An Ontological Driven Approach of HAD Specific Language Designing

Marcellin Nkenlifack, Serge Mboubé - Etouké

An Ontological Driven Approach of HAD Specific Language Designing

Marcellin Nkenlifack† et Mboubé-Etouké Serge‡

† Département de Génie Informatique

Institut Universitaire de Technologies FOTSO Victor de Bandjoun
Université de Dschang
PO BOX 134 BANDJOUN
CAMEROUN

marcellin.nkenlifack@gmail.com

‡ Département de Mathématiques et Informatique
Université de Dschang
PO BOX 69 DSCHANG

CAMEROUN

mboube.etouke.serge@gmail.com

RÉSUMÉ: La plupart de langages qui ont été développés pour concevoir les systèmes automatiques n'offrent qu'une compréhension graphique (pour l'utilisateur), au lieu d'une compréhension machine (par un ordinateur) en même temps qu'une capacité à traiter les modèles desdits systèmes. Nous nous intéressons ici à la proposition d'une approche de conception d'un langage dédié aux systèmes automatiques, plus précisément les systèmes dynamiques hybrides à travers la modélisation basée sur le **Diagramme d'Activité Hybride HAD**. L'approche développée tire avantage de la description de l'ontologie du domaine au moyen d'un réseau sémantique afin de révéler les faits, concepts, relations ou liens d'association, comportements et sémantiques implicites et/ou explicites qui gouvernent la connaissance des systèmes dynamiques hybrides via **HAD**.

ABSTRACT. Most of the languages that have been developed to design control system have only a graphical understanding than machine understanding and processing ability. In this paper, we address and introduce the issue of designing a domain specific programming language in the field of control system, more especially, hybrids dynamics systems designed through **HAD modeling**. The approach presented takes advantage of semantic networks ontological description that reveals the implicit facts, concepts, relationships, behaviors and semantics which govern HAD knowledge to design a sound domain specific language which is more convenient when writing a control system code. The facts, association's links and behaviors that implicitly characterize an automatism model through HAD modeling, forming therefore an influence network among entities of the model, are regrouped in concepts, relationships and semantics which takes all ist meaning in a semantic network.

Edition Spéciale CRI'2015 – 2015, pages 1 à 21 – **A R I M A**

MOTS-CLÉS : Systèmes Dynamiques Hybrides, Diagramme d'Activité Hybride (HAD), Langages dédiés, Réseaux Sémantiques, Xtext, Java, Eclipse.

KEYWORDS: Automatics Hybrids Dynamics Systems, Hybrid Activity Diagram (HAD), Domain Specific Languages (DSL), Semantic Networks, Xtext, Java, Eclipse.

1. Introduction

An automatic system is one that can run itself (both in ist control and process) without human intervention. Several approaches of description of this type of system have been developed [1]: the Discrete Systems (DS), the Continuous Systems (CS) and the Hybrid Dynamic Systems (HDS) that regroup both the first two. Like is done in software engineering, the development of those systems start within a stage that makes a model of the future system. In automation, languages that are used to design the model depend on which approach is used. The Discrete Systems Approach most prefers languages of state's machines in which we find StateCharts, Grafcets and Gemma [2]. The Continuous Systems Approach prefers Differentials Equations [6], and Hybrid Activity Diagram (HAD), introduced in [3], recently developed in LAIA - FV UIT – University of Dschang, Cameroon, has proved his height ability in the designing of Hybrid Dynamics Systems.

Even if are StateCharts, Grafcets or Gemma in Discrete Systems Approach, Differentials Equations in Continuous Systems Approach or, most recently HAD in Hybrids Dynamics Systems Approach, all those languages despite they gives a thoroughly understanding of compositions and behaviors of the system under design through his model, they only give a graphical meaning of the functionality of the system by or for an engineer or designer, not a machine understanding of the system model that can be eventually given in text code, in a specific language handling concepts and semantics of the domain. This is the issue that has attracted our attention.

In the following, we will focus our attention on the Hybrids Dynamics Systems Approach of description of systems, and we will talk about those systems through a model designed via HAD modeling. Given that automation in general, and Hybrids Dynamics Systems in particular are specific fields, we propose to design a domain specific language that will capture facts, concepts, behaviors and semantics that lives in automation viewed through HAD approach. To reach our issue, we adopt an intuitive approach starting by a domain analysis which falls on the ontology of the domain represented in semantic networks. From there, we initiate an approach that derives our ontology in to a production rules set that, with some other arrangements will constitute the grammar of the proposed language.

So, we are not dealing about the concrete semantic meaning in semantic networks based knowledge representation language, but we are taking advantage of his cognitive

plausibility and expressivity [4] to reveal all types of facts, concepts, relationships and semantics that implicitly live in Hybrids Dynamics Systems viewed through HAD modeling. The sound semantic network representing HAD knowledge domain consist of our ontology that, like [5] stated, drives our approach in a characterization of a specific programming language that has only specificities on Hybrids Dynamic Systems viewed through HAD.

This article is presented as follows. Section 2 provides overviews on Hybrids Dynamics Controls Systems and HAD modeling approach, ist mains concepts and semantics. Section 3 gives the semantic network that we have designed to reveal the ontology of HAD in the purpose of designing a domain specific language. Section 4 introduces our driven policy from semantic network based HAD-ontology to a HAD programming language, the ANTLR-based grammar, Xtext-based IDE of HAD Programming Language that we have designed and the Rolling Mill code in example of use. Finally, section 5 presents the conclusion and perspectives.

2. Hybrids Dynamics Control Systems and HAD

2.1. Hybrids Dynamics Control Systems

Researches on industrials automatics systems in general, and on Hybrids Dynamics Controls Systems in particular have taken initiatives to solve some essentials issues of the domain [1]:

- Designing, which consist of having a systemic approach structuring all different objects of the system in accordance with the physical meaning of the causality of their interactions.
- Analysis, that includes the development of a set of verification and validation tools of Hybrids Dynamics Systems then, a mastering of the complexity of this analysis and the physical interpretation of some properties to examine some properties like the system global stability through all is running stages.
- Simulation, in which actual researches concerns formal methods and tools relating to Hybrids Dynamics Systems behaviors analysis, and the synthesis of control principles which are still in their beginning [7].

Given that, simulation, above all, is still an inescapable path when is necessary to help design an installation, validate some control system designed for the installation or validate the model proposed.

Hybrids Dynamics Systems are those in which coexists a discrete sub-system interacting with a continuous sub-system:

The global state of the system can be described via a combination of continuous variables, discrete variables, or symbolic ones (likes "open", "close", "defective").

Variables used to define the time can be in continuous type (in differentials-algebra equations), discrete type (sampling of the signal describing variable evolution, each sample having its own date), and symbolic type (in this case, different events are not still joint to some determined instance and can never be used like dates).

The process can also be continuous and factual. This is the case of installations of continuous productions with final stages of discontinuous packaging.

The particularity of those types (hybrids ones) of systems is their interactions. For example, the sequential-continuous interactions can be materialized at actions level (stage of the Grafcet). We then talk of action-interaction. The continuous-sequential interactions are found at receptivity levels associated to transitions. **Figure 1** gives a glance of a minimal hybrid Grafcet [3].

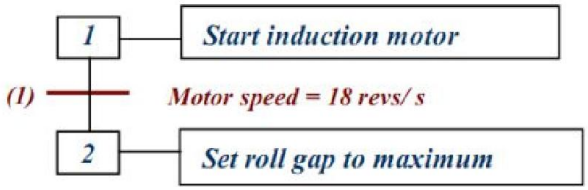


Figure 1. ***Example of a mini Hybrid Grafcet***

Observing the above minimal Grafcet (Fig.1), we can notice that:

- The action 1 : "Start induction motor" starts an interaction from the stage 1 of the Grafcet and it is applied on the induction motor (a continuous sub-system). It is an action-interaction example of sequential-continuous type.
- The receptivity (1) : "Motor speed = 18 rev/s" calls an interaction coming from the induction motor and acts on the following of the Grafcet. It is a receptivity-interaction, of continuous-sequential type.

One minimal designing of Hybrids Dynamics Systems can there be represented as **Figure 2** shows [1]

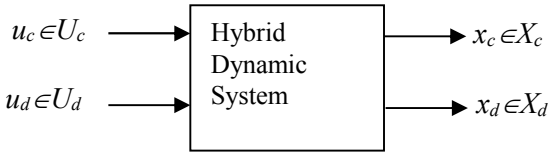


Figure 2. ***Hybrids Dynamics Systems Designing pattern.***

- The state follows in X=Xc×Xd, where Xc is included or equals of Rn and Xd is included or equals of N.

- The inputs of the system are functions of controls U=Uc×Ud.
- The hybrid system can thus be structured under two followings parts:
  - One continuous dynamic sub-system Sc that its evolution is described through a transition continuous function φc that depends of the value of xd : xc (t) = φc (t, to, xc(to), xd, uc) ;
  - One factual discrete sub-system Sd that its evolution is described through a transition discrete function φd : xd (t+) = φd (t, xc, xc(t), ud) ;
  - A set of links among the two sub-systems.

2.2. HAD modeling approach

Work done in [3] introduces Hybrid Activity Diagram HAD, a modeling approach that gives a solution of hybrids dynamics systems object-oriented designing. HAD takes in to account causes to effects relationships among entities, and has this advantage to be compatible with both the languages of industrials systems specifications (Grafcet, MSMC) and classical UML diagrams [8][9]. The foundations of HAD are built on activity diagram model of UML, causes to effects physics behaviors, parallelism structure and influences network among entities.

The UML activity diagram model shows correctly the global sequential organization of activities of several objects in several uses cases. Also, an activity diagram like Grafcet reveals the parallelism structure of the system through some pseudo-states of type convergence and divergence. So, activities diagrams models are close to Grafcets, and in more broad view, to industrials automatisms specifications tools. Thus, and like [3] [10] stated, activity diagram model is more convenient for designing multithreading applications.

Causality is a fundamental notion that helps in the handling of physical system because it allows understanding how a system reaches a given state from the study of interactions among variables [11]. It's a notion that is tightly related to running conditions of the system. Once that these conditions are well defined, interactions express causes to effects relationships among variables of the system, illustrating the mechanism through which they influences each others. This mechanism then built some influences network among entities of the system. Causality concept can be handled following two mains approaches, bond-graph and temporal where more details can be found at [11] [12].

2.2.1. "Activity Class" Concept

The *Class Activity* that [3] introduce incorporate a causes to effects organ or component. Influences that it's under or captures are causes, when influences that it exerts are affects. An *Activity Class* is characterized by an internal mechanism that, for a right given combination of causes, produces a determined effect. It's logical unit having three fundamentals characteristics:

- Type of influence that it exert,
- Nature of his behavior,
- Running mechanism.

Figure 3 gives an illustration of his structure [3].

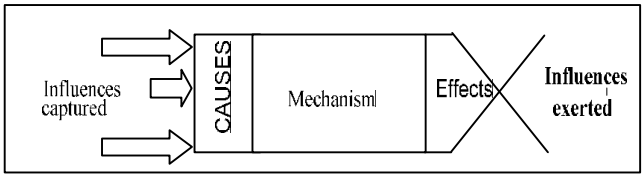


Figure 3. **General's characteristics of an *ActivityClass*.**

ActivityClass notion is light comparable at MSMC language *phenomenon* concept that descriptions are well detailed in [7].

2.2.2. HAD "Activity Module" Concept.

An *ActivityModule* represent an influence module. It's constituted of some set of internals activity classes. Entities that don't belong to the application, but influences it from outside are calls *ActivityCause*. Ideally, any activity class instance can exert his influence outside of the module.

Instances of some module that exert no influences are calls *ActivityNoEffect*. **Figure 4** following presents some components of *ActivityModule*.

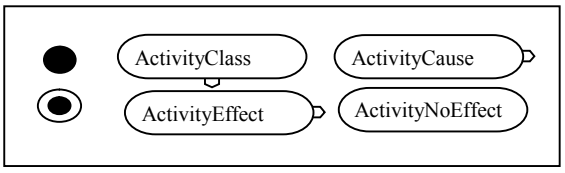


Figure 4. ***ActivityModule* components.**

2.2.3. Conditional's behaviors and Parallelim handling.

In UML classical diagrams, "*connections*" are pseudo-states having one input transition and several watched output transitions. Only one of these output transitions can be taken. A "*fusion*" marks the end of a conditional behavior initiated by a connection [13->8]. Parallelism is described by "*disconnections*" and "*junctions*". Work [3] authors have proposed to represent "*connections*", "*fusion*", "*disconnections*", and "*junctions*" through particulars objects that they have called "*ActivitySlectON*", "*ActivitySeectOFF*", "*ActivityThreadON*", and "*ActivityThreadOFF*".

2.2.4. Running bloc of HAD model.

The schema presented by **Figure 5** gives an illustration and a well understanding of the input/output dynamic of HAD modeling and ist functional or running decomposition [3]:

- *y1 = f(x1, x2, x3, ...);*
- *y2=g(x1, x2, x3, ...);*
- *y3=h(x1, x2, x3, ...) ;*
- *etc.*

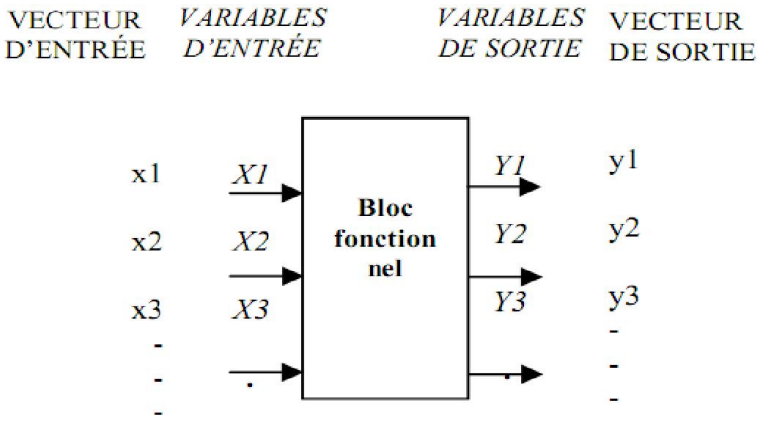


Figure 5. **Input/output HAD running bloc pattern.**

Because of the summary character of this part, we refer readers to numerous articles and books that have been written, some illustrating HAD performances, and others HAD improvement, giving all details on HAD metagraph, compatibilities with Grafcet, UML, applications putting HAD in use and others [3] [13] [14] [15] [16] [17].

3. Semantic Network Based HAD – Ontology

Even if formal languages, like HAD, have been developed to handle functionalities of hybrids dynamics systems in designing, most of those languages only have a graphical understanding of the model proposed than a machine understanding. To handle this issue, we take advantage of Domain Specific Languages discipline which stated that it has the advantage of representing, or coding, several aspect of the system using a language that is not only close to the domain in study, but tightly using concepts and semantics of the domain of interest [18]. Because the language will be specific for some specific domain, it's necessary to start by an analysis of the domain of interest, here, HAD. The results of this analysis constitute our HAD-Ontology. It's necessary to start from here like [5] stated on one hand, and because the specific language must capture all concepts, relationships and semantics implicitly or explicitly living in the domain.

Ontology is a set of knowledge terms, including the vocabulary, the semantic interconnections, and some simple rules of inference and logic for some particular topic [19], and it can also be defined as an explicit representation of a shared understanding of the important concepts in some domain of interest [20][23].

After studies carried out by [3] [13] [14] [15] [16] [17] and others on hybrids dynamics systems domain through HAD, we have observed that any hybrid dynamic system HAD model unveils five distinct entity families or modules: *InputActivityModule* representing the input of the model, *HADCommandSystem* the control, *OutputActivityModule* the output of the model, *MainActivityModule* the set of specials mechanisms of the system in designing, and *NoEffectActivityModule* giving information on the current running. Those modules also interact through some influencial networks: *HADCommandSystem* captures influences from *InputActivityModule*, runs *MainActivityModule*, exerts influences to *OutputActivityModule*, and provides information through *NoEffectActivityModule*.

To have a right understanding of this, looks **Figure 6** following. All of those modules and influences running work like a semantic network.

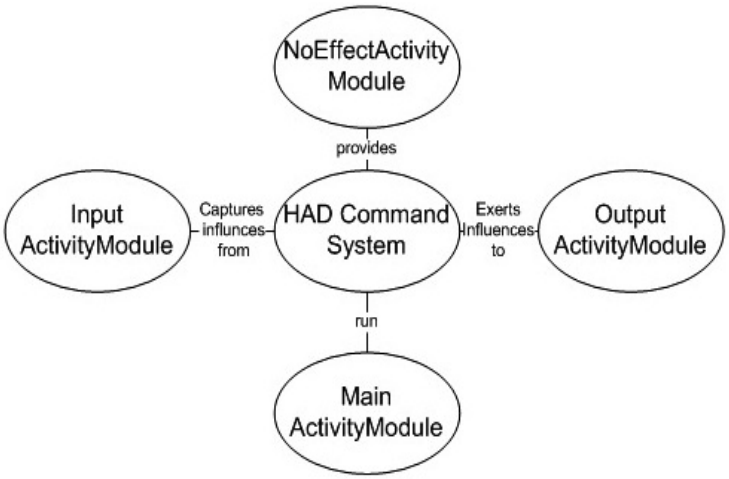


Figure 6**. Semantic Influences Network of HAD Domain.**

All facts, association's links and behaviors of entities in the domain are respectively regrouped in Module concept (***HADCommandSystem***, ***InputActivityModule***, ***MainActivityModule***, ***OutputActivityModule***, and ***NoEffectActivityModule***), influence relationships (***Capturing***, ***running***, ***exerting***, and ***information providing***) and a special semantic according to:

- if $h \in HADCommandSystem \rightarrow \exists (i \in InputActivityModule)$ and $(m \in MainActivityModule)$ / $capturesInfluencesFrom(h, i)$ and $runs(h, m)$;
- if $o \in OutputActivityModule \rightarrow \exists\ h \in HADCommandSystem$ / $exertsInfluenceTo(h, o)$;
- if $n \in NoEffectActivityModule \rightarrow \exists\ h \in HADCommandSystem$ / $providesInformationVia(h, n)$.

The above element gives an effective comprehension of the domain. With this semantic network, we are really taking advantage of ist expressivity, and his cognitive plausibility like [4] observed. One can therefore fairly understands what is about reading the influences network drawn as a semantic network. Also, they well understand the organization in which all HAD model of some hybrid dynamic system is structured. Here, we are taking advantage of modularity that allows many engineers to work in the same model of the system, but each of them focusing his attention on a particular module.

The central idea in HAD domain is the *Activity* notion [3]. Thus, in the domain, all is an *Activity* or a kind one that has been specialized and specified to the module in which it takes all is meaning. This means that, each *ActivityModule* regroups a set of *Activity* according to the semantic of that module.

3.1. In the InputActivityModule

The only special and specific kind of Activity is ActivityCause which is a unit that only flowing some type of influences, depending on the design (it can be a signal, a message, …). It's the basic unit of causes influences of the entire model in designing. An ActivityCause is represented as **Figure 7** following.

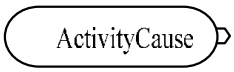


Figure 7. **An *ActivityCause* representation.**

3.2. In the OutputActivityModule

The only kind of Activity is ActivityEffect. It's the basic unit of output influences of the entire model in designing. This same module can somtimes play as a transition one among two HAD models in designing. "Fig. 8" that follows give a representation.

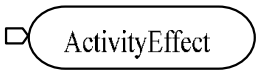


Figure 8. **An *ActivityEffect* representation**.

3.3. In the NoEffectActivityModule

We only have ActivityNoEffect where mechanism don't have any influences on the system running, but allows information provided by control. "Fig. 9" gives a representation.

Fig. 9. An *ActivityNoEffect* representation.

3.4. In the MainActivityModule

We have several kind of Activity among those who are complex, and those who are more light and simple. The more simple one is IActivity which captures influences, simply runs his internal mechanism after it has verified some conditions, and produces one influence. It can be represented like **Figure 10** shows.

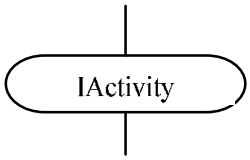


Figure 10. **An IActivity representation**

The other kind of *Activity* in this module are very complex and corresponds to *connections*, *disconnections*, *junctions*, and *fusions* situations in the model [3]:

- *ActivityThreadsONs* which opens or initiates a multi-thread situation from one influence that it has captured, and *ActivitySelectONs* which opens or initiates a selection situation. They can have as many outputs as possible depending of the situation.
- *ActivityThreadsOFFs* which closes or ends a multi-threading situation and *ActivitySelectOFFs* which closes or ends a selection situation. They can also have as many inputs as possible depending of the situation.

Those concepts respect some logic when designing a model of hybrid dynamic system:

- if $i \in InputActivityModule \rightarrow \exists\, i_j, ActivityCause(i_j)$ / $i = U_{j=1\,...m}\{i_j\}$;
- if $o \in OutputActivityModule \rightarrow \exists\, o_j, ActivityEffect(o_j)$ / $o = U_{j=1\,...m}\{o_j\}$;
- if $n \in NoEffectActivityModule \rightarrow \exists\, n_j, ActivityNoEffect(n_j)$ / $n = U_{=1\,...m}\{n_j\}$;
- if $ActivityThreadsONs(a) \rightarrow \exists\, x_{j,\ j=1,..m}, IActivity(x_j)$ / $influenceCause(x_1, a)$ & $influenceCause(a, x_{j,\ j=2,..m})$ & $run(x_j)$;
- if $ActivitySelectONs(a) \rightarrow \exists\ \ x_{j,\ j=1,...m}, IActivity(x_j)$ / $influenceCause(x_1, a)$ & $influenceCause(a, x_{j,\ j=2,..m})$ & $!run(x_s)$, s the selection;
- if $ActivityThreadsOFFs(a) \rightarrow \exists\, x_{j,\ j=1,..m}, IActivity(x_j)$ / $influenceCause(x_{j,\ j=1,..m-1}, a)$ & $influenceCause(a, x_m)$;
- if $ActivitySelectOFFs(a) \rightarrow \exists\, x_{j,\ j=1,..m}, IActivity(x_j)$ / $influenceCause(x_{j,\ j=1,..m-1}, a)$ & $influenceCause(a, x_s)$ & $!run(x_s)$, s the selection;

The other concepts in the field are those we qualify of more complex and corresponds to situations in which we leaves from threads to thread, selections to threads, and threads to selections. There are called: *ActivityThreads2Thread*, *ActivitySlects2Thread*, and *ActivityThreads2Select*. Those concepts are governed by the following logic:

- if *ActivityThreads2Thread*(a) → ∃ $x_{j,\ j=1,..m}$, *IActivity*(x_j) / *influenceCause*($x_{j,\ j=1,..x}$, a) & *influenceCause*(a, $x_{x+1,...,m}$);
- if *ActivityThreads2Select*(a) → ∃ $x_{j,\ j=1,..m}$, *IActivity*(x_j) / *influenceCause*($x_{j,\ j=1,..x}$, a) & *influenceCause*(a, $x_{x+1,...,m}$) & !run(x_s), s ∈ [$x+1$; m] the selection;
- if *ActivitySelects2Thread*(a) → ∃ $x_{j,\ j=1,..m}$, *IActivity*(x_j) / *influenceCause*($x_{j,\ j=1,..x}$, a) & *influenceCause*(a, $x_{x+1,...,m}$) & !run(x_s), s ∈ [1; x] the selection;

  $i, j, m \in \mathbb{N}$.

Because we are not dealing with the graphical aspect of the language here, we avoid graphics for those complex and more complex concepts. But, they can be found in works [3] [13] [14] [15] [16] [17]. Anyway, **Figure 11** gives an example of a graphical illustration for deep view.

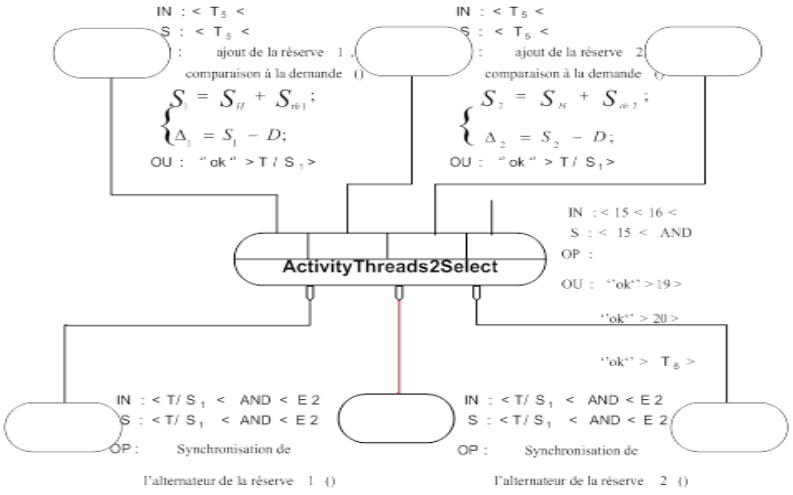


Figure 11. **An excerpt of a HAD model in use for an example respecting some above logic.**

In **HADCommandSystem** : the most interest Activity we have are the Begin and the End Activity, simply for starting and stopping the system.

Even that all those kind of *Activity* are specific of some semantic, all of them incorporates a set of basics properties and mechanism regrouped in the *Activity*. Those properties are: *Name* giving the name on the entity, *INnbre* giving the number of input influences flows, *IN* the input influences flows interfaces, *Syn* that synchronize all input flows according to the specific semantic of the entity, *Message* containing message that influence its neighbor, *OUnbre* giving the number of output influences flows, and *OU* the output influences flows interfaces.

4. Semantic Network based HAD – Ontology Driven Policy Designing HAD Specific Language

Once the domain analysis have been done and the domain knowledge have been unveiled through his semantic network based ontology, logics of reasoning, and others, we now can deal in the domain, well understanding and shared.

4.1. The Driving Approach

The approach that we are introducing here is an intuitive one regarding the structure of HAD domain knowledge. We state that, given that each module in the HAD domain knowledge has its own specific semantic, thus, each of them corresponds to a specific production rule which recognizes each instance of this module. In the same way, given that each specific concept in each HAD knowledge domain module characterizes and means a specific semantic, each of them also corresponds to a specific production rule that recognizes each instance of this concept. Thinking like this, we just derives the following:

1. HADCommandSystem → 'Begin' '{' ((InputActivityModule)+ (MainActivityModule)+ (NoEffectActivityModule)* (OutputActivityModule)*) '}' 'End' ;

2. InputActivityModule → : ID '{' (ActivityCause)+ '}' ;

3. MainActivityModule → : ID '{' (Activity)+ '}' ;

4. NoEffectActivityModule → : ID '{' (NoEffectActivity)+ '} ;

5. OutputActivityModule → : ID '{' (OutputActivity)+ '}' ;

6. ActivityCause → 'Activity' ID 'is' 'ActivityCause' ((RuleMessage) (RuleOperation)? (RuleSortie) 'EndActivity' ID ;

7. Activity → 'Activity' ID 'is' Type ((RuleEntree) (RuleSyn) (RuleMessage) (RuleOperation) (RuleSortie)) 'EndActivity' ID ;

8. OutputActivity → 'Activity' ID 'is' 'OutputActivity' ((RuleEntree) (RuleSyn) (RuleMessage) (RuleOperation) (RuleSortie)) 'EndActivity' ID ;

9. RuleEnntre → ID 'as' 'IN' ((< ID)+);

10. RuleSyn → ID 'as' 'SYN' (((RuleSyn1) | (RuleSyn2))) ;

11. RuleMessage → ID 'as' 'Message' (STRING) ;

12. RuleOperation → ID 'as' 'OP' (ID()) ;

13. RuleSortie → ID 'as' 'OU' ((ID>)+) ;

14. RuleSyn1 → ('OR' ID)+ (RuleSyn2)? ;

15. RuleSyn2 →('AND' ID)+ (RuleSyn1)? ;

16. Type → 'IActivity' | 'ActivitySelectONs' | 'ActivitySelectOFFs' | 'ActivityThreadsONs' | 'ActivityThreadsOFFs' | 'ActivityThreads2Thread' | 'ActivityThreads2Select' | 'ActivitySelects2Thread' ;

4.1. The ANTLR-based HAD grammar

For more convenient development, we have taken advantage of effective tools that can be helpful having a cool *LR*(*k*) recognizer. ANTLR is a favorite one on the field of language engineering [21]. The followings table gives the ANTLR-Java implementation of HAD grammar.

TABLE I. **ANTLR-Java (Xtext) HAD Grammar implementation.**

```
grammar org.xtext.example.mydsl.HADsl with
org.eclipse.xtext.common.Terminals
generate hADsl "http://www.xtext.org/example/mydsl/HADsl"
Programme:(pream=Preambul)
;
```

```
Preambul: 'HADProgram' nameprgrm=ID '{' core=Core '}';
Core: 'Begin' '{' ((moduleIn+=InputActivityModule)+
(moduleMain+=MainActivityModule)+
(moduleSansEffet+=NoEffectActivityModule)*
(moduleOut+=OutputActivityModule)?)* '}' 'End' (fin=ID)?;

InputActivityModule:            'InputActivityModule' ':'
nameModuleIn=ID '{' (nameAcs+=ActivityCause)+ '}'  ;

ActivityCause:            'Activity' nameAc1=ID 'is'
'ActivityCause'  '(' (rulemsg=RuleMessage)
(ruleop=RuleOperation)?  (rulesortie=RuleSortie)?  ')'
'EndActivity'  nameAc2=ID  ';' ;

RuleMessage:       nameM=ID 'as' 'Message'  '(' message=STRING
')'  ';';
RuleSortie:  nameS=ID 'as' 'OU'  '(' ( sorties+=ID  '>' )+ ')'
;

RuleOperation:     nameO=ID  'as'  'OP'  '('  operation=ID
'()'  ')'  ';';
MainActivityModule:             'MainActivityModule'  ':'
nameModuleMain=ID  '{'  (iactivities+=IActivity)+  '}';

IActivity:   'Activity' nameIAc1=ID 'is'  nameType=Type  '('
(ruleentree=RuleEntree)  (ruleSyn=RuleSyn)
(rulemsg=RuleMessage)  (ruleop+=RuleOperation)+
(rulesortie=RuleSortie) ')'  'EndActivity' nameIAc2=ID  ';';


NoEffectActivityModule:         'NoEffectActivityModule'  ':'
nameNoEffect=ID  '{'  (nameNoEffects+=NoEffectActivity)+  '}';
NoEffectActivity:  nameNoEffect1=ID 'is' 'NoEffectActivity'
'(' (rulemsg=RuleMessage) ')'  ';'  'EndActivity'
nameNoEffect2=ID  ';' ;

OutputActivityModule: 'OutputActivityModule'  ':'  nameOut=ID
'{'  (nameOuts+=OutputActivity)+  '}';
OutputActivity: nameOutAc1=ID  'OutputtActivity'  '('
(ruleentree=RuleEntree)   (ruleSyn=RuleSyn)
(rulemsg=RuleMessage)   (rulesorti=RuleSortie)?  ')'  ';'
'EndActivity'  nameOutAc2=ID  ';' ;
```

```
RuleEntree:  nameE=ID  'as' 'IN'  '(' ('<' entrees+=ID )+ ')'
';';
RuleSyn:     nameSyn=ID  'as' 'SYN'  '('((ruleSyn1=RuleSyn1) |
(ruleSyn2=RuleSyn2)) ')'  ';' ;

RuleSyn1:    ('OR' id1+=ID)+ (rulS1=RuleSyn2)?;

RuleSyn2:    ('AND' id2+=ID)+ (rulS2=RuleSyn1)?;

Type:'IActivity'|'ActivitySelectONs'|'ActivitySelectOFFs'|'Acti
vityThreadsONs'|'ActivityThreadsOFFs'|'ActivityThreads2Thread'|
'ActivityThreads2Select'|'ActivitySelects2Thread';
```

Listing 1. ***ANTLR-Xtext based Implementation of HAD Grammar***

4.2. HAD Grammar in use: Case of Rolling Mill code

4.2.1. The Rolling Mill

The *Rolling Mill* is an effective example of hybrid automatic system. The system transforms metallic blocs to steel sheets. The lamination process calls sequences of operations: the metal is heated at a precise temperature; the opening among rolls is adjusted to allow the metal to inter. The bloc is inserted in rolls; the induction motor move rolls with constant velocity until the opening is stabilized at some fairly weak value; steel sheets produced are ejected from rolls, and the sequence restart. More details can be found in [22]. The logic (sequence) of operations, defined by the Grafcet, is implemented by programmable automaton (Programmable Logic Controller), which thus constitutes sequential sub-system of the hybrid system. The other components of the system (servo-motor, opening roll gap controller, induction motor, rolls, temperature controller) constitute the continuous sub-system. They are designed by a set of algebraic and differentials equations.

4.2.2. The HAD Rolling Mill Model

HAD Rolling Mill Model that we will code with the new HAD-specific language is shown at **Figure 12**, takes from [3]:

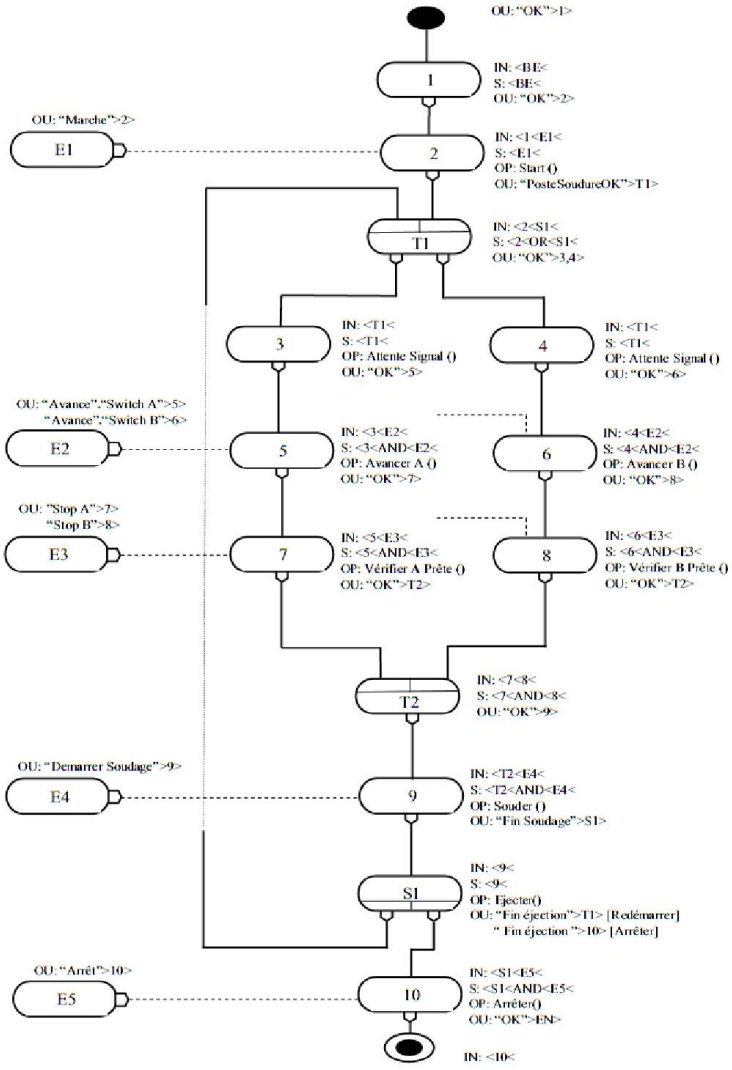


Figure 12. **HAD Rolling Mill Model.**

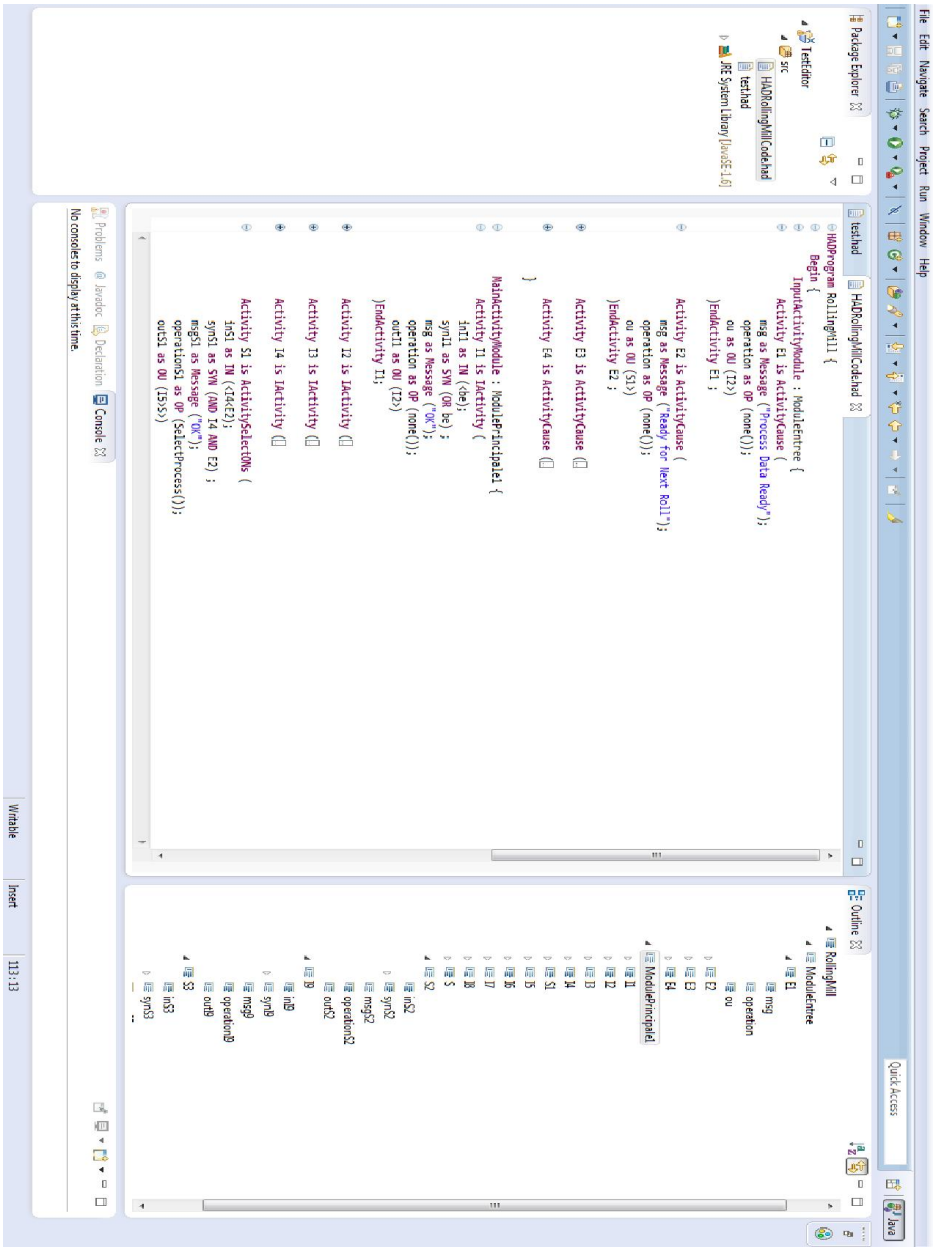
4.2.3. HAD's excerpt code of Rolling Mill

Listing 2 gives us a view of an excerpt of the HAD code of "Rolling Mill" system.

```
HADProgram RollingMill {
       Begin {
       InputActivityModule : ModuleEntree {

       Activity E1 is ActivityCause (
              msg as Message ("Process Data Ready");
              operation as OP (none());
              ou as OU (I2>)
       )EndActivity E1 ;

       Activity E2 is ActivityCause (
              msg as Message ("Ready for Next Roll");
              operation as OP (none());
              ou as OU (S1>)
       )EndActivity E2 ;

       Activity E3 is ActivityCause (
              msg as Message ("Specimen in Position");
              operation as OP (none());
              ou as OU (I7>)
       )EndActivity E3 ;

       Activity E4 is ActivityCause (
              msg as Message ("Mill Reverse");
              operation as OP (none());
              ou as OU (S>)
       )EndActivity E4 ;
       }

      MainActivityModule : ModulePrincipal1 {
      Activity I1 is IActivity (
              inI1 as IN (<be);
              synI1 as SYN (OR be) ;
              msg as Message ("OK");
              operation as OP (none());
              outI1 as OU (I2>)
       )EndActivity I1;
…}
```

Listing 2. ***An*** **excerpt code of Rolling Mill**

The next figure, **Figure 13** shows us a view of HAD Eclipse based IDE developed to handle Hybrids and Dynamics system designed via HAD modeling approach.



Figue 13. **HAD Eclipse based IDE**

5. Conclusion and Perspectives

We have addressed and introduced the issue of designing a domain specific language driven by the specific ontology of the domain under study and we have taken this occasion to propose a programming language of hybrids dynamics system viewed through HAD modeling. The approach is simple and very useful. We have demonstrated its usability when designed a semantic network HAD domain knowledge and derived it to a set of production rules that finally, has constituted the grammar of the language corresponding to the domain, HAD here. The results presented here are among works that are currently conducted in the purpose to make HAD a throughly Domain-Specific Language in both modeling and programming for handling hybrids dynamics systems simulation. Those results are among those that formalized HAD according to MOF2, developing of an Eclipse based IDE for HAD modeling and designing an MDA-based architecture that will support HAD entire framework (modeling and programming).

The work introduced here are still in progress. In the future, we will present *HADtalk* (the HAD-specific programming language corresponding to HAD modeling) in use, in a real world hybrid dynamic system, its entire simulation, its stable Eclipse-based plug-in for developing any hybrid dynamic system, and tutorials. We also consider the issue of model transformation that allows to move from a HAD model to a HAD code, and vice versa, depending of the requirements or convenience of the user.

Bibliographie

[1] J. Zaytoon, "Systèmes Dynamiques Hybrides", Traité Information - Commande - Communication Hermes, Paris, 2001.

[2] H. Brenier, Métamodèles des machines d'état, Les spécifications fonctionnelles des automatismes industriles et temps réel, Dunood Paris, 2001, pp.143-176.

[3] E. Tanyi and M. Nkenlifack, "Une Adaptation d'UML à la Modélisation des Systèmes Hybrides", in e-STA, vol. 7, n°2, 2010, pp 46-57.

[4] R. Nkambou, Modeling the Domain, "Advances in Intelligent Tutoring Systems ", R. Nkambou, J. Bourdeau and R. Mizoguchi (Eds), Sptinger-Verlag, Berlin Heidelberg, 2010, pp 15-32.

[5] D. Gasevié et al., Ontologies, "Model Driven Engineering and Ontology Development", 2nd edn, Sptinger-Verlag, Berlin Heidelberg, 2009, pp 44-80.

[6] S. Lipschutz, Mathématiques pour informaticiens: Cours et exercices, série Schaum, McGraw-Hill Inc, New York, 1983.

[7] H. Brenier, Métamodèles MSMC (Modélisation et Simulation des Machines Cybernétiques), Les spécifications fonctionnelles des automatismes industriles et temps réel, Dunod Paris, 2001, pp.194-248.

[8] OMG, UML2 Reference Manual. www.omg.org.

[9] J. Rumbaugh, I; Jacobson and G. Booch, "The Unified Modeling Language Reference Manuel", Addison-Wesley, 1999.

[10] E. Tanyi and M. Nkenlifack, "An object oriented simulation platform for hybrid control systems, Analysis and Design of Hybrid Systems (ADHS) 2003, Proc. Of the IFAC International Conference, St Malo, France, June 16-18 2003, Edited by S. Engell, H. Gueguen and J. Zaytoon, ISBN 0-08-044094-0.

[11] S. Traoré, "Une Approche Orientée Objet dans la Simulation des Systèmes Dynamiques Semi-Continus", Thèse de Doctorat de l'Institut National Polytechnique de Grenoble, Grenoble, France, 1999.

[12] J. Buisson, "Bond graphs à commutations, in Systèmes Dynamiques Hybrides", traité Systèmes automatisés, Information Commande et Communication, Hermès, Paris, 2001, pp. 93-117.

[13] M. Nkenlifack, E. Tanyi and F. Fokou, "Establishing bridges between UML, HAD and GRAFCET Metamodels for the Modeling of Dynamic Systems", International Journal of Scientific & Engineering Research, Vol. 2, Issue 3, March-2011, ISSN 2229-5518.

[14] M. Nkenlifack, E. Tanyi, and F. Fokou, "Ameliorationof the HAD Metamodel for the Modeling of Complex Hybridd Systems", International Journal of Advanced Research in Computer Science, Vol. 2, N° 1, Jan-Feb 2011, ISSN N° 0976-5697.

[15] M. Nkenlifack, E. Tanyi, and L. Domche, "Interoperability between the improved Metagraph HAD and Grafcet: Case of the power Network of Cameroon", IJRRAS 9(1) October 2011, Vol. 9, Issue 1, October 2011.

[16] M. J. A. Nkenlifack, "HAD: Une Extention du Langage UML pour la Modélisation des Systèmes Hybrides ", Du Génie Logiciel au Génie Automatique, Editions Universitaire Européennes, Dudweiler landstr. 99, 66123 Sarrebruck, Allemagne, 2004, ISBN 978-613-1-56194-8, pp. 157-172.

[17] M. J. A. Nkenlifack, "Amélioration et Normalisation du Métagraphe HAD pour l'Analyse des Systèmes Hybrides Complexes ", Du Génie Logiciel au Génie Automatique, Editions Universitaire Européennes, Dudweiler landstr. 99, 66123 Sarrebruck, Allemagne, 2004, ISBN 978-613-1-56194-8, pp. 173-217.

[18] M. Fowler and R. Parsons, "Domain – Specific Languages", Addison – Wesley Professional, September 2010, Print ISBN-10: 0-321-71294-3.

[19] J. Hendler, "Agents and semantic web", IEEE Intelligent Systems, 2001, Vol. 16, N° 2, pp. 30 – 37.

[20] Y. Kalfoglou, "Exploring ontologies", Handbook of Software Engineering and Knowledge Engineering, Vol. 1, Fundamentals, ed. S.K. Chang, World Scientific, Singapore, pp. 863-887.

[21] T. Parr, "The Definitive ANTLR Reference. Building Domain-Specific Languages", The Pragmatic Programmers, Pragmatic Bookshelf, USA, May 2007, ISBN-10: 0-9787392-5-6.

[22] E. Tanyi and D. Linkens, "A G2 based Hybrid Modeling and simulation strategy and its Application to a Rolling Mill", Control Engineering Pratice, London, 1998.

[23] T. R. Gruber, "A translation approach to portable ontology specifications", Knowledge Acquisition, June 1993, Volume 5, N° 2, pp 199–220.