Int J Interact Des Manuf (2008) 2:99–106
DOI 10.1007/s12008-008-0039-3

ORIGINAL PAPER

Using high-level models for modeling industrial machines in a virtual environment

Adel Sghaier · Thierry Soriano

Received: 18 June 2007 / Revised: 15 July 2007 / Accepted: 25 July 2007 / Published online: 19 March 2008

Abstract The work presented in this paper deals with a platform independent model formalism for designing virtual reality applications. In our approach, we focused on industrial machine simulators design. The structure of an industrial machine model is composed by "Virtual Components" which corresponds to the physical industrial components of the machine and a control part which corresponds to the functional specification of the machine. Each virtual component is modeled by VRML model for geometry and by hybrid automata (HA) for behavior. The control part is modeled by Sequential Function Charts (SFC), as it is the case in the majority of industrial machines. Those SFC are translated to HA and composed with the virtual components HA. The whole HA model of the machine is then implemented in the generic virtual environment "OpenMASK" by specific software translation tools which was developed previously. This method makes virtual prototyping accessible by the specialists in the domain of industrial machine design domain In this paper, we describe the high level modeling method putting the accent on the coupling between the control part and the process and then we apply it in the case of an assembly machine.

Keywords Virtual prototyping · Industrial machine · High level modeling · Hybrid automata · SFC

A. Sghaier · T. Soriano (✉)
Institut Supérieur de Mécanique de Paris SUPMECA TOULON - LISMMA, Maison des technologies, 83000 Toulon, France
e-mail: thierry.soriano@supmeca.fr

A. Sghaier
e-mail: adel.sghaier@supmeca.fr

1 Introduction

One of the new domains where the virtual reality can be applied is the virtual prototyping of industrial machines. The existent virtual prototype building tools are not adapted to the usual practice of industrial design engineers. Indeed, this tools requires low level coding and so the entails programming knowledge. The solution that we propose to facilitate design of virtual prototypes of industrial machines is to use formalism with a high abstraction level independent of platforms and implementation solution. Indeed, we have to keep the designer of virtual prototypes the farthest from implementation problems [1]. So, our approach has as principal goal to make the virtual prototype building accessible for the industrial domain specialists.

Two distinct parts are identified in design of the virtual prototype as in physical prototyping; firstly there are the modular components which must be defined and secondly the building of the assembly of components to get a complete machine.

1.1 Virtual component

A real industrial machine is generally an assembly constituted of various industrial standard components (motors, jacks, conveyer belts…). In order to stay as faithful as possible with the design of a real machine, our tool will allow to build virtual prototypes in the same way, that is, by assembling virtual components.

We propose as definition of a "virtual component" the pair of 3D geometric representation of solid parts and an associated behavior which is the representation of different modes of cinematic dynamic.

The behavior of a virtual components, in order to be the most realistic, must be described by different models with

continuous equations and discrete jumps between them. The best way to represent and study this type behavior is to apply Hybrid Dynamical Systems formalisms [2].

The geometry is obtained from a CAD tool in a VRML file format [3] which is accepted by most of virtual environments.

1.2 Virtual machine

From the geometrical point of view, an assembly of virtual components with cinematic relationship can be seen as a concatenation of geometrical models.

From the behavior point of view, a more complex transformation of the model has to be engaged involving degrees of freedom, inertia, mass.

The design process of the virtual machine will be developed in paragraph 2.

1.3 Virtual machine control

Once the whole machine described and in order to simulate its functions and evaluate them, a control model must be added. The control model of a real industrial machine is often sequential, boolean and realized by a Programmable Logical Controller (PLC). The most frequently encountered language for this purpose is the Sequential Function Chart (SFC). With the aim of being closed to real machines design, we use the SFC to control the virtual prototype and we have developed a method to translate SFC in hybrid automata (control HA) and then to compose them, with virtual component HA. We will develop this feature in paragraph 3.

The last paragraph 4 is dedicated to the application which is an virtual assembly machine model implemented in the OpenMASK environment [4].

2 Virtual machine design

2.1 Virtual components design

The design of a virtual prototype of an industrial machine could be a time consuming task. In order to reduce to the minimum the design time of a virtual prototype it is necessary to employ the re-usability of models. Industrial machines are generally composed of assembly of standard industrial components. That is why we have decided to consider as a basic module for building virtual machines the "virtual component". The basic module used in the majority of virtual environments is the "virtual object". This module corresponds to an elementary geometrical object associated to its behavioral model. In our case, the virtual component represents the virtual model of an industrial component. It can, therefore, be composed by several virtual objects. Once we built virtual components, these models will constitute a library of reusable models.

An industrial component comprises, in general, a static part which will be fixed on other elements of the machines and one or more moving parts which will perform the action. The displacement of the moving parts compared to the fixed parts constitutes the behavior of the virtual component. This behavior can be represented neither by continuous models nor by discrete models. This is why we use hybrid models to describe this behavior. This behavior can be described by several formalism of which we hold up as an example the hybrid Petri nets, hybrid SFC, bond graphs, hybrid automata …

We carried out a study of the most known hybrid dynamical systems description formalisms [2]. The hybrid automata formalism was chosen because it is the best formalism to model industrial components motions [5]. Indeed, this formalism allows the user to describe a rather general behavior thanks to the differential equations. Another reason of this choice was that hybrid automata profit from modeling and formal checking tools [6]. Once the choice of the formalism made, it remains the choice of the implementing method of this formalism in the chosen virtual environment (OpenMASK in our case).

OpenMASK is a virtual environment which uses modular architecture to build virtual applications. The modules composing the application are specialization of a generic object (PsSimulateObject). The Interaction between modules is realized with communications through the Data bus.

An OpenMASK module is a prototype of a C++ class containing among others the following methods:

- A method for the initialization (Init ()),
- A method of behavior computing (Comput ()),
- A method for events handling …

An OpenMASK module can also contain input–outputs for the exchange of data. Modules can exchange events or send events to the core. So we noticed very quickly the analogy between the states of the hybrid automata and the notion of module in OpenMASK. Indeed, the calculation of the continuous evolution (differential equations) can be made in the comput() method. One notice also that the transition from a continuous state to the other can easily correspond to a sending of event between modules [5].

The chosen method was to model every state of the hybrid automaton representing the behavior of a virtual corposant by an OpenMASK module. This method, in spite of appearances, does not imply a high number of OpenMASK modules. Indeed, there is only a single active state at the same time in a hybrid automaton. This consideration implies that there is only a single OpenMASK module active for each hybrid automaton.

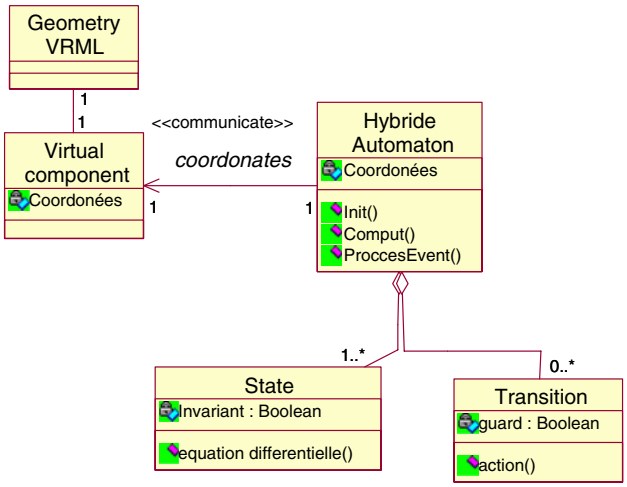


Fig. 1 Hybrid external behavior

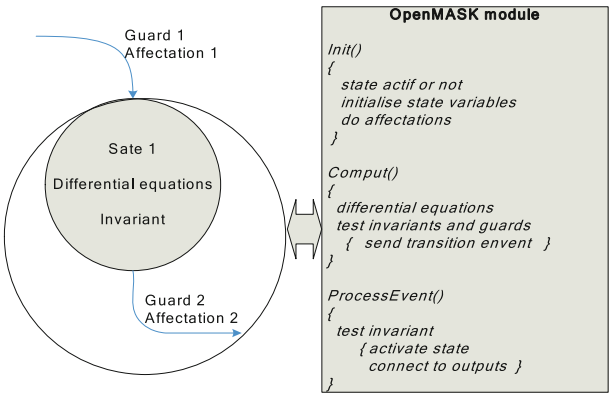


Fig. 2 From hybrid automaton state to OpenMASK module

2.2 Virtual machine design

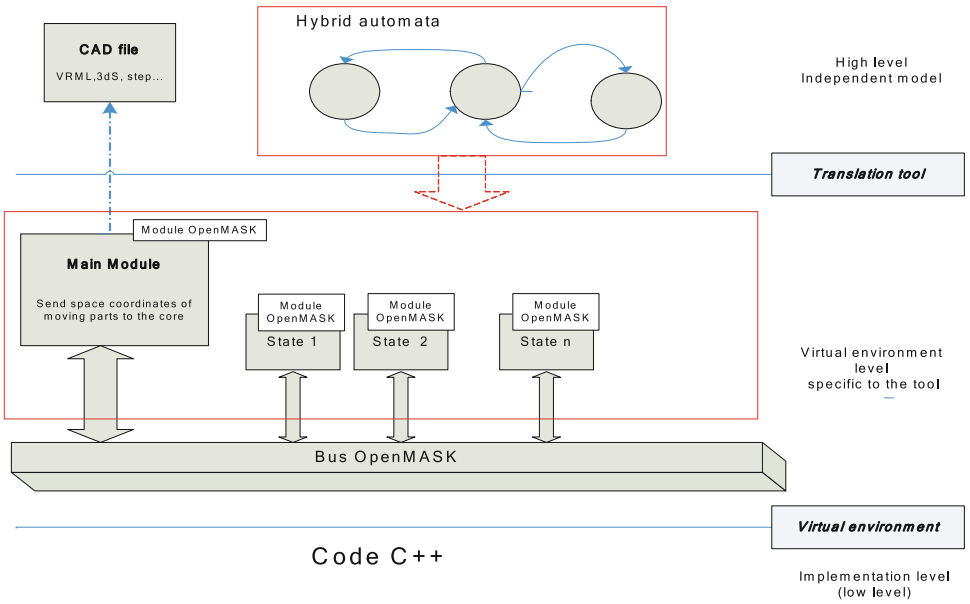
Building a virtual machine is based on the assembly of virtual components. Once the library achieved, the user has to select components needed and assemble them. This assembly has to be made with respect to the manufacturing process of real industrial machines. The assembly has to favor the future implementation in the virtual environment.

The user assembles two virtual components of an industrial machine by positioning them in the 3D scene and by defining the type of the connection which exists between their common parts. This operation can be made in a building graphic interface. Several types of connection are possible between the industrial machine components. The commonly used connections are cinematic connections and especially incitement connections. Indeed, in most cases components are fixed with each others or they are fixed to the scene (Figs. 1, 2, 3).

There is a hierarchy between the assembled virtual components. The virtual component which is generally firstly defined and fixed to the scene is the basic structure of the virtual machine. This component is then the highest hierarchical one and it is defined in the absolute referential. Every component assembled to the basic structure is not defined in the absolute referential any more but is defined including a displacement from the absolute referential.

Every time a virtual component is added, a displacement is defined (Fig. 4). The position of the virtual components should take into account hierarchical successive displacements from the initial referential. In another hand a component must take into account the motion of all components assembled to it higher in the hierarchy. To move from a

Fig. 3 Virtual component architecture



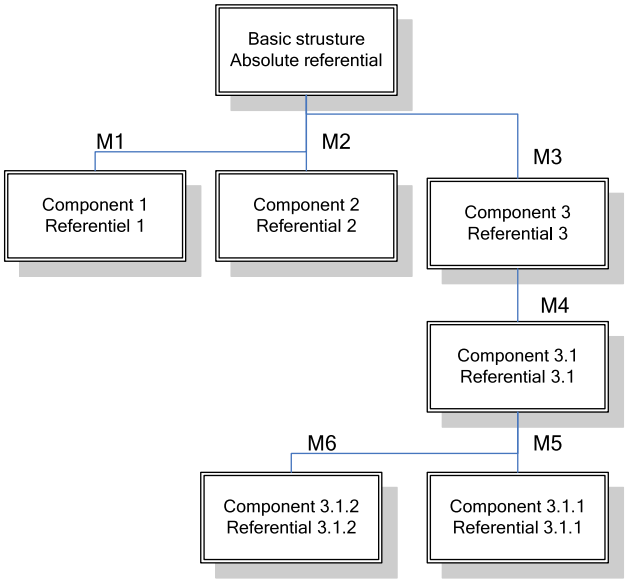


Fig. 4 Hierarchy of components links

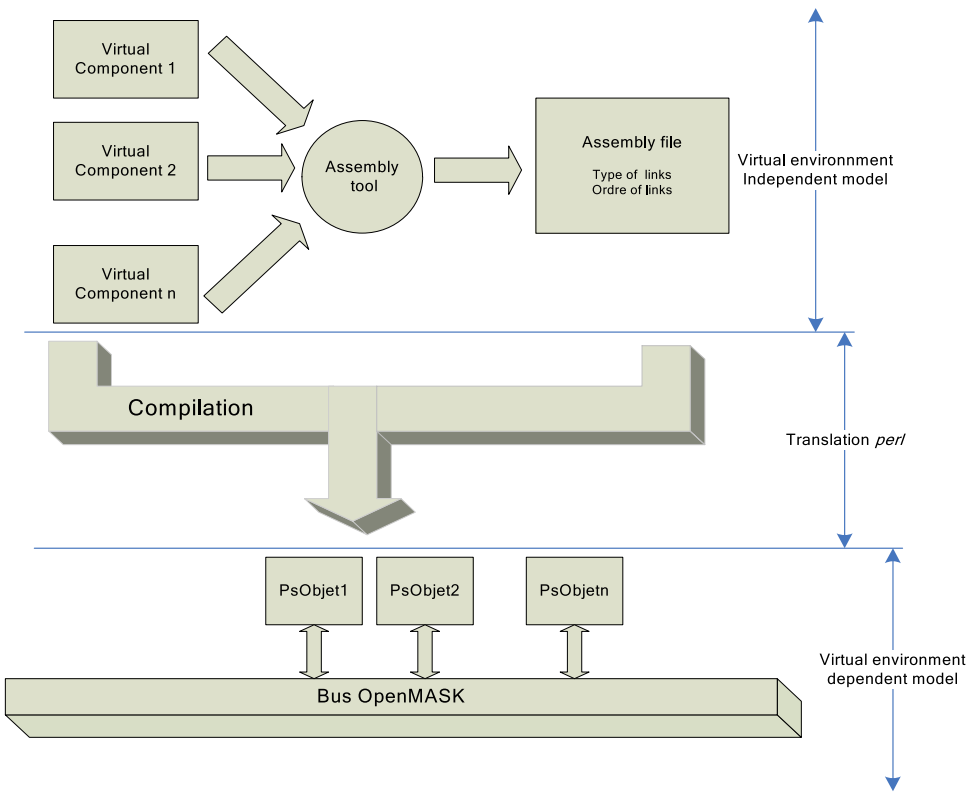
referential to an other we use the displacement matrixes. If one wants to calculate the motion of a component in its own referential, it is necessary to multiply his local motion by all the displacement matrixes of the components which are higher than him in the hierarchy of connections.

The specification of connections will take place during the assembly of the virtual machine. Indeed, the assembly software generates an assembly description file in which it specifies the position of every component and its connections with the others. In this way the association of the models of the virtual components with the assembly description file constitutes a high level model completely independent from the virtual environment (Fig. 5). Indeed we can generate from virtual machine models the corresponding model in the virtual environment which we want to use by the way of the adequate translation software. In our case, we are interested in the OpenMASK virtual environment. This way to define models independent from platforms is very similar to the platform independent model (PIM) and platform specific model (PSM) in the Model driven architecture (MDA) in the UML2 language [7].

3 The virtual machine control

The model defined previously represents geometry and dynamic behavior of the machine. This virtual prototype cannot run without a control model. As it was said previously the control model for real machines is often described in SFC language and it was decided to follow such an industrial standard. This will allow to test control scenarios on the virtual machine before implementing them in the real machine.

Fig. 5 Virtual machine assembly



In order to control virtual components with SFCs, the most natural solution is to translate SFCs in hybrid automata (HA) and then to couple them with the virtual machine HA. This path is possible since a hybrid formalism like HA has a great expressive power for discrete event systems and continuous equations systems. This leads to use HA states without the continuous functions description inside them. In the translation, there is still a difficult point which is the representation of concurrent states. They are supported by the SFC formalism and not by the HA one.

This transformation has already been studied [8]. In their approach, authors proposed to generate the situation graph from the SFC. This situation graph represents the tree of all the possible SFC situations encountered in the dynamic exploration. The advantage of the method is to obtain only one hybrid automaton with only one active state at each instant and so to save computing resources. The most important drawback is that this kind of exploration leads generally to combinatory explosion when divergence structures are used in the model. Thus, finally, it is difficult to retain such a method in the case of industrial SFC.

We have been led to develop an new method to translate SFC into HA. This method is based on the rule that for each "AND" divergence contained in the SFC, a new independent HA is created. The "OR" divergence is treated inside the same HA. So, there is no combinatory explosion.

This method of translation is applied with a certain number of hypotheses. Indeed, we try to exclude cases which risk to weigh down the algorithm of translation. The hypotheses of departure are the following ones:

- Uniqueness of the initial step in the control SFC,
- Symmetry of divergences (the same number of branches),
- There is at least a step between every divergence,
- "or" divergences are exclusive.

3.1 Translation algorithm

The following translation algorithm is the skeleton of the translation algorithm developed in an internal report [8].

1. Position on the initial step of the SFC
2. Build the corresponding initial state in the main hybrid automaton (HA) and build the initial transition
3. Test divergence presence
   - If "and" divergence go to (13)
   - If "or" divergence go to (8)
   - If no divergence go to (4)
4. Go to the next step of the main SFC or branches.
5. Build the next state in the HA corresponding to the step and build a transition from the previous state
   - If treating the main branch go to (6)
   - If "and" go to (16)
   - If "or" branches go to (10)
6. Test the end of the main SFC
   - If the end go to (7)
   - If not go to (3)
7. Build transition from the current state to the initial state and finish
8. Position on each first step of each branch
9. Build a state in the main HA for each branch and build transitions from the previous state the each of built state.
10. Test convergence presence
    - If convergence go to (11)
    - If not go to (3)
11. Position on the next step in the higher branch and if corresponding state is not built, build it.
12. From each state of end of the divergence build a transition to the current state and go to (3)
13. Build an initial state of an auxiliary HA for each branch with coupled initial transitions
14. Position on the first step of each branch

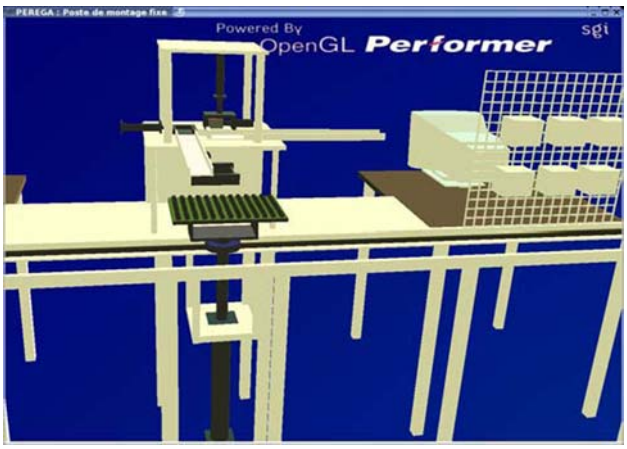


Fig. 6 The model of the assembly post of the flexible cell

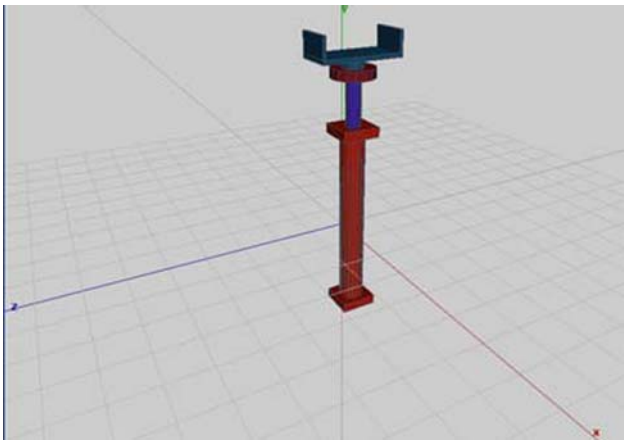
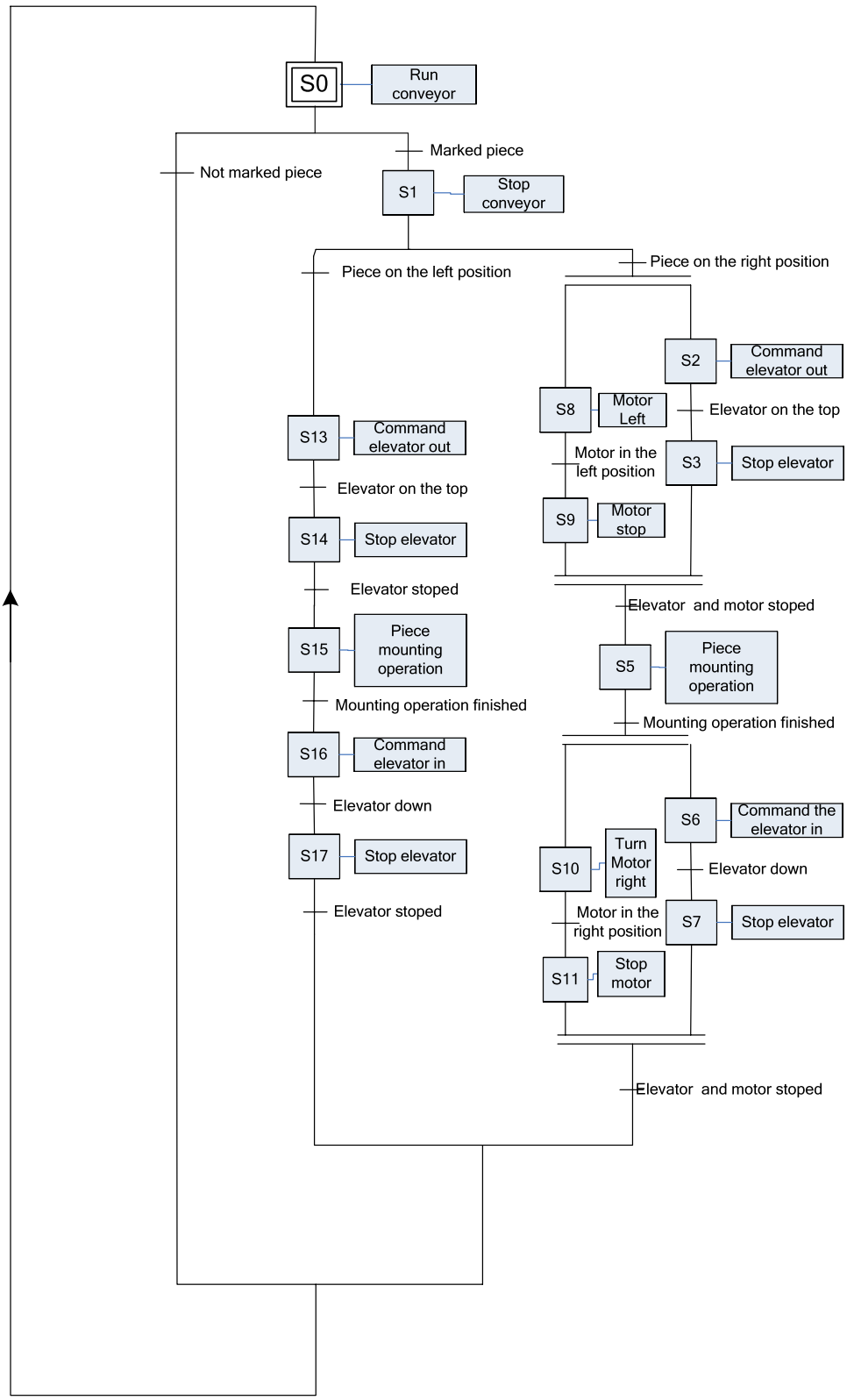


Fig. 7 The motor assembled on the elevator

Fig. 8 The control SFC



15. For each step build a second state in the auxiliary HA for and a transition from the initial state
16. Test convergence
    (a) If convergence go to (17)
    (b) If not for each branch go to (3)
17. Position on the next step in the higher branch

18. Build the corresponding state and build a transition from the last state in the higher branch and transitions from the last state in each auxiliary HA and couple them with the precedent transition and go to 3.

4 Application

The complete design process of an industrial machine virtual prototype is applied to the particular case of an assembly station of an flexible cell. We put the stress in this example on the modeling of the control of the machine and the coupling of the control of the station and its operative part.

The operation of assembly consists in awaiting the presence of the one of the plates which are transported by a conveyor belt. Once the detection of a plate occurred, and if that one is marked, a piece must be mounted on the free side of the plate considering that the marked plates are those which do not comprise piece on one of their sides (Fig. 6).

This example allows us to illustrate the modeling of an industrial machine from the operative and control point of view. Operative part model is constituted by the graphical model and by the behavioral one. Graphical models were realized with a CAD tool (AMAPI 3D) and were exported as Inventor files. We associate to every virtual component a behavior described the hybrid automata formalism. Only the conveyor belt, the motor and the elevator behavioral models are detailed to reduce the size of the model.

If one considers the example of the assembly of the elevator, which is connected with the basic structure of the machine, with the motor (Fig.7). The motion of the motor is dependent on the motion of the elevator. So the behavior of the assembled motor is different from its autonomous behavior. The motion of the stem of the jack in its referential (R) is modeled by the following displacement matrix:

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

and the motion of the motor compared its referential (R) is modeled with:

$$M_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(H') & \sin(H') & 0 \\ 0 & -\sin(H') & -\cos(H') & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

Knowing that the motor element is the son of elevator element in the hierarchy tree and that the elevator itself is the son of the basic structure of the machine, the motion of the motor in the absolute referential is computed using the following displacement matrix:

$$M_2^{'} = M_1 * M_2 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(H') & \sin(H') & Ty \\ 0 & -\sin(H') & -\cos(H') & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

The control of the machine is described by the SFC of the Fig. 8. Through this simple example we can efficiently test the design method of the control part considering that we encounter most of the divergence cases of the SFC (Fig. 8).

We does not detail in this part the operation of piece assembly in order to reduce the size of the example. The translation algorithm is applied to the control part of the machine and we obtain the control hybrid automata described in Fig. 9.

It is noticed that the two "And" divergences of the SFC are translated to two auxiliary hybrid automata (Fig. 10).

Each state of the control hybrid automata is defined by the differential equation of the evolution of time (we do not use the continuous states for the control SFC). The transition invariant corresponds to the complement of the decision of the SFC. The transitions from the control hybrid

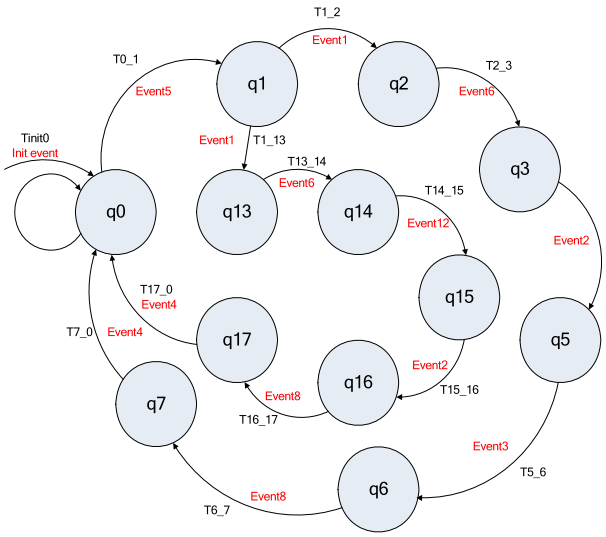


Fig. 9 The main branch control hybrid automaton

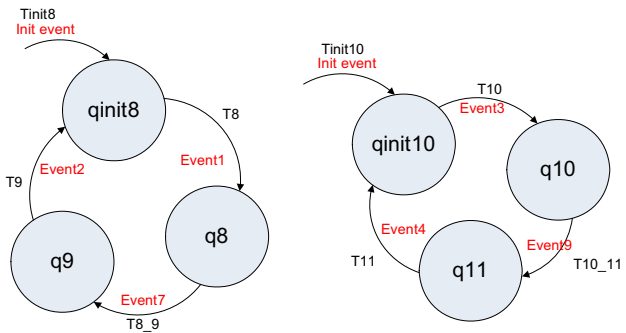


Fig. 10 The "And" divergent branches control hybrid automata

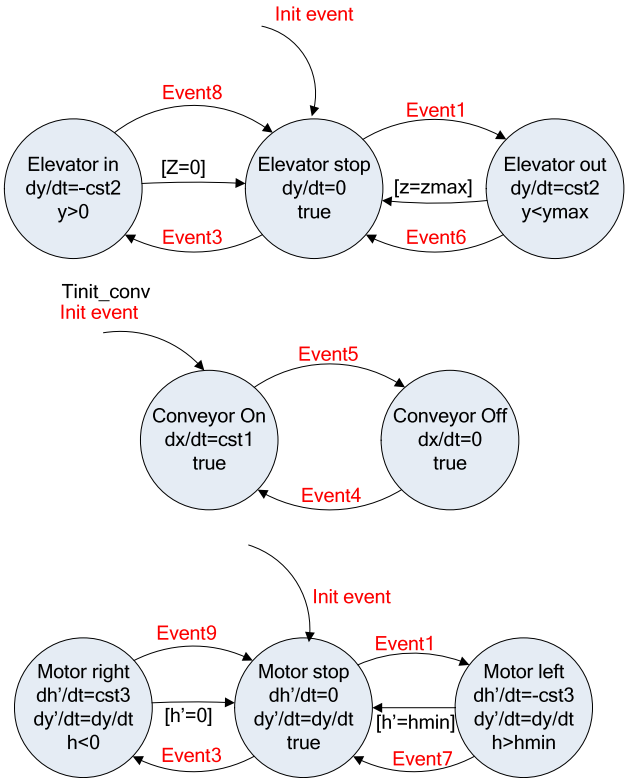


Fig. 11 The operative part hybrid automata

 # Assembly file

Comp1 $(x_1, y_1, z_1, H_1, P_1, R_1)$ LinkType/Compi
Comp2 $(x_2, y_2, z_2, H_2, P_2, R_2)$ LinkType/Compj
LinkType/Compk

. . .
. . .
. . .

Fig. 12 Assembly file

automata are conditioned by the guards which correspond to the SFC decision. Each transition of the hybrid automaton generates an event. These events are used to couple the main control hybrid automaton with the auxiliary ones and to control transitions of the virtual components hybrid automata. For example the transition T1–2 (Fig. 9), which represents the cross between stage 1 and 2 of the SFC, will control the output of the elevator (Fig. 11) throw the event 1. It also will control the activation of the auxiliary SFC (Fig. 10) throw the same event. The result of the implementation of the whole method is illustrated in the Fig. 10. This little example was made to illustrate the method developed before (Fig. 12).

5 Conclusion

We develop in our approach a method for modeling industrial machine on a high level of abstraction. The high level model of the machine describes the geometry as well as the behavior. This model is completely independent of the virtual environment. The use of a translation software implementing this method in a virtual environment (OpenMASK for example) allows to generate a virtual prototype of a machine.

The virtual prototype of the machine thus generated will be used as test and training platform.

Translation tools for implementing the high level models under OpenMASK are developed using the PERL language. In another side we are developing a graphical interface for specifying the high level model. In order to improve the immersion of user, the tool could integer in the future compatibility with external interfaces.

References

1. Boeck, J.D., Cuppens E., Raymaekers, C., Conix, K., Flerackers, E.: High-level interaction modelling to facilitate the development of virtual environments. IEEE VRIC (2004)
2. Zaytoon, J.: Systèmes dynamiques hybrides. Hermes science publications. (ISBN 2-7462-0247-6) (2001)
3. Dille, Ed. Explorer Moving Worlds. VRML 2.0 et les mondes animés sur le web. ISBN 2-84180-146-2. International Thomson publications, London (1997)
4. Duval, T., Le Teneir, C.: Interactions 3d coopératives en environnements virtuels avec openmask pour l'exploitation d'objets techniques. J. Mécanique Ind. **5**, 129–137 (2004)
5. Sghaier, A., Soriano, T.: Modeling parts behavior on virtual environments. In: IEEE VRIC 2004, Laval, France (2004)
6. Henzinger, T.A., Ho, P.-H.: Hytech: the cornell hybrid technology tool. Lecture Notes in Computer Science, LNCS 999, pp. 265–293. Springer, Heidelberg (1995)
7. Soriano, T., Sghaier, A., Turki, S.: Towards a pim for virtual prototyping. WSEAS Trans. Commun. **1**(3), 1707–1713 (2004)
8. Sghaier, A.: The sfc to hybrid automata translation algorithm. SupMECA Internal note (2005)