORBA and others. This is why we need to build these extensions for SysML. The activity diagram is already used to express EFFBDs (Enhanced Functional Flow Block Diagrams). BGs are also usually used by systems engineers. In [11] W. Borutzky establishes a relationship between Bond Graphs and object-oriented modelling. So, we are presenting their application

using SysML. The BG formalism is quickly introduced in the following.

3 Bond Graphs notation

Mechatronic-systems design often requires to analyse the energetic transfers between subsystems. In addition a homogenisation of such an analysis must be accomplished to visualize on the same diagram a phenomenological description of the system. This is done by the BG formalism that allows in the same time to discover the system's equations, which are used for the simulation and the dimensioning. There are three levels of BG representation [6]. The word-BG that is used to have a first approach in describing the energy map of the system and its composition. The Acausal BG used to show the energetic transfers and subsystems natures (i.e: energetic functions) assuming that the decomposition is advanced enough so that we can replace each subsystem by an elementary component. The causal BG which helps finding the system's equations. In the BG formalism, a subsystem is represented by a closed line (system's frontiers) with a name. For each energy interchange with its environment we associate to this subsystem an energetic port of a defined type (mechanical energy, electrical energy, etc). A port is represented by a unidirectional-semi-headed arrow and carries the data of the power transported; Effort and flow that correspond to a couple of variables in each energetic domain (tab 1).

Table 1: Effort and flow in different energetic domains.

Energetic domain	**Effort e**	**Flow f**
Translational mechanics	Force	Velocity
Rotational mechanics	Torque	Angular velocity
Electricity	Voltage	Current
Magnetic	Magneto-motive force	Magnetic flow

3.1 Elementary components or nodes

The elementary components are classified (tab 2) by their energetic behaviour or function.

In addition to these elements, each element may be modulated except I and C. In this case an M is added (MR, MSe, MSf,…).

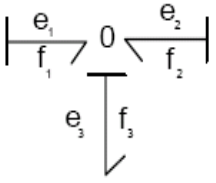
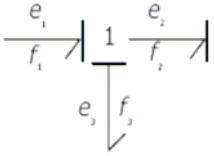
3.2 The junctions

Junctions (tab 3) are used to associate those elementary components. They transmit the energy instantaneously. They must connect a number of arrows higher than 1.

Table 2: Bond Graph Elementary components

Active elements	$S_e \longrightarrow$	Effort generation.
	$S_f \longrightarrow$	Flow generation.
Passif elements	$\longrightarrow R$	Energie dissipation node.
	$\longrightarrow I$	Effort storage node
	$\longrightarrow C$	Flow storage node.
Sensors	$\longrightarrow D_f$	Flow sensor.
	$\longrightarrow D_e$	Effort sensor.
Conversion elements	$1 \xrightarrow{m} TF \longrightarrow 2$	Transformation implying : $e_1 = m\,e_2 \;;\; f_2 = m\,f_1$
	$1 \xrightarrow{r} GY \longrightarrow 2$	Transformation implying: $e_1 = r\,f_2 \;;\; e_2 = r\,f_1$

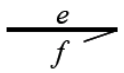
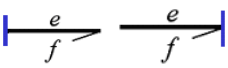
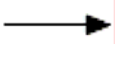
Table 3 : The two kinds of junctions

Junction 0 : all efforts are equal ***Ex:*** Parallel connection in electrics.		$e_1 = e_3$ $e_2 = e_3$ $f_3 = f_1 - f_2$
Junction 1 : all flows are equal ***Ex :*** Series connection in electrics.		$f_1 = f_2$ $f_3 = f_2$ $e_2 = e_1 - e_3$

3.3 The arrows or bonds

In BGs there are two types of edges (Tab 4). The first shows an informational transfer and the second shows an energetic transfer. The first one is represented by a full headed unidirectional arrow. The second, by a semi-headed unidirectional arrow. In the case of a causal BG, a vertical line is added on one of the extremities of the arrow. The energetic arrows are assigned a number for identification.

Table 4: Bond Graph's bonds/arrows

Energetic transfer		Informational transfer
No causality	With causality	
		

4 Mapping to SysML/UML constructs

4.1 Candidate diagrams

It is important to choose a diagram that will be easily recognisable as a BG diagram to limit the learning

effort of systems engineers. In the other hand, it is essential to respect the semantics of the UML diagrams. For example, a final state in a UML statechart is defined as a state that cannot have any outgoing transition (self.outgoing->size()=0, [3] page 581). This constraint must be respected for any extension added to the original diagram.
In SysML the diagrams used to describe behaviour are: Use cases, interaction diagrams, parametric constraints diagram, statecharts and activity diagrams.
Use cases cannot express control nodes and are used to express top level system requirements. They cannot express energy or information transfer. Interactions are to be avoided because of the life line representation of objects which is too much far from BGs. Parametric constraints are used to associate objects properties to express mathematical relations between physical variables which may be useful when we'll need to extract the system's equations. It is not useful at this stage. Statecharts can only represent control flow and not object flow.
Activity diagrams are the most appropriate view because they use constructs that express object transfers and control. They combine system's composition with communication and sequencing between actions. It can also be useful to compare this proposal with a representation of BGs that uses diagram assemblies. In fact, they depict a system as a collection of components with specific roles. They also show connections between subsystems inside the hole system ([1] page 49). It seems that the assembly diagram is closer to BGs than activity diagrams. In fact assemblies are used to depict the system's composition in a static way. This is the opposite of activity diagrams that are intended to depict a sequence of actions. Of course we can "just not take into account" this actions sequencing, but there is still a risk of misunderstanding of BGs when expressed by activity diagrams. On the other hand the assembly diagram is composed of very general constructs, it also doesn't have control nodes which is useful to depict junctions.

4.2 Mapping Bond Graphs to activity diagrams

4.2.1 Elementary components or nodes

With the BG formalism, systems are decomposed until we obtain subsystems that can be assimilated to an elementary-energetic phenomenon. We can then assimilate an elementary subsystem to an "action" as it is defined in the UML2 specification: It is the fundamental unit of behaviour specification. It takes a set of inputs and converts them into a set of outputs."[3] page 229. An *Action* is defined as an abstract class. Then we have to inherit from it. We must remember that one of the advantages of SysML is that it will benefit from UML tools. This is why we cannot introduce contradictions between our extension and UML2.0 as it is the case in B Fig.1 where we need to add new constructs in the MOF (meta-Object Facility), which is also the meta-model of UML2.0.

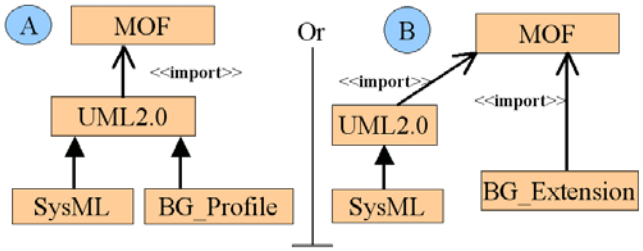


Fig.1 SysML : Profile of UML2.0 or new elements.

So case A of fig.1 is chosen; an extension of UML2.0 is built.. Applying this configuration, we can use two different solutions. First, we can add another child class to *Action* (Fig 2) and call it BondGraphAction. Second we can use one of the existent child actions and map it to a Bond Graph node.

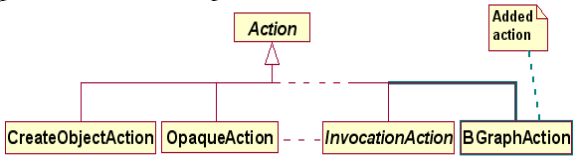


Fig.2 Adding a child class to *Action* in the meta-model.

The UML meta-model is usually not open to modifications. The second solution seems then to be more easily realisable. This is why we use the OpaqueAction meta-class to represent a Bond Graph action. OpaqueAction is composed of a set of inputs, a set of outputs, has a string that contains the body of the action (in our case the mathematic formula) and has another string that specifies the language in which that formula is expressed. Fig 3. [3] page 233. This representation of an elementary component is equivalent to the "general multiport component" described by Hales and Rosenberg in [10].

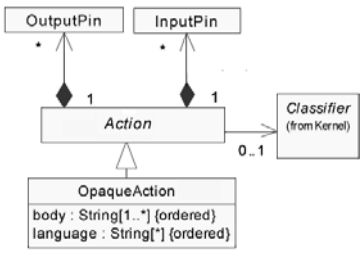


Fig.3 OpaqueAction's meta-model.

An OpaqueAction is defined in [3] page 280 as an action that has been introduced for implementation specific actions. This is why it is the most general specific action of the specification. To use the OpaqueAction, we will specialize it for each of the Bond Graph functions. We will not use inheritance of

classes because that will imply that we will have access to the metamodel but we will use the extension mechanisms of UML like stereotypes. Stereotypes are used to add semantics to a metaclass its advantage is that it is accessible in the UML tools. In this Fig 4 we show the stereotypes created for the Bond Graph elements. The inheritance arrow used is full-headed to express extension on the meta-model, see [4] page 167. This way we can express on the same diagram, the three levels of Bond Graphs for better understanding. We use the Body attribute Fig 3, to express the mathematic formula that goes with the element. This formula can be used later for system's equations generation. We used tagged values to add a property "factor" to the stereotyped class BGraph_TF.

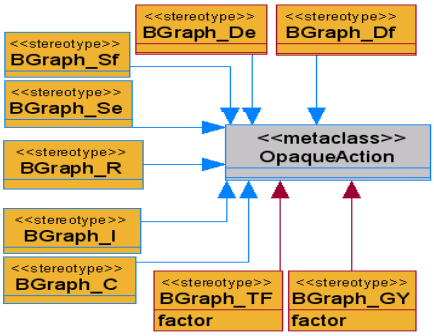


Fig.4 OpaqueAction stereotypes.

4.2.2 Junctions

Bond Graph junctions correspond to the control nodes in the activity diagram. In the meta-model, control nodes inherit from the abstract class *ControlNode*. To represent the two junctions of BGs we can inherit two new nodes of control Fig 5, 6.

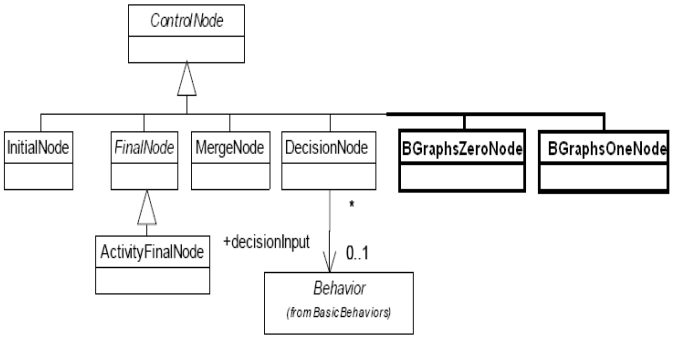


Fig.5 ControlNode's hierarchy

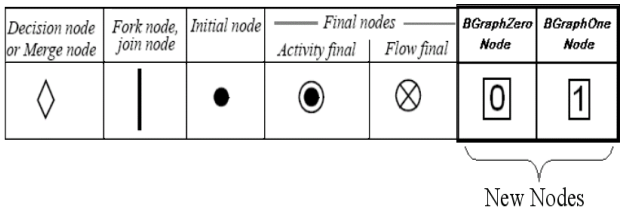


Fig.6 ControlNode notations

We can also use together the decision node and the merge node represented by a diamond for both junction 0 and junction 1. This can be done by stereotyping those nodes Fig 7. In fact the merge node and decision node can be used in the same diagram element [3] page 417. This is the better solution because it prevents us from accessing the meta-model, we just use the extension mechanisms.

In Fig 7, we use a BGraph_junction abstract class to inherit from both MergeNode and DecisionNode. Then we stereotype this node to BG_0 and BG_1. Both resulting nodes will be drawn using the diamond.

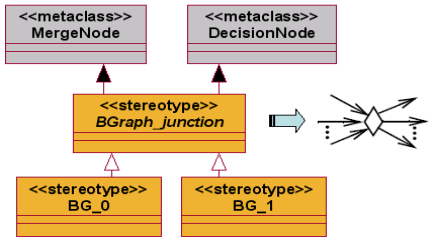


Fig.7 Bond Graph nodes hierarchy

4.2.3 The edges or bonds

Two different edges are defined. We use the stereotypes/tagged values mechanism to define these child classes of the UML 2.0 ObjectFlow class. The black arrow in Fig 8 means it is an extension to the meta-class ObjectFlow. The attributes of the new defined classes are also called tagged values. The causality attribute/tagged value is of type CausalityType which is an enumeration (start, end, nonCausal).

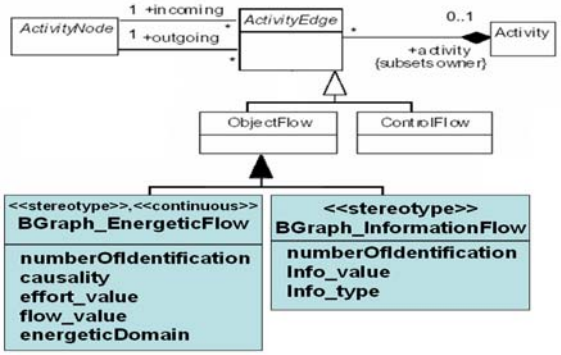


Fig.8 ObjectFlow stereotypes for Bond Graphs

The energetic transfer is done through an energetic port in BGs. They are represented by pins Fig 9

Fig.9 Energy bond representation.

Object flow of activity diagrams in UML 2.0 cannot associate two actions directly. This is why we use pins. Pin is an abstract class, so we use OutputPin or InputPin. In this case they are shown as black little squares to express that it is a streaming port. This means that the isStream attribute is set to True and

consequently, the isException attribute is set to False Fig 10.

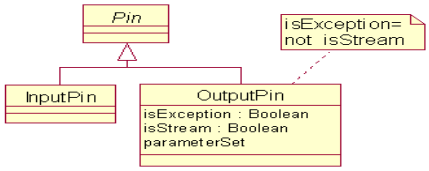


Fig.10 Pins hierarchy

In [3] page 352 we can read “Parameters are extended in complete activities to add support for streaming, exceptions, and parameter sets”. A specific ObjectNode called ActivityParameterNode is defined to use this parameter (Fig 11) so that an ObjectNode can support streaming, exceptions and parameter sets. We can also read in page 355 that OutputPins are used with an annotation text {steam} to show streaming (or the little black square Fig 9) and has attributes that can express exceptions and parameter sets. So there are two ways to express streaming for object nodes. We use only input and output pins, because the use of an ActivityParameterNode will result in too many nodes in the activity diagram, the use of pins takes less place in the diagram an shows the existence of energetic ports.

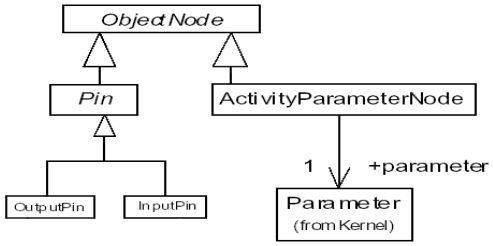


Fig.11 ActivityParameterNode use of the Parameter construct

4.2.4 Constraints on the defined elements

For each element a list of constraints need to be defined so that we can express correct bond graphs with this extension. As an example we present here the constraints that come from the causality concept used in bond graphs:

-Junction 0 : Only one causality sign close to the junction. (sum(causality=end of entering flows),(causality=start for leaving flows))=1.

-Junction 1 : Only one EnergeticFlow whitout a causality signs next to the junction (sum(causality=end of entering flows),(causality=start for leaving flows))=number of flows-1.

-De,Df : No causality.

-EnergeticFlow from an Se element: causality=end.

-EnergeticFlow from an Sf element: causality=start.

-Tf element: Both EnergeticFlows with causality=start Or both EnergeticFlows with causality=End.

-Gy element: One EnergeticFlow with causality=end, the other causality=start.

4.2.5 Block diagram extension

In order to use this BG representation, we need to add the elements that correspond to the Block Diagram elements. In fact, block diagram is used conjointly with BGs to depict the control part of the system. These generic elements are used Fig.12, the transmittance element (transfer functions), the operator element (comparator, additioner) and the Setpoint element. We also add input pins for the BlockDgOperator to determine the kind of operation the input will undergo. The *Pin* class inherits from MultiplicityElement then InputPin accepts multiple entries. We keep the same informational flow described previously.

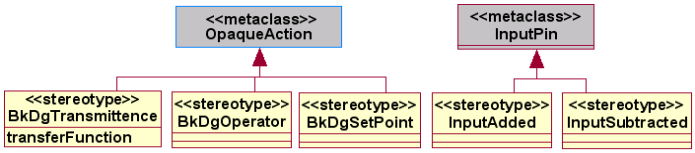


Fig.12 Block diagram extensions.

4.2.6 Extension package

These extensions are delivered into one package that will be added when using Bond Graphs in a SysML project Fig.13.

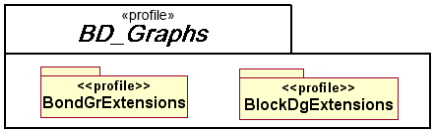


Fig.13 Deliverable package for Bond Graphs support.

5 Formalism application

In this section we are showing an example of usage of our SysML profile. We describe this servo system (Fig 14) with causal bond graphs then with the SysML activity bond graphs.

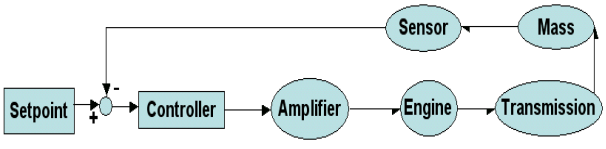


Fig.14 Combined word-bond graph of a servo system.

The resulting causal bond graph representation is shown in the following Fig 15.

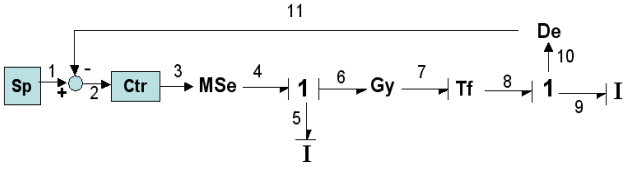


Fig.15 Associated Bond Graph.

This causal bond graph will be represented in SysML by Fig 16. The flow number 4 is described as an object flow stereotyped Bgraph_EnergeticFlow. Its causality is set to the end position and its energetic domain is set to electricity which is one of the values of energeticDomainType enumeration.

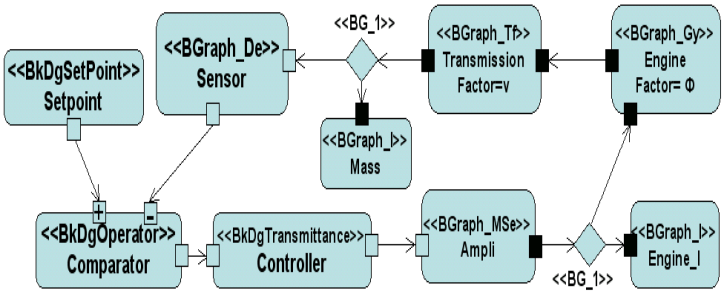


Fig.16 Activity representation of the system

We depict in the following diagram Fig 17 the specification of two bonds. Bond number 1 and bond number 10. The first one is an energeticFlow and the second is an informationalFlow.

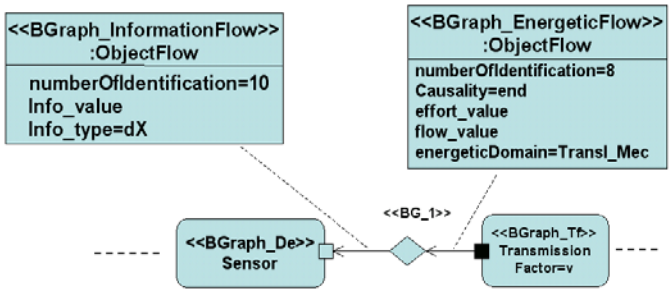


Fig.17 Specification of diagram elements

6 Conclusion

SysML is intended to support the specification, analysis, design, verification and validation phases. Adding Bond Graphs to SysML will help with the spreading of SysML inside the mechtronics community. This extension will be a powerful argument to convince systems engineers to use this language that is presented as the future de facto standard.

The activity diagrams are a good formalism to express Bond Graphs. The extensions that are necessary for activities to comply with Bond Graphs are not heavy.

We only used classical extension mechanisms such as stereotypes, tagged values and constraints to define this activities extension. This is why SysML users can easily integrate Bond Graphs to their design environment be it Rational Rose, Objecteering, Poseidon or any other tool that supports usual extension mechanisms which is considered as a basic feature of CASE tools (Computer-Aided Software Engineering Environments).

The Bond Graphs/Block diagrams composition can combine the command part with the physical part of the system.

7 Perspectives

One of the advantages of bond graphs is that it can be used to extract the system equations to be simulated. In a future work we could generate these equations from our activity diagram and describe them using the parametric diagram included in SysML.

We can also use other simulation tools. One possibility is the generation of Modelica code [7]. Modelica is an object-oriented modelling language with a textual definition to describe physical systems.

Finally , we need to continuously keep this proposal up to date as both UML 2.0 and SysML specifications are still evolving.

References:

[1] SysML specification V0.9. 10 january 2005.

[2] UML for systems engineering RFP, OMG Document: ad/03-03-41.

[3] UML 2.0 Superstructure FTF convenience document: ptc/04-10-02. October 8, 2004.

[4] UML 2.0 infrastructure specification, OMG Adopted Specification ptc/03-09-15.

[5] G.Gandanegara, « Méthodologie de conception systémique en Génie Electrique à l'aide de l'outil Bond Graph Application à une chaîne de traction ferroviaire », PhD report for the « Institut National Polytechnique de Toulouse», page 5, 2003.

[6] J. Zaytoon, *Systèmes dynamiques Hybrides*, Hermes Science Europe Ltd, page 94, 2001.

[7] J.F. Broenink,, "Object-oriented modelling with bond graphs and modelica". *International conference on bond graph modelling ICBGM'99, Simulation series Vol 31 no 1,SCS, page 163-168.*

[8] I. Bailey, F. Dandashi, H. Ang, D. Hardy, "Using Systems Engineering Standards In an Architecture Framework", white paper eurostep company.

[9] Systems Modeling Language (SysML) Specification Addendum to SysML v. 0.9 "*Profiles and Model Libraries* Chapter", 30 Mai 2005.

[10] M.K. Hales, R C. Rosenberg, Structured modelling of mechatronic components using multiport templates, Proceedings ASME IMECE 2000 DSC-Vol.69-2, page 787-794.

[11] W.Borutzky, Relations between graph based and object-oriented physical systems modelling, ICBGM'99 International Conference on Bond Graph Modelling and Simulation, San Fransisco, CA, Jan. 17-20, 1999, pp.11-17.

[12] M.Paludetto, J.Delatour, A.Benzina, UML2, Vers une formalisation des besoins des systèmes embarqués, *RSTI – TSI*. Vol 23 – n° 4, 2004, pages 543 to 567.