Vietnam Journal of Mechanics, VAST, Vol. 33, No. 3 (2011), pp. 182 – 193

AN OBJECT-ORIENTED DESIGN METHOD TO IMPLEMENT THE MECHATRONIC SYSTEM CONTROL BY USING HYBRID AUTOMATA AND REAL-TIME UML

Ngo Van Hien, Vu Duy Quang
Hanoi University of Science and Technology

Abstract. In this paper, we present a method, which is based on hybrid automata and Real-Time Unified Modeling Language (UML) to analyze and design the control parts of mechatronic systems with input or output events and signals in order to effectively gather their structure and behaviour. We introduce step-by-step analysis and design activities of a controlled mechatronic system such as the specification of its hybrid automaton and realization hypotheses, the identification of object collaborations of this system, the identification of main control capsules, their ports and communication protocols, with their static and dynamic links. These activities are conducted by specializing the iterative life cycle of system development. Then, we indicate important hypotheses, which allow all the identified capsules of this system to make their evolutions. We apply this method to develop an Electro-Hydraulic Governor (EHG) system, which allows the frequency of an electro-hydraulic station to be stabilized.

Keywords: Mechatronics, hybrid automata, electro-hydraulics and real-Time UML.

1. INTRODUCTION

A mechatronic system consists by definition of a mechanical part that has to perform certain motions and an electronic part (in many cases an embedded computer system) that adds intelligence to the system [15]. In most cases, this system takes into account models with discrete events and continuous behaviour models; they are called Hybrid Dynamic Systems (HDS) [7], [17]. These behaviour models are distributed on different operating modes, which are associated with processes related to the interactivity with users such as the designer, supervisor, maintainer etc. In addition, controlled mechatronic systems always do not have the same behaviour because this one is associated with validity hypotheses to check at any moment; the security requirement forces to envisage events and behaviours different than the nominal behaviour. The behaviour of such systems is thus complex. In this paper, we consider a controlled mechatronic system, which is the HDS and, so can be modelled by hybrid automata.

The mechatronic design deals with the integrated and optimal design of a mechanical system and its embedded control system. To carry out from the analysis to the design in our method, the iterative development life cycle [12], [18] which has been interpreted and depicted in various ways, is chosen to be an unified process of system development. This model is specified by using object-oriented design principles, which are being largely spread and appreciated in the industrial control. In addition, we choose the Real-Time UML (Unified Modeling Language) version [1], [3] based on the capsule concept that we adapted by specializing a set of capsules in precise behaviours in order to describe precisely the intercommunication type between objects of the developed HDS.

This paper is depicted by the following sections:

Section 2 presents the hybrid automaton definition, its specification and realization hypotheses of a HDS, and introduces the main notations of Real-Time UML for describing the analysis and design model of this system;

Section 3 indicates the specialization of a cycle of the iterative life cycle for developing a HDS. In this section, we present the analysis of a HDS with the identified hybrid automaton on different object collaboration. We also present the design of this system with the global communication between the identified main capsules, evolution hypotheses of capsules;

Section 4 introduces the strategies and tools, which can be used to implement and test analysis and design models.

Finally, we apply this method to develop an Electro-Hydraulic Governor (EHG) system, which allows the frequency of an electro-hydraulic station to be stabilized.

2. REALIZATION HYPOTHESES OF A HYBRID AUTOMATON AND REAL-TIME UML FOR MODELING THE HDS

2.1. Overview of hybrid automata

A hybrid automaton [2], [6] is defined by data of $H = (Q, X, \Sigma, A, Inv, F, q_0, x_0)$ where: - Q is a set of states describing operational modes of the system, called situations, q_0 is the initial situation.

- X presents the continuous state space of the automaton, $X \subset \Re^n$, x_0 is the initial value of this space.

- Σ is a finite set of events.

- A is a set of transitions being defined by $(q, Guard, \sigma, Jump, q')$ and represented by an arc between situations, where:

+ $q \in Q, q' \in Q$.

+ $Guard$ is a subset of the state space in which the continuous state must be, so that the transition can be crossed.

+ $Jump$ represents the continuous state transformation during the change of situation; it is generally expressed by a state value function, whose result is affected like initial value of the continuous state in the new situation.

+ $\sigma \in \Sigma$ introduces the event being associated to the transition; this association does not imply to give an input or output direction to the event.

- Inv is an application, which associates a subset of the state space to each situation. It is called the invariant of the situation, in which the continuous state must remain, when the situation is q, the continuous state must verifies $x \in inv(q)$.

- F is defined for each situation; the evolution of continuous state is occurred when the situation is active; this evolution of continuous state is generally expressed by a differential equation. It will be named continuous fluid F.

2.2. Realization hypotheses

To describe a HDS with the hybrid automaton's formalism, we introduce the following constraints [7], [9]:

- Events $\sigma \in \Sigma$ are considered in term of inputs/outputs and internal/external.
- X contains input/output signals.
- The global continuous evolution F coming from an extended functional diagram, which has been defined in [7], [10].

We apply the following rules, so that the invariant and guard control can generate internal events [9]:

- If $x_q \notin inv(q)$ and $Guard(a) = True, a \in A$, then there is a generated internal event; the system changes its situation from q to q' described in the set of situations of the system, with the initial value $Jump_{q'}$ identified from the concrete continuous fluid $F_{q'}$; this evolution is realized by the application state machine.
- If $x_q \in inv(q)$ and $Guard(a) = True, a \in A$, then the system remains in its actual situation q.
- If $x_q \in inv(q)$ and $Guard(a) = False, a \in A$, then the system remains in its actual situation q.
- If $x_q \notin inv(q)$ and $Guard(a) = False, a \in A$, then there is a generated internal event; the system changes into the state q", which is called the irreversible default state.

2.3. Using the "capsules, ports, protocols" notations of Real-Time UML

A capsule [1] is represented as a class, stereotyped "capsule" [14]. Capsules have much of the same properties as classes; for example, they can have operations and attributes. However, they also have several specialized properties such as public ports, private operations, message passing for modeling their transmission relationships and behaviours.

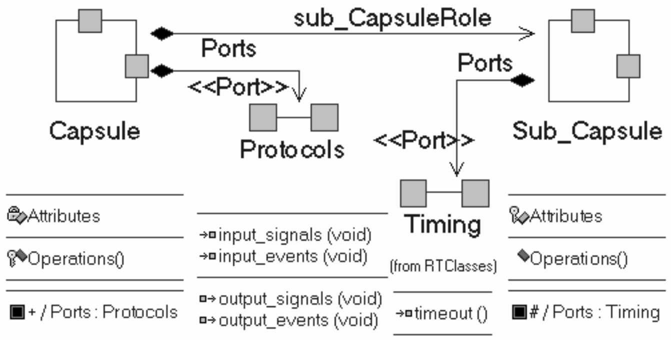


Fig. 1. Capsules, ports and protocols

The protocol is a set of messages exchanged between two objects conforms to some communication pattern called a protocol.

Ports are objects whose purpose is to send and to receive messages to and from capsule instances. They are owned by the capsule instance in the sense that they are created along with their capsule and destroyed when the capsule is destroyed.

An example of capsules, sub-capsules, ports and protocols is presented in the Fig. 1 by using the class diagram.

3. ANALYSIS AND DESIGN MODEL OF A HDS

Our approach is based on the iterative life cycle of system development (Fig. 2) which permits us to lead main development phases of system such as the analysis, design, implementation and test, and create an executable prototype [7], [18].

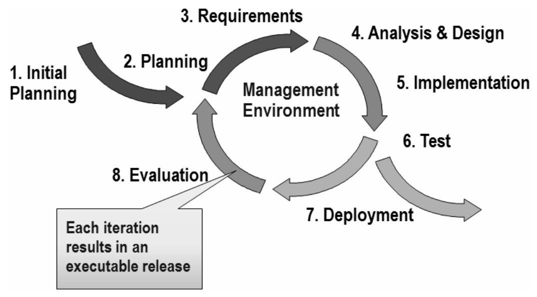


Fig. 2. Iterative development life cycle

The iterative development is a technique that is used to deliver the functionality of a system in a successive series of releases of increasing completeness. Each release is developed in a specific, fixed time period called an iteration. Each iteration is focused on defining, analyzing, designing, building, and testing a set of requirements. The earliest iterations address the greatest risks. Each iteration includes integration and testing and produces an executable release. In this paper, we concentrate on the analysis and design phase of a HDS, so that we would specialize it in the next sub-sections in order to obtain an optimal design model.

3.1. Analyzing the HDS

From the approach described in [16], we propose here 5 object collaborations: the continuous part, discrete part, Instantaneous Global Continuous Behaviour (IGCB) to develop a HDS. They are defined and used according to a virtual mechanism of components such as the object, class or class hierarchy [12].

- The discrete part's collaboration contains the set of situations Q and set of transitions A of the hybrid automaton. Each situation makes an association with a concrete IGCB.

- The continuous part's collaboration contains entity classes coming from boxes of the extended functional diagram [7] to store and process the transformational activities of the developed HDS. We can build an abstract entity class so that these entity classes can inherit it in order to simplify models by avoiding the information duplication. We will also find common attributes between entity classes to re-use them by applying heritage properties. At this moment, we limit ourselves to continuous elements which can be elements: *power amplification, inertia, delay, vibration and regulator.*

- The IGCB's collaboration consists of entity classes, which present instantaneous global continuous behaviours (continuous fluids F in the hybrid automaton of a developed HDS). Each fluid is connected to a concrete situation q_i. There is only one concrete IGCB at time given, i.e., there is only one entity class activated at one given moment in this collaboration. We can also build an abstract entity class in this collaboration so that these entity classes can inherit it.

- The internal interface's collaboration generates internal events of the developed HDS so that the discrete part's collaboration can treat these generated events. This collaboration is an intermediary between the continuous part's collaboration and discrete part's collaboration. It is used to check invariants, controls $(q, Guard, \sigma, Jump, q')$ of the hybrid automaton and generates if necessary internal events allowing the evolution.

- The external interface's collaboration is an intermediary between the developed system and intervening systems. It receives or sends episodic events, periodic signals between the developed system and intervening systems. It makes it possible to display control results, parameter settings etc. by using the MVC (Model - View - Controller) pattern [5].

The detailed structures and behaviors of these collaborations have been shown in [7].

3.2. Designing the HDS

We find that the direct transformation of object collaborations to the implementation environment must be supplemented to carry out a general control system. For example, collaborations of the control system are not well adapted to visualize, model interconnection types between the objects or sub-systems. In the design phase, we transform object collaborations identified above into capsules to carry out a HDS completely and to re-use generic capsules in different applications.

Stages to build the main capsule collaboration are the following ones:

- Identifying capsules and sub-capsules from main classes in object collaborations identified above.

- Each identified object collaboration needs at least a capsule such as the global HDS's capsule, continuous part's capsule, discrete part's capsule, internal interface's capsule, IGCB's capsule and external interface's capsule.

- Classes, which lead the exchange of messages in the object collaboration, will become capsules.

- Identifying classes in a capsule collaboration: all the entity classes become classes in the capsule collaboration.

- Identifying ports and the protocols general object collaborations with associations and messages.

- Identifying sequence diagrams and generic state machines of these capsules; these diagrams and state machines make it possible to model and carry out the interconnection between these capsules.

3.2.1. *Global interconnection structure*

We propose 5 main capsules, which take part in the realization of the hybrid automaton of a HDS: the continuous part's capsule, discrete part's capsule, internal interface's capsule, external interface's capsule and Instantaneous Global Continuous Behaviour (IGCB's capsule). The connection for communicating between main capsules, which is carried out by their ports and protocols, is introduced by the capsule structure diagram (Fig. 3).

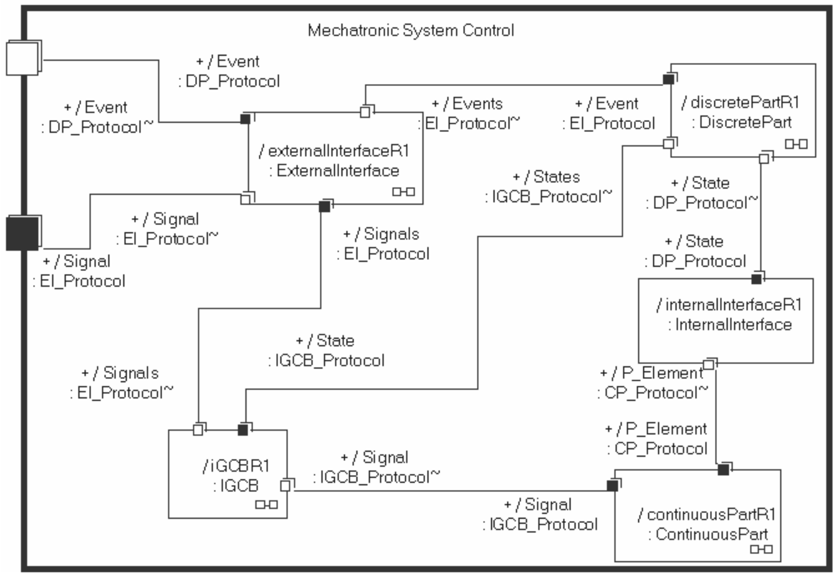

Mechatronic System Control
+ / Event : DP_Protocol
+ / Event : DP_Protocol~
/ externalInterfaceR1 : ExternalInterface
+ / Events : EI_Protocol~
+ / Event : EI_Protocol
/ discretePartR1 : DiscretePart
+ / Signal : EI_Protocol~
+ / Signal : EI_Protocol
+ / Signals : EI_Protocol
+ / States : IGCB_Protocol~
+ / State : DP_Protocol~
+ / State : DP_Protocol
/ internalInterfaceR1 : InternalInterface
+ / State : IGCB_Protocol
+ / Signals : EI_Protocol~
+ / P_Element : CP_Protocol~
+ / P_Element : CP_Protocol
/iGCBR1 : IGCB
+ / Signal : IGCB_Protocol~
+ / Signal : IGCB_Protocol
/ continuousPartR1 : ContinuousPart

Fig. 3. Capsule structure diagram of a HDS

3.2.2. *Detailing behaviours of identified main capsules*

Hypotheses that we propose for executing the identified set of main capsules of a hybrid automaton are the following ones:

- If the end of the discrete part's evolution is located before or just of the sampling date (ΔT) of the IGCB's capsule, then the current IGCB continuous model will pass to the new IGCB model corresponding to this evolution.

- If the end of the discrete part's evolution is located after of the sampling date (ΔT), then the IGCB current is not commutated.

- If an event appears during the evolution of the application state machine, then this event is memorized and treated later on.

- External and internal events have the same process by the discrete part's capsule.

- During the sampling period of the IGCB's capsule, the continue part's capsule, internal interface's capsule and discrete part's capsule make their own evolutions to possibly commutate to a new IGCB's operational mode, the IGCB continuous model remains in its current mode for this period.

- So during the period of the IGCB's capsule, the current IGCB continuous model has detected two or several states appeared, then at just end of this period, the IGCB's capsule synchronizes all these states with the null timing duration; the current IGCB continuous model passes to a new operational mode, which corresponds to the last state appeared during this period.

Let us take a simple example for explaining these hypotheses. The Fig. 4 presents a hybrid automaton example, where: F_i, is the continuous fluid i; Ee$_1$, Ee$_2$, Ee$_3$...., are external events.

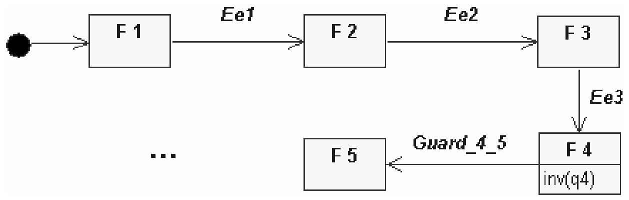


Fig. 4. A hybrid automaton example

We transform this hybrid automaton into another hybrid automaton (Fig. 5), which contains the internal event generated in the internal interface's capsule and will be the application state machine.

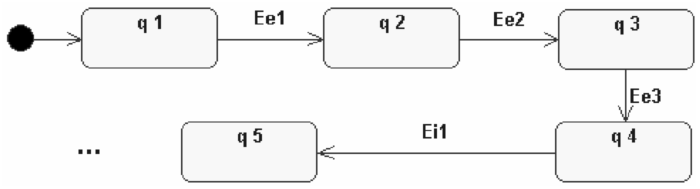
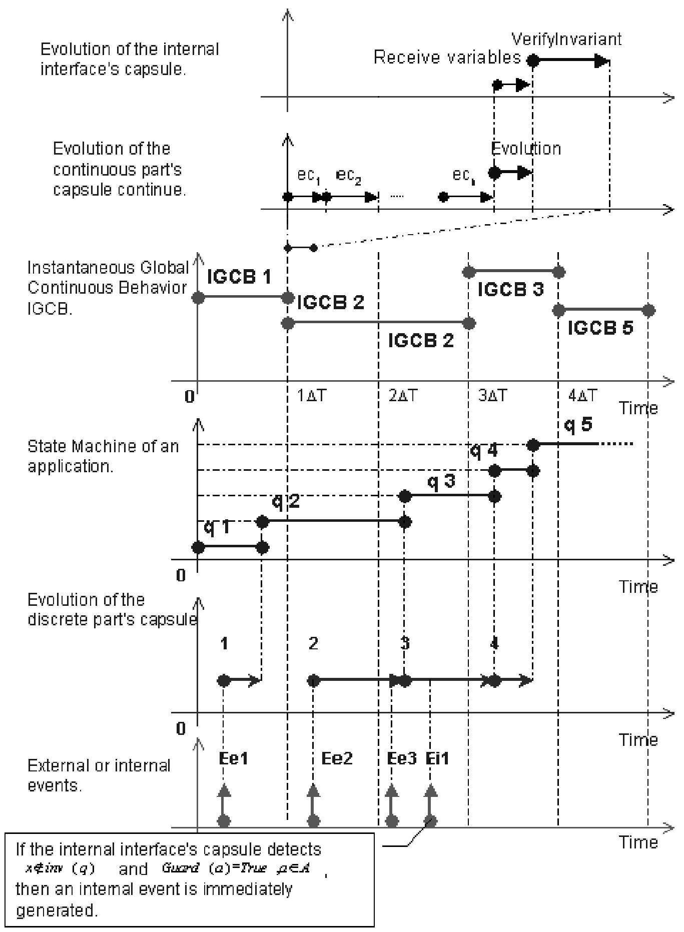


Fig. 5. Transformed hybrid automaton with the internal event Ee$_\text{i}$

When $x_q \notin inv(q_4)$ and $Guard\_4\_5 = True$; where: q_i, is the situation i; this evolution is realized by the application state machine. The concurrent timing diagram introduces evolutions of this hybrid automaton (Fig. 6) where:

- Ee$_1$, Ee$_2$, Ee$_3$...., are external events.
- Ei$_1$, Ei$_2$...., are internal events.
- q_1, q_2, ..., introduce situations of the hybrid automaton.
- ec$_1$, ec$_2$, ..., ec$_n$, present evolutions of continuous elements in the continuous part's capsule.
- ΔT, is a sampling period of the IGCB's capsule.

To explain in detail the dynamic behaviours of main capsules, we use the example given above. The Fig. 7 presents the evolution in one concrete period $(2\Delta T - 3\Delta T)$ between main capsules of the hybrid automat given above. In this figure, all the messages, which exchange between the main capsules, are synchronous; the interval between two adjacent *timeout* messages indicates the sampling period of the IGCB's capsule. The external


Evolution of the internal interface's capsule.
Receive variables
VerifyInvariant
Evolution of the continuous part's capsule continue.
Evolution
ec_1 ec_2 ..... ec_i
Instantaneous Global Continuous Behavior IGCB.
IGCB 1
IGCB 2
IGCB 2
IGCB 3
IGCB 5
0
1ΔT
2ΔT
3ΔT
4ΔT
Time
State Machine of an application.
q 1
q 2
q 3
q 4
q 5
0
Time
Evolution of the discrete part's capsule
1
2
3
4
0
Time
External or internal events.
Ee1
Ee2
Ee3
Ei1
Time

If the internal interface's capsule detects $x \notin inv\ (q)$ and $Guard\ (a)=True\ a \in A$, then an internal event is immediately generated.

Fig. 6. Concurrent timing diagram of the global evolution of main capsules

interface's capsule receives period signals coming from external continuous components. It gives the *"ContinuousElement"* message to the IGCB's capsule so that the IGCB's capsule can call all continuous elements, which correspond to the concrete IGCB: $IGCB_2$. During the call of the IGCB's capsule, the external interface's capsule receives an event Ee_3 coming from actors [3] of the developed system, and gives this event to the discrete part's capsule. Then, the discrete part's capsule memorizes and will treat this event. If the IGCB's capsule receives the *"LastContinuousElement"* message coming from the continuous part's capsule, then it gives the *"ContinuousEvolution"* message to the continuous part's capsule so that the internal interface's capsule can receive all updated variables.

The internal interface's capsule verifies the invariant of the situation q_3; in this case, there is a generated internal event. Then, the internal interface's capsule gives this event to the discrete part's capsule. The discrete part's capsule memorizes and will treat this event. In this period, the event Ee_2 has been treated by the discrete part's capsule. The IGCB's capsule identifies the concrete IGCB: $IGCB_3$ and gives output signals to the external

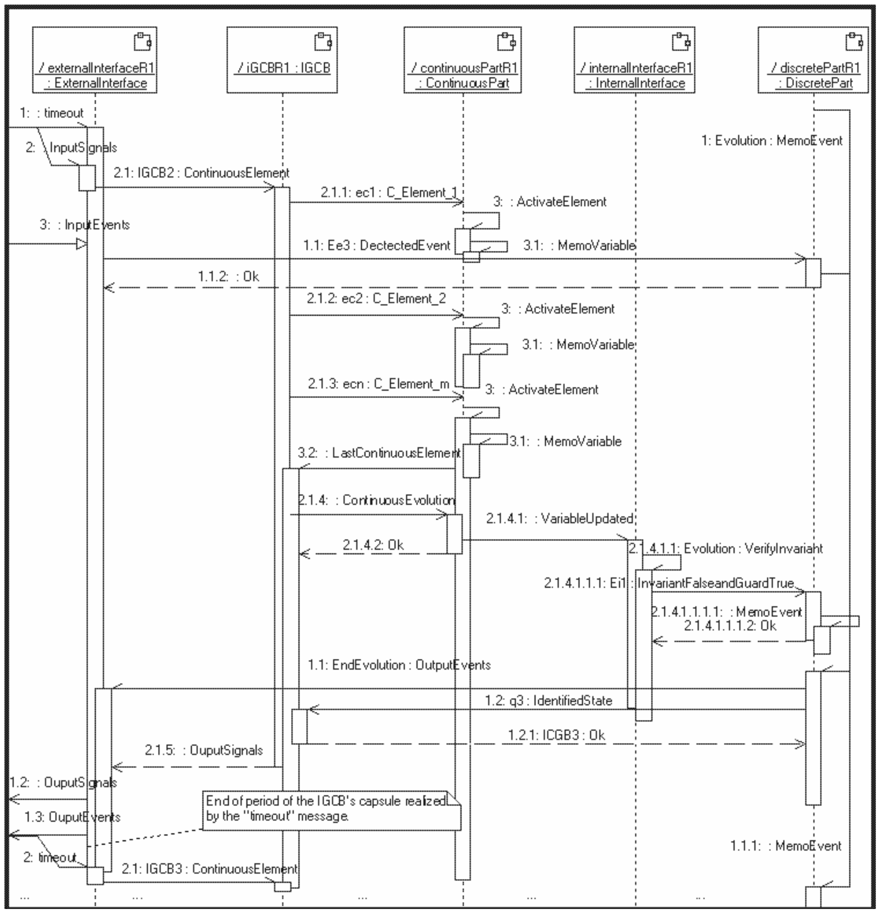

/ externalInterfaceR1 : ExternalInterface
/ iGCBR1 : IGCB
/ continuousPartR1 : ContinuousPart
/ internalInterfaceR1 : InternalInterface
/ discretePartR1 : DiscretePart
1: : timeout
2: : InputSignals
2.1: IGCB2 : ContinuousElement
1: Evolution : MemoEvent
2.1.1: ec1 : C_Element_1
3: : ActivateElement
3: : InputEvents
1.1: Ee3 : DectectedEvent
3.1: : MemoVariable
1.1.2: : Ok
2.1.2: ec2 : C_Element_2
3: : ActivateElement
3.1: : MemoVariable
2.1.3: ecn : C_Element_m
3: : ActivateElement
3.1: : MemoVariable
3.2: : LastContinuousElement
2.1.4: : ContinuousEvolution
2.1.4.1: : VariableUpdated
2.1.4.2: Ok
2.1.4.1.1: Evolution : VerifyInvariant
2.1.4.1.1.1: Ei1 : InvariantFalseandGuardTrue
2.1.4.1.1.1.1: : MemoEvent
2.1.4.1.1.2: Ok
1.1: EndEvolution : OutputEvents
1.2: q3 : IdentifiedState
1.2.1: ICGB3 : Ok
2.1.5: : OuputSignals
1.2: : OuputSignals
End of period of the IGCB's capsule realized by the "timeout" message.
1.3: OuputEvents
1.1.1: : MemoEvent
2: timeout
2.1: IGCB3 : ContinuousElement

Fig. 7. Sequence diagram of the hybrid automaton given for a concrete evolution period $[2\Delta T - 3\Delta T]$ in the Fig. 6

interface's capsule. At the end of this sample period, the external interface's capsule gives the output event and control signals to the external environment of the developed system; this system operates with its concrete IGCB: IGCB$_3$.

All the detailed static structures and dynamic behaviours of these capsules, which are described by using the state machines and sequence diagrams, can be found in [7], [11].

In addition, the re-use is very important for developing the system; because it makes it possible to reduce the time and development cost. We find different re-use view in the development phase of this system such as:

- The re-use view based on the virtual mechanism of objects, classes, or class hierarch.

- The re-use view based on design components. For example, the generic state machine of identified main capsules, industrial constraints can be specialized to develop different industrial control applications.

The specialization, which makes it possible to re-use design elements of a HDS, has been specified in [7].

4. IMPLEMENTATION AND TEST OF THE DESIGN MODEL

4.1. Implementing a HDS

To carry out a HDS, we have to convert the design model identified above into platform specific models by using the different specific technology or platforms such as .NET, Java-J2EE, Ada, IEC64199 etc... in order to obtain its implementation model. If we are only interested in the simulation of a HDS, we can use the "sub-system" paradigms, which are supported by software tools such as: *LabView-VI, MatLab-Simulink, OpenModelica, etc...* To realize a HDS, we can use industrial technology or platforms such as IEC61131 [13], [15] to realize the implementation model of this system.

4.2. Testing the analysis and design model

There are software tools such as *HyTech, CheckMate, PHAver* and *HSolver* [2] to test the hybrid automaton. We can use them to verify generally dynamic behaviours of a HDS. At this moment, it exist also formal tools such as *Rational Real-Time Test* [12] for testing directly the capsule collaboration; this tool permits us to validate a set of input sequences and defined object for ensuring the operation of the developed system. Moreover, we can find *B STERIA Object Workshop* [4] and *Rational Real-Time Architect version 7.5.2* [12], which can be used to support formally re-use of design component. They permit us to make the reverse-engineering of applications for checking requirements of the developed systems.

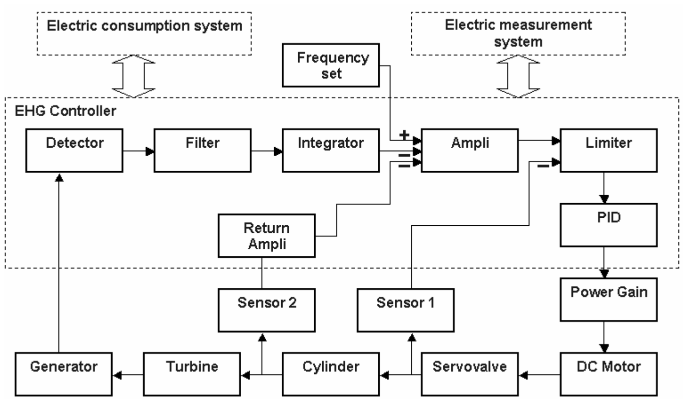
5. APPLICATION

There are many regions in the world, which are far from a national electric supplying networks, for example in Vietnam. We must then build small electric stations to improve the life of residents of these regions. The quality of the small electric stations is characterized by: the stability and precision of the frequency, tension stability, duration of operation and power.

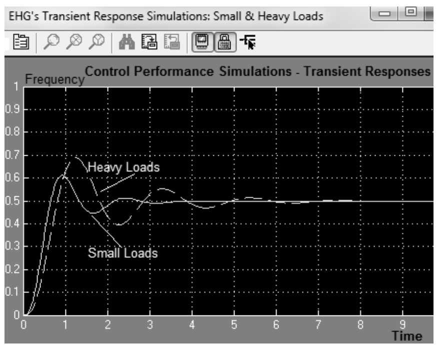
We applied our method described above to completely make the analysis and the design for an EHG (Electro-Hydraulic Governor) system, which makes it possible to stabilize the frequency an electro-hydraulic station having a power lower than 10.000 kW. EHG contains external events such as connection or the disconnection by the electrical consumption system and the internal events identified by the components such as the limiter of the control part.

We have found an extended functional diagram (Fig. 8a) to carry out activities in the state machine of EHG. The detailed features of EHG have been described in [7].

We produced all the static and dynamic structures of use cases, hybrid automata, main capsules, sub-capsules, their ports and protocols to effectively implement this application; one of the control performance simulation results is shown in Fig. 8b. All the detailed results can be are found in [7], [11].



(a)



(b)

Fig. 8. Extended functional diagram of EHG (a), Control performance simulations of an EGH for the concrete small and heavy loads (b)

6. CONCLUSION

We have developed a method, which makes it possible to make completely from the analysis to the design of the control parts of mechatronic systems, which are Hybrid Dynamic Systems (HDS) and can be modelled by hybrid automata. This method contains following important points:

- Specifying the hybrid automaton and Real-time UML to capture structures and behaviours of the developed HDS.
- Identifying main capsules for presenting the hybrid automaton of this system.
- Structuring main capsules, sub-capsules and their ports, protocols for introducing a pattern of communication between capsules.
- Indicating the hypotheses to make the evolution for all main capsules of the developed HDS.
- The global sequence diagram presents the exchange of events and signals between main capsules; the local sequence diagram introduces the exchange of messages between sub-capsules.

In the future, we will develop our approach with different formalisms and architectures in order to deploy different industrial control applications communicating by networks.

ACKNOWLEDGEMENT

This work has been partially completed by financial supports of the scientific research project at the Department of Ship Engineering and Fluid Mechanics, Hanoi University of Science and Technology.

REFERENCES

[1] Bui M. D., Real-Time Object Uniform Design Methodology with UML, *Springer*, (2007).

[2] Carloni L. P., Passerone R., Pinto A. and Sangiovanni-Vincentelli A. L., Languages and Tools for Hybrid Systems Design, *now Publishers Inc.*, (2006).

[3] Douglass B.P., Real Time UML: Advances in the UML for Real-Time Systems, *Third Edition, Addison-Wesley*, (2004).

[4] Desforges P., Utilisation de la méthode B pour la conception des logiciels critiques d'automatismes ferroviaires, *la revue RATP: Recherche et Développement*, (1997).

[5] Gamma E., Helm R., Johnson R., Vlissides J, Design Patterns: Catalogue de modèles de conception réutilisables, *Thomson*, (1996).

[6] Henzinger T. A., Kopke P. W., Puri A., Varaiya P., What's decidable about Hybrid Automata?, 27th *Annual ACM Symp. On Theory and Computing*, (1995).

[7] Hien N. V., Une Méthode Industrielle de Conception de Commande par Automate Hybride Développée en Objets, *Thèse de Doctorat, SUPMECA et Univertsité Marseille III, France*, (2001).

[8] Hien N. V., Soriano T., Implementing Hybrid Automata for Developing Industrial Control Systems, 8th *IEEE - ETFA, Nice, France*, (2001).

[9] Hien N. V., Quang V. D., Vinh H. T., Soriano T., A General Implementation Model of Industrial Control Systems Using Real-Time UML and Functional Block, *VICA6, Vietnam*, (2005).

[10] Hien N. V., Quang L., A Design Pattern of Hybrid Dynamic Systems Using Real-Time UML, *has been accepted to publish in SEATUC*, (2011).

[11] Huan P. D., An Industrial Method for the Object Oriented Design of Control Systems Using Hybrid Automata Model, *Master Thesis, HUST*, (2010).

[12] IBM Rational Software, IBM Rational online documentation, Redbooks and training kit, *https : //www.ibm.com/developerworks/university/*, (2009).

[13] Karl-Heinz J., Tiegelkamp M., IEC 61131-3: Programming Industrial Systems, *Second Edition, Springer*, (2010).

[14] OMG, UML2.x Specifications, *http : //www.omg.org/spec/UML/*, (2008).

[15] Robert H. Bishop, The mechatronics handbook: Mechatronic System Control, Logic, and Data Acquisition, *Second Edition, CRC Press*, (2008).

[16] Soriano T., Hien N. V., Using Objects Collaboration to Model the Control of an Industrial System, 7th *IEEE - ETFA, Barcelona, Spain*, (1999).

[17] Soriano T., Sghaier A., Hien N. V., Mechatronics Design From an Object Oriented Point of View, *WSEAS Transactions on Communications, ISSN*, (2004) 1109-2742.

[18] Verries J., Approche Pour la Conception de Systèmes Aeronautiques Innovants en Vue d'Optimiser L'Architecture Application au Système Portes Passagers, *Thèse de Doctorat, Université Toulouse 3 Paul Sabatier, France*, (2010).

Received February 14, 2011