A Dynamic Construction Algorithm and Verification of the Grafcet

Marcellin Nkenlifack, Emmanuel Tanyi and Janvier Nzeutchap

Abstract— This article proposes an approach of interactive construction of Grafcets. We especially studied the problem of the research of a verification method and of holds it of the consistency, and we arrived to formulas and an algorithmic method of interactive construction and progressive checking of this characteristic that we present. This algorithm has been implemented and has been examined on convenient examples. The developed environment includes in more, a module of description and a module of simulation. The object approach and the UML language have been put to profit during the process. We illustrate functionalities of the simulator to specifics systems.

Index Terms— Hybrid Dynamic Systems, Algorithms, Verification, Simulation, Sequential Systems, Grafcet, Validation, Human-Computer Interface.

1 Introduction

THe present methods of production are more and more complexes in the industrial enterprises. They undergo important technological transformations and need in general on the computerization, in order to reduce to the strict minimum the physical stress as intellectual of the man. This evolution is characterized by a spectacular development of the programmed systems. Automatic control systems are often modeled either as continuous systems (equations, transfer functions) or discret systems (Grafcets, Petri Nets). In practice, however, most industrial control systems incorporate both continuous and discret elements. These so called hybrid systems have recently become the subject of intensive research [1] [2] [3] [4] [5] [6] [7]. The research on the automatic industrial checking systems tackle to solve the following questions [1] :

- The modelling

It is about having resort to a "system approach" structuring the different objects while taking into account the physical sense and the causality of their interactions.

- The analysis

It assumes the development of tools of checking and validation of the Hybrid Dynamic Systems (HDS), then the mastery of the complexity of this analysis and the physical interpretation of some qualities to analyze as the global stability of the system through his consecutive phases of performance.

- The Simulation

The present research concerning the methods and formal tools relative to the analysis of the behavior of the HDS and to the synthesis of the command laws are some again to their beginnings [8].

The simulation remains therefore an obligated passage when it is necessary to help towards the implementation of an installation, to enable the model elaborated (in a goal of forecast) for an existing installation, or to validate the command conceived for an installation.

We are interested in the implementation of an environment of verification of the Grafcets in construction. This paper is a shutter of an a lot vast project on the setting up of a modelling and simulation of the hybrid process automations including the discret systems (described by the Grafcets) and the continuous systems (described by the differential equations), combining the techniques of the software engineering and the control engineering, and having been the object of several experimentations and subjects [2] [4] [5] [6] [7] [9] [10] [11] [12].

Our paper that describes the implementation of a checking environment and validation of the Grafcets in construction, start with a brief presentation of the automatic industrial control systems and the Grafcets, follow by the modelling of the components, the specification of the criterias of validation of the modules in construction and the algorithmic design. After the techniques of implementation used, we illustrate the setting through the simulation of the Grafcets of the command system of a mill.

2 The Automatic Industrial Control System

The different methods and tools (specification, analysis, simulation) are numerous on the mathematical plan, in control engineering and software engineering. Some approaches and basic mathematical tools that can be operated are described in [1] [13]: Theory of command, classified Command, Boolean algebra, Graph theory, Petri networks (one of the forebears of the Grafcet), Statemachines or Statecharts.

- *M. Nkenlifack is member of LAIA Laboratory, IUT-FV, University of Dschang, Cameroun , E-mail: marcellin.nkenlifack@gmail.com*
- *E. Tanyi is member of ACL Laboratory, National Polytechnic, University of Yaounde 1, E-mail: emmantanyi@yahoo.com*
- *J. Nzeutchap is member of LITIS Laboratory, University of Rouen, France, E-mail: janvier.nzeutchap@univ-rouen.fr*

One distinguishes three approaches of description of the Automatic Control Systems [1] [14] :

- The Discretes Systems (DS)
- The Continuous Systems (CS)
- The Hybrid Dynamic Systems (HDS) combining the two first approaches.

The GRAFCET (in French "*Graphe Fonctionnel de Commande Etape Transition*") is a "language" of the control engineering, adapted to the description of automatic systems. It is also a normalized tool [16], and functional flowchart whose goal is to describe (graphically) the different behaviors of a sequential automatic system [15].

The Grafcet formalism has some constructs - STEPs, TRANSITIONs, LINKs, ACTIONs, and TRANSITION CONDITIONs -. Steps, denoted by numbered rectangles, represent the steps in a sequence. Transitions, denoted by horizontal bars, define the logical conditions which govern the passage from one Step to another. Links are vertical lines which interconnect Steps and Transitions. The Actions, denoted by long rectangles connected to Steps, define the operations performed by the actuators of the system when the associated Steps become active.

It is necessary to note that today, a lot of researchs lean on the Grafcet and try to improve it or to add some add-on facilities there [1] [3] [6] [7] [12] [14] [17] [18].

The Grafcets are also a tool of functional specification of some types of Hybrid Automatic Systems. As particularity of the Hybrid Systems, one has the Interactions (mutual action between parts of the System). The sequential-continuous Interactions is materialize to the level of the Actions (Steps of the Grafcet). The continuous-sequential Interactions recovered to the level of the Transition condition bound to the Transitions of the Grafcet.

An excerpt of the Rolling Mill Grafcet is shown in figure 1. The details of this Hybrid System are provided in [3].

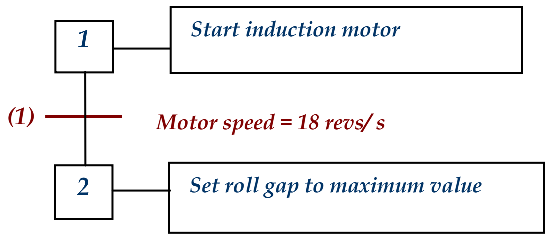


Fig. 1. An Excerpt from the Rolling Mill Grafcet (a hybrid system)

The problem studied in the article dedicates itself to the interactive construction of a coherent specification of a system. The Grafcet will be described on differents plans: lexical (construction), syntactic (description), and semantical (functional). To every time, one of the checking algorithms is started to permit to alter the Grafcet. At the end, it is simulated.

3 Definition of a Control Mode of the Algorithms and Modelling of the Components

3.1 Conceptual survey of the constraints on the Sequential Systems (Grafcet)

We haved specified several constraints regrouped by types. In this article, we are going to quote some of it for every category, as illustration. The reader can find the list and details of all other constraints in [2].

- **Technical constraints**

- *The constraints bound to the studied systems*: example, the C4 constraint determines the layout of the elements of a Grafcet while imposing that the alternation Step-Transition and Transition-Step is always respected.

- *The constraints of programming*: To the level of the description, C9 indicates that the crossing of a vertical directed link with a horizontal directed link can be admitted without it corresponds to a relation between these links. To the level of the simulation, the C12 constraint indicates that the evolution of the corresponding Grafcet to the validation of a transition cannot occur until the transition condition bound to this transition is true.

- *The constraints of modelling*: The constraint static structural C14 on the values of attributes and the cardinalities indicates that a "parallel Link" will have a "number of pathes > = 2" and that a "multiple Action" will have a "number of applicable actions > = 2". The constraints of uniqueness as C15 guarantee the unicity of the objects of a class from the uniqueness of the values of an attribute or a set of attributes of the class (example: a Step or a Transition may possess a unique number). The constraints of inheritance as C16 restrict the possibilities of existence of specialized objects (example: the exjunction is obligatory between an unique link and a parallel link).

- **Datas and elements which is manipulated**

It is here about identifying the different phenomena of our domain, susceptible to be represented by objects and classes. We specified several score of classes of objects, all listed in [2].

- **Operations, events and processings**

For every script, a set of operations has been specified and modelized. One has several hundreds of them to the total. For example:

- To represent or to describe the Grafcet, we will need to draw the elements as the "Actions", the "Steps", the "Transition conditions", the "Transitions", the "Links", etc. One will bind them the activities as *Relocate, Replicate, Delete, Modified, Mark, Find, Select …*

- To simulate the Grafcet, we need the activities as *Activate and Deactivate* any Step, to *Examine the Transition condition*, to *Enable* and to *Validate any Transition*, to *Execute one or several Actions*, to *Execute the Actions in way of other systems*.

3.2 Object-Oriented Modeling of Grafcets and Interactions

With the UML language [19], we have modelized in general all points of view of a sequentials systems, what made several score of diagrams [2]. Only some will be presented in this article. The Use-case diagram (figure 2) represents the functions of our system and gives the views that the users have to the system activity.

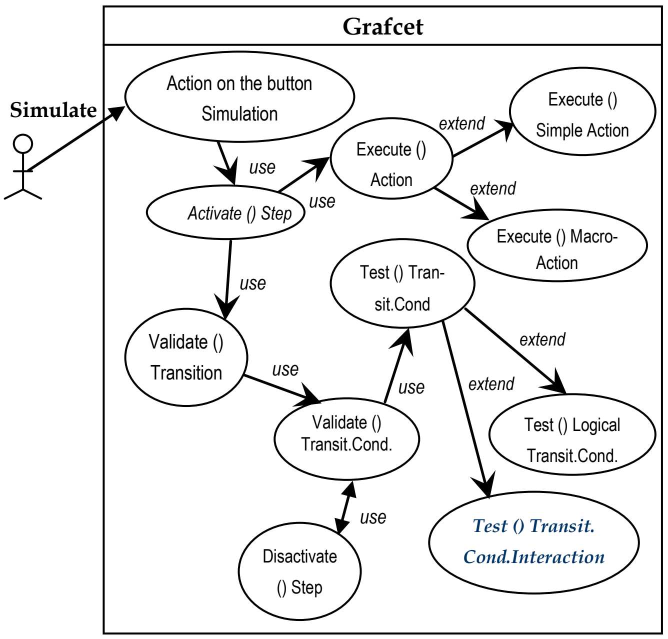


Fig. 2. "Use-cage diagram" of the sequential systems and interactions with the continuous systems

Each Grafcet is part of a system which is defined as a schematic diagram. The simulation of the Grafcet provides information on its functional behavior. The top-level objects in the model base, thus, include SCHEMATIC, GRAFCET and SIMULATION. The basic class diagram of sequentials systems is presented at figure 3.

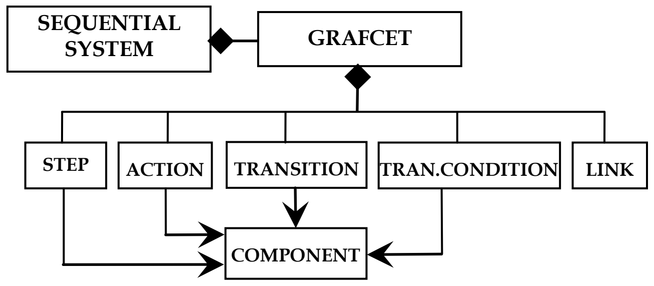


Fig. 3. Basic class diagram of sequentials systems

These top-level objects were successively decomposed into other lower-level objects to obtain the class hierarchy (inheritance and composition) shown in figure 4. The description of all attributes of the different classes (static attributes and methods) is made in [2].

3.3 Interactions

Interactions are included in each sub-system. They are modeled by four object classes, two for sequential and two for continuous systems. The class ACTONSYS (Action applied on continuous system) contains attributes and functions which enable a sequential element to apply actions on a continuous system. The class ACTFROMSYS (Action from a sequential system) contains attributes and functions which enable a continuous system to apply an action initiated by a sequential element. The class TRANONSYS contains attributes and functions which enable a transition condition in a Grafcet to be assigned a value by a continuous element. The class TRANFROMSYS (transition resulting from a continuous system) contains attributes and functions which enable a continuous element to assign a value to a transition condition in a Grafcet. The objects ACTONSYS and ACTFROMSYS define the source (cause) and destination (effect) of a sequential-continuous interaction. TRANFROMSYS and TRANONSYS define the source and destination of a continuous-sequential interaction. The objects TRANFROMSYS and ACTONSYS are shown in figure 4.

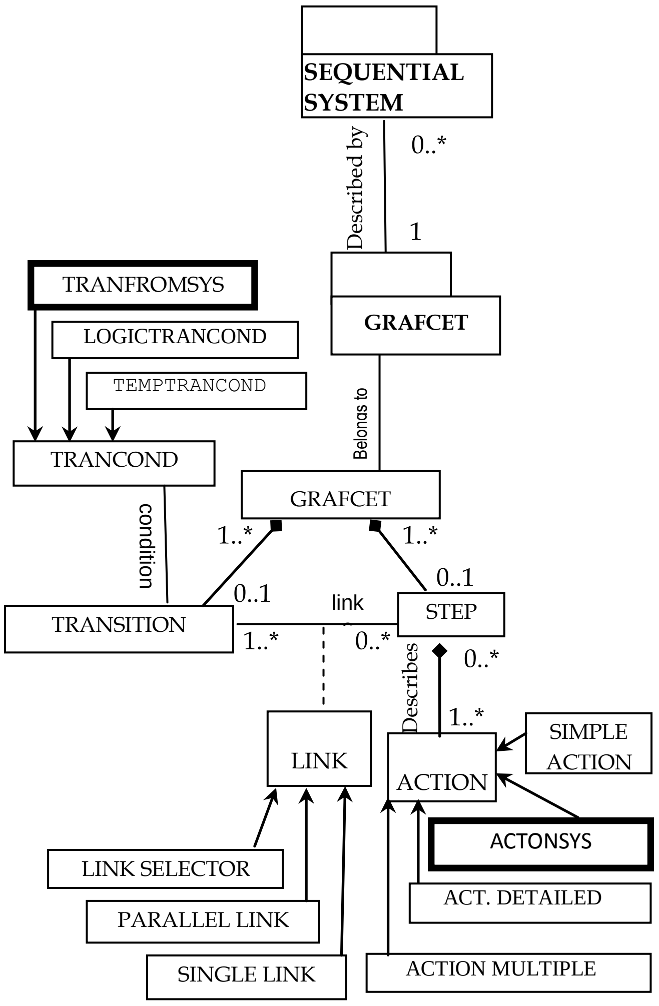


Fig. 4. Hierarchy of Classes for Sequential Systems

3.4 When does need to verify the components of the Grafcet who are created?

After the analysis, two cases are foreseeable:

- One only permits the creation of valid components. Thus, all attempt of creation of a component of which the layout with the existing components would cause incoherence in the global schema fails: the component is not created in this case.

- One allows the "in bulk" creation of a set of components (linked or no), and one doesn't perform the check that when a component comes to be bound assembly-line starter of the initial component (Step initial zero).

We adopt the second solution that doesn't impose a too big rigor on behalf of the user, permitting more flexibility thus in the constructions. Otherwise, we also suggest the automatic creation (drawing) of an initial Step number zero (0) every time the user chooses to begin a new Grafcet. The one is not here deletable by the user and will act as source to identify the components already linked, therefore controlled, of those not making even gone of the Grafcet, although present in the window of description.

3.5 Stated of the principle of creation of the components

A progressive verification algorithm assumes the possibility to verify every component individually, at the time of an attempt of creation. Thus, every new component will be created temporarily in memory. If the user wishes to bind it already in the correct Grafcet, then the component is checked. If it is in harmony with the existing schema, the program connects it to the other components of the chain (that is to say in the Grafcet), otherwise the attempt of creation is annulled by a trouble report indicating the nature of the incoherence, and the component is suppressed of the memory. On the other hand, a created component and non linked in the Grafcet by the user will be marked "unchecked" and considered like not being part of the Grafcet. The user will be able to then either to suppress it, either to relocate it toward another position. Verification is performed at the end of this displacement. Nevertheless, at the end of the construction of the Grafcet, the program will propose to automatically suppress the created components and non linked to the main schema.

4 Conceptual Specification of the Criterias of Validation of the Modules in the Construction

We enumerated in [2] [6], a certain number of criterias that the set of combined elements must respect. These criterias define for every type of component (STEP, ACTION, TRANSITION, TRANSITION CONDITION, LINK), the nature of the components to which this one can be bound up-stream or down-stream.

To permit the correct checking of these constraints, we have in our basic class model, in the global class "COMPONENT", define an attribute "Previous" that permits to know what components are directly above the concerned component, an attribute “Next” which is a list permitting to get all components directly connected to a down-stream component, a "Left" and a "Right" attributes, permitting to know what components respectively has been bound directly to the current component in his Left or in his Right.

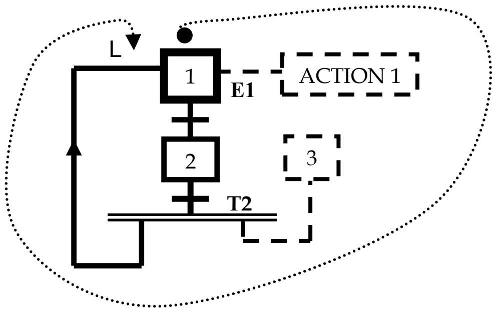
We will use these attributes to perform the verification. The verification will be in a first time specified for every category of components, then we will describe the algorithm of construction of the Grafcet that will call on the algorithms of individual check of the components, every time that a new component will be under creation.

- The notions of “Left”, “Right”, “Previous” and “Next” don't correspond to physical positions in the window of construction. It is more about a concept bound to the internal organization of the Grafcet. A component can be placed below another in the window of construction, and to be considered yet as the "Previous" of this one.

- To establish these criterias of validity, it would be indicated to make a schematic representation of the possible cases by type of components. Let's note that it will be difficult enough here of to enumerate all possible cases. We choose some for this article.

4.1 Notion of loop of a component

We say that components are looped if it is possible to revert on the same component while covering the Grafcet during the simulation of the system, while only crossing the Next one of the elements met. The figure 5 is an example of illustration of the loop notion.



- All elements in continuous lines are looped

- Tthe elements in interrupted lines are not looped

- The Transition (T2) and the Step (E1) in bold are respectively the "Start" and the "End" of the created loop

- The curve oriented in dotted lines indicating the sense of creation of the components, one will notice that it is the creation of the link (L) that generates the loop

Fig. 5. Illustration of the loop notion

What interests us when a loop is created, are the elements of “Start” and “End”. It allows us to verify the following constraints:

- if the "Start" is a Transition, then the "End" must be a Step.

- if the "Start" is a Step, then the "End" must be a Transition.

No other configuration of loop is admitted.

In general, it is the creation of a new Link that generates the possibility of existence of a loop. Thus, the test of existence of loop will be performed to the creation (by an user) of every new Link. Thus, the algorithm of verification of a component will call on the algorithm of check of "loop" for this component.

As having presented the notion of loop, we are going to enter the different configurations that we could have during the construction of the modules.

4.2 Case of a Step

For a Step, we counted some of the situations capable to occur. They are presented to the figure 6.

One can carry therefore that:

- The element "Right" of a Step always exists, is unique and is a Link toward an Action. Unusually, this element is not obligatory for the case of an initial Step or a final Step for which the system is pending or to the pause (stop).
- The "Next" of a Step always exists, is unique and can be a Transition (when the Step is bound directly to his transition of exit) or a Single-Link, when the Step is bound to a Transition by a succession of links (constrained linked to the available drawing space in the window of construction). Exception, this element doesn't exist for a final Step, last Step (stop) of the Grafcet.
- The "Previous" of a Step always exists, is unique and can be a Transition (when the Step is bound directly to his transition of entry) or a single Link. Nevertheless, this element doesn't exist for the case of an initial Step.
- The "Left" of a Step, when it exists, is a Link coming from a chain of links. It can have of it several of them when the Step is "looped" several times.

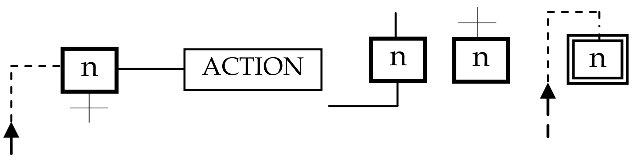


Fig. 6. Checking of the Grafcet: Some possible situations for a Step, during the construction

4.3 Case of an Action

The "Previous" of an Action always exists and is a Single-Link toward a Step. Its "Next", "Left" and "Right" doesn't exist.

4.4 Case of a Transition

In addition to the previous configurations, one can count the few supplementary cases of the figure 7 :

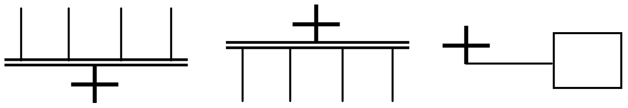


Fig. 7. Verification of the Grafcet: Some possible cases for a Transition, during the construction

One can carry that:

- The "Previous" of a Transition always exists and can be a Step or a Link (Single or Parallel). In the same way for the "Next" one of a Transition.
- If the "Previous" of a Transition is a Link, and possess several "Previous", then this Link must be a Parallel-Link-Sequence. This check will be performed on the Link.
- In the same way if the "Next" of a Transition is a Link and possess several Next one. Once besides, this check will be performed on the Link.
- The "Left" of a Transition when it exists is a Link (Simple or Oriented). It can have several of them when the Transition is "looped" several times.
- The "Right" of a Transition when it exists is a Link (Simple or Oriented). It can have several of them when the Transition is "looped" several times.

4.5 Case of a Transition Condition

A Transition-Condition doesn't possess "Previous", nor "Next", nor "Left", nor "Right" element. It is not physically linked to any component in the window of construction. But in the internal structure of the Grafcet, It is associated with an Action and a Transition. It is sufficient to place it enough close to this Transition in the window of construction then to put this association in evidence.

4.6 Case of a Link

The particularity of the Links is that they can be bound at any other component. The "Previous" and the "Next" of a Link always exists and is unique. The "Left" and the "Right" of a Link (Simple or Directed Link) when they exist are Simple or Directed Links. It can have of it several of them when the Link is "looped" several times.

5 PRESENTATION OF THE ALGORITHMS DESIGNED

The algorithms that we propose correspond precisely to the class model and to the constraints as stated or definite previously.

5.1 Algorithm of "loop check"

This algorithm will be applied in priority to the components of type "Link" because these last are those susceptible to generate a loop in a Grafcet. It uses the following algorithms:

- ***Algorithm*** of determination of the "Start" element of a Link: Permits to know the component that uses a Link to connect to another component in the Grafcet;

- ***Algorithm*** of determination of the "End" element of a Link: it is the first element different of a Link situated downstream this Link in the construction;

- ***Algorithm*** of determination and checking if a component comes after another in the Grafcet. This algorithm is not the one of recognition of the element or elements following immediate of a component. It has for goal to determine if while leaving from the C1 component in the Grafcet, one can arrive to the C2 component while only covering the Next one of the components met. This algorithm is "recursive". Indeed, to determine if C2 comes after C1, one determines successively if C2 comes after every immediate Next one of C1. It is a sub-algorithm very determinant that will serve to verify if the "Start" of a loop comes after the "End" of this loop.

During the research of the "Start" and "End" of a Link, a variable is use to count the number of valid Directed Links met. The validity test of a Directed link is performed as soon as this Link is connected to another component in the Grafcet. We won't specify his algorithm here. Also, a loop must comprise at least one Directed link to indicate the sense of runaround of the loop.

Skeleton of the algorithm of loop check :

This algorithm permits to determine if a link (L) generates a loop in the Grafcet.

```
D ← L.depart() ;  // call Algorithm of research of the Start of the link
A ← L.arrivee() ;        // call Algorithm of research of the End of the link
n ← nLiaosonsOrienteesTotalRencontrées ;
Si (D = NULL OU A = NULL) Alors resultat = lienNonBouclé // isolated or partially
isolated link
Sinon
        Si D vientAprès(A)  Alors
        Si n = 0 Alors    // component looped but without directed link: mistake
            messageErreur("transformer un lien en une Liaison Orientée") ;
        Sinon
        Si  (D.type() = Transition ET A.type() = Etape)  OU
            (D.type() = Action ET A.type() = Transition)
        Alors resultat = lienBouclé ;
        Sinon             messageErreur("mauvais type de boucle") ;
        Fin si
        Sinon
        Si n ≠ 0 Alors messageErreur("supprimer la Liason Orientée")
         resultat = LienNonBouclé ;
        Fin si
Fin si
Fin test de bouclage.
```

5.2 Algorithm of validity checking of a component

In the beginning, one initialize to zero (0) the numbering of components (they follow a numbering). When a component is enabled, if it must be numbered, then his number is automatically gotten while incrementing to 1 the number of the previous component (in the same way type). What allows us to have the following schema:

numEtapeCourant $\leftarrow 0$; *numTransitionCourant* $\leftarrow 0$; etc.

Algorithm of check of a Step: *Determine if an element of type "Step" is bound correctly in the internal structure of the Grafcet.*

Algorithm of check of a Transition: *Determine if an element "Transition" is bound correctly in the internal structure of Grafcet.*

Algorithms of check of an Action and a Transition-Condition: *Determine if the elements of type "Action" or "Transition-Condition" are bound correctly.*

We didn't judge useful to present these different algorithms in detail in this article, for reasons of space.

Remarks :

- During the implementation, we should insert in all these algorithms of check, except for the case of the first initial Step and Transition-Condition, the condition :

"*if Previous = NULL Then Result = Invalid component*"

because all components, except the first initial Step, possesses a Previous component.

In the same way, except for the final Step and the Actions, the condition (because all components possesses a following component) :

"*if Next = NULL Then Result = Invalid component*"

We will implement the test of uniqueness check when it is necessary. In fact, a component of which a necessary attribute is absent, will be considered temporarily like *Valid* and won't be declared *Invalid* at the end of the construction if this attribute remains absent. Thus, an intermediate Step (not initial or final) not having an associated action is considered like *Valid*. If the user declares the *stop* of the description of his Grafcet to this moment, then this Step becomes *Invalid* and the "disjointed" Grafcet.

5.3 Algorithm of construction of the Grafcet

To enlarge the comfort of usage of the application, the construction will be an "*Assisted Construction*", because the user will be helped in his gait. Thus, when he will choose to begin the construction of a new Grafcet, a virgin page will be open on which will be placed a Step zero representing the first initial Step of the system. This first component is important in the schema of the Grafcet. Indeed, it is the indicator of beginning of the chain of the integrated components on the window of construction and being part of the Grafcet effectively. The components that are not aligned to this chain are all the same create, but thereafter they are not controled and remain so much unknown that one doesn't relocate them to connected them to a component of the chain. They are controled then as one wants to bind them assembly-line. Thus, in the window of construction, one will have the Grafcet in construction, and a set of components "scattered" that one will be able to add to the Grafcet to the moment and Right side up appropriate, either to suppress it once the construction of the Grafcet finished.

Every time that the user creates a new component, one of the elements “Left”, “Right”, “Next” are suggested by default and possibly linked to the one here to reduce the time. Thus, when you create a new Step, a Link on the Right toward an Action as well as a Transition are automatically created and associated to the created Step. If these suggestions don't satisfy you, you can always select these and suppress them to put yours instead.

The "*code*" of the recursive algorithm of actual construction of the Grafcet, is provided in [2] (it is called when the user enables the *File->New* command of the Application):

The selection, the unselection, the deletion and the displacement of the components will be assured by the "*Event-Listener*"(Java language [20]) linked to the component during his creation. The algorithm of construction uses implicitly this *Listener* which cannot be described here.

5.4 Test of the incoherences generated when one adds a component

It is important and necessary to examine all elements preceding a component until the first initial Step again (Step 0), when the one is created here. Indeed, the creation of a component yet valid can generate incoherence in another link very distant of this last in the Grafcet. We illustrate it by the example of the figure 8.

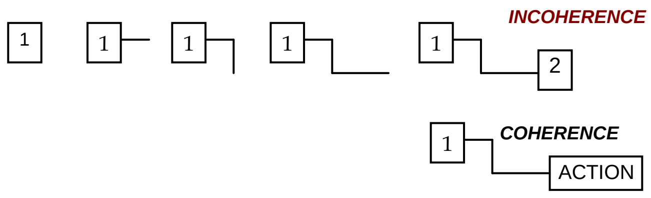


Fig. 8. Illustration of the necessity to control the components again after evry link

Comments on the figure 8:

Gone of a Step (1), one joins him successively, one after the other, the simple links. At the end, if one joins another Step, then two cases are possible:

- one considers that only the Step of departure becomes disjointed
- or then one considers that only this last Step is disjointed, all others components remaining valid.

It is why the creation of a new component must entail a new test of all his precedents until the Step initial zero of the Grafcet.

At the end of the construction, all components are examined again, one after the other for a bigger reliability. In fact, the reliability of the result of the verification is bound to the number of time that every component is examined during the construction of the Grafcet. More this number is big, more the Grafcet is coherent. It seems to enlarge the time of construction of the Grafcet, but one must weigh the equalization brought by the big reliability of the system thus gotten.

5.5 Algorithm of link of an isolated component in the Grafcet

Since in the window of drawing, one has the linked components being already part of the Grafcet in construction, as well as of the components even integrated to the Grafcet, and that will be him subsequently or will be suppressed then. The algorithm that we propose here (used by the algorithm of construction of the Grafcet) permits to bind one of these components in the Grafcet so possible therefore when the user wishes (either when he selects two components of which a non linked component, or then if he relocates a non linked component up to bind it physically to already linked component).

5.6 Algorithm of simulation of the Grafcets

In the Grafcet module, we have a general function (process) that permits to cover the different objects of the Grafcet, activating the Steps whose previous Transitions are validated and deactivating the Steps whose following Transitions are valid.

```
Function  parcours_recursif (Vecteur vectetape, Vecteur vectransition)
Debut
/* Vectors containing respectively active Steps and the valid transitions */
 Vecteur vecstapestampon, vectranstampon;
/* vector contains the Steps whose Previous transitions are validated */
Vecteur vectetape_suiv;
/* In this vector, one keep the active Steps  */
 vecstapestampon=recherche_etapes_actives(vectetape);
 /* In this vector, one stocks the transitions whose Previous Steps are active, this in order to enable them and possibly to validate them */
vectranstampon=recherche_transitions_etapes_actives(vectransition, vecstapestampon);
/* Stop condition of the recursive function. In fact, if it  has no transition to enable, it is that one achieved the end of a Grafcet */
Si (vectranstampon.taille()!=0)
debut
/* Enable the transitions whose Previous Steps are active */
 valider_toutes_transition (vectranstampon);
/* clears the valid transitions and whose associated receptiveness is true */
.franchir (vecstapes,vectetape,vectrans);
/* Deactivate the Steps whose following transitions are validated */
          desactiver (vecstapes,vectetape,vectransition);
  vectetape_suiv=recherche_etapes_a_activer(vectetape,vectrans);
  activer_etapes (vectetape_suiv,vectetape);
  .reinitialiser (vectranstampon,vectransition);
  parcours_recursif (vectetape,vectransition);
fin
sinon
/*deactivate  actives Steps when there are no more transitions to enable*/
  Desactiver (vecstapestampon,vectetape);
 Finsi
Fin
```

This routine will be called in the "*simulate*" method of the Grafcet, presented below:

```
Methode simulate (Vecteur vectetape,Vecteur vectransition)
{
   vect=parcours_etape_init(vectetape) ;
   activer_etapes_initiales(vectetape);
   parcours_recursif(vectetape,vectransition);
}
```

6 Implementation and Results

The simulation platform, referred to by its French acronym SAHY ("Simulateur des Automatismes HYbrides"), was modeling with UML and implemented in JAVA – a multithreading, portable and dynamic language [20]. The general plateform includes a simulator of Grafcet, a simulator of equations and a graphic interface of construction and description. During the implementation, close reference was made to the UML model base to ensure that the objects and object classes created reflected the required structure. The programming of the hybrid simulator was based on the use of *Thread* packages to create processes in JAVA. This technique enables multiple processes to be executed concurrently within the same environment. A process was implemented for the Grafcet simulator and another one for the equation simulator.

The simulation of the Sequential systems (Grafcets) consists in representing the evolution of the different Steps of the Grafcet progressively, while respecting the features of the parameters provided by the user, and the sequence of activation of its Steps. An active Step is highlighted by displaying it in color and placing a dot inside the step symbol. This Step will resume its normal color and the point will disappear, when it will be inactive.

Continuous systems are simulated by displaying graphs of output variables and tables of performance indices such as response time, steady-state accuracy and stability margin.

6.1 Logical Specification of attributes, events and processings of packages and classes of systems to construct

While coming closer maximally of the reality, the logical gait draws in our real environment while formalizing the representation of the introverted knowledge maximally. Our approche helps the programmer in his tasks of complete implementation. We were inspired by the used approach in [21].

To respect the full rigor of the Software engineering in order to put in place a progressive and maintenable system, we clearly specified in a textual way the set work while putting in evidence the events, the messages and the anticipated triggering. This important stage preceded the final implementation.

Several score of classes have been described. The reader will have a complete visibility of these project and the consequent experimentations while consulting the different articles there relative [2] [4] [5] [6] [7] [9] [10] [11] [12].

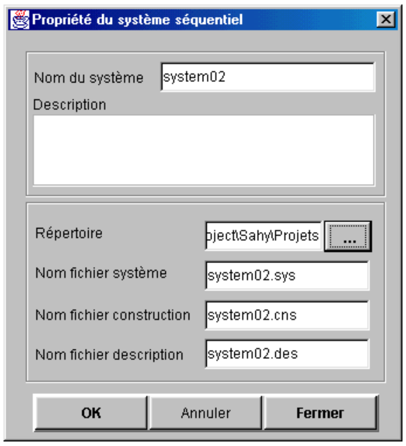
Due to space limitation, only the skeleton of code for the object class STEP is presented here.

```
Public class STEP
//Attributes
  Num : integer // step number
  x,y,cote : integer //screen co-ordinates
  etat : boolean // status of the step : active/inactive
  Description : String // description of function of the step
  Type : Boolean // TRUE for initial steps, FALSE for ordinary steps
  vec_act_assoc : Vector // List of Actions associated with the step.
//Methods
  public void Draw() //draws step.
  public void InputElement()
  public void OutputElement()
  public void Activate() // places a dot inside the step symbol.
  public void Disactivate()
  public void Zoom() // magnifies or reduces a transition by a specified
scale.
  public void move() // displaces a transition to a new position on the
screen.
  public Step find()
  public Vector copy() // copies an object of type step.
  public void modifyAttribute()
  public void compile() //checks if there are links before and after the tran-
sition. If this is not so, an error message is displayed.
End.
```

6.2 Application to the Construction and Description of the Grafcet of the Mill

The Rolling Mill is a typical industrial hybrid control system. The system converts metallic blocks to sheets. The Programmable Logic Controller (PLC) is the sequential sub-system. It implements the sequence logic for controlling the sequence of operations : a metallic block is heated to a specified temperature ; the gap between the rolls is adjusted to admit the block ; the block is inserted into the rolls ; the induction motor drives the rolls at constant speed until the roll gap is reduced to a specified value ; the sheet produced is removed from the rolls and the sequence repeats. The sequence logic is implemented in Grafcet.

The construction, the description and the simulation of the Mill have been performed completely. The figures 9, 10 and 11 describe the beginning of the parameterization of the Grafcet.


Propriété du système séquentiel

Nom du système	system02
Description	
Répertoire	oject\Sahy\Projets
Nom fichier système	system02.sys
Nom fichier construction	system02.cns
Nom fichier description	system02.des

Fig. 9. Sequential system parameter definition screen

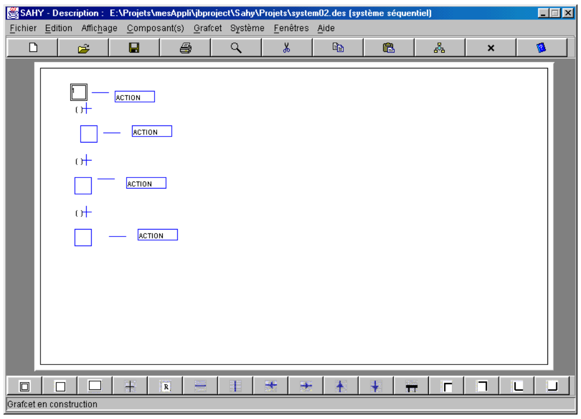
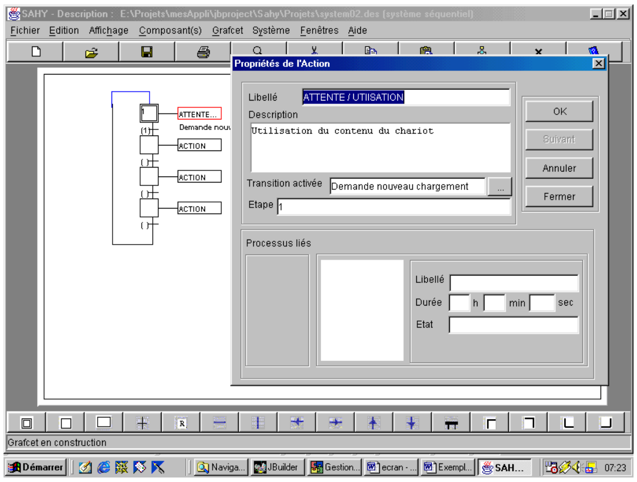

SAHY - Description : E:\Projets\mesAppli\jbproject\Sahy\Projets\system02.des (système séquentiel)

Fig. 10. Components placed in cascade during the construction


SAHY - Description : E:\Projets\mesAppli\jbproject\Sahy\Projets\system02.des (système séquentiel)

Propriétés de l'Action

Libellé: ATTENTE / UTIISATION

Description: Utilisation du contenu du chariot

Transition activée: Demande nouveau chargement

Etape: 1

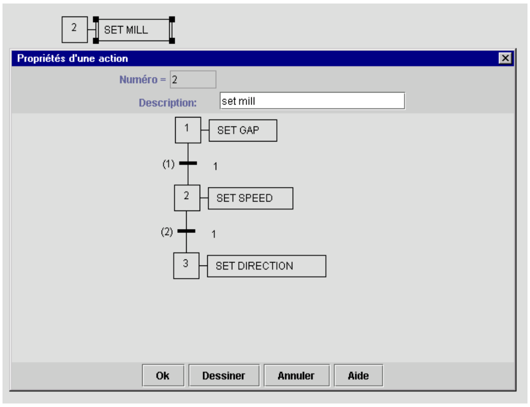
Processus liés

Libellé:
Durée: h min sec
Etat:

Grafcet en construction

Fig. 11. Parameterization of Action selected during the description

Figure 12 shows part of the Rolling Mill Grafcet. The screen used in describing interactions is shown in figure 13. The *Action-interaction* invoked (figure 13) calls the methods (functions) of reading of the parameters of the differential equation and of the order (it is depending of the type of chosen interaction).


Propriétés d'une action

Numéro = 2

Description: set mill

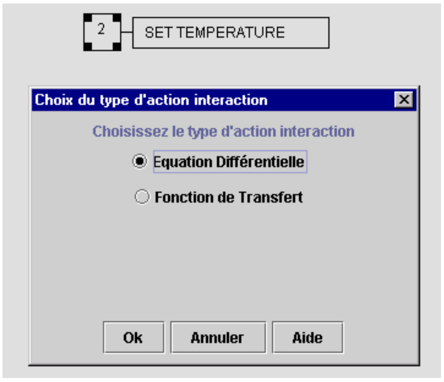
Fig. 12. Macro step parameter definition screen

6.3 Simulation of the Grafcet of the Mill

The Rolling Mill was simulated by creating icons for the various components of the system, designing the Grafcet of the system and defining the equations of the continuous part of the system.

To the departure, the continuous variables are fixed to their initial values (conditions), then the simulation is activated. We present in figure 14 the simulaton Grafcet of the evolution of the reversible lamination process.

It is necessary to note that every time one has synchronization between the evolution of the Grafcet and the one of the different curves to the various phases of the hybrid simulation. The technique of synchronization used here is explained in details in [2].

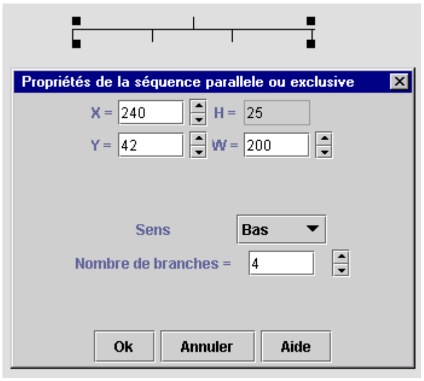

Choix du type d'action interaction

Choisissez le type d'action interaction

[✓ Equation Différentielle]

Fonction de Transfert

Parameterization of an Interaction-Action


Propriétés de la séquence parallele ou exclusive

X = 240 H = 25

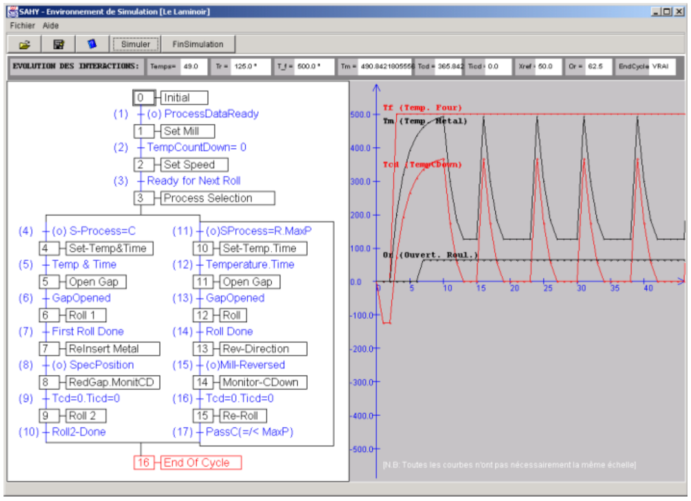
Y = 42 W = 200

Sens: Bas

Nombre de branches = 4

Parameterization of the Exclusive sequence Link

Fig. 13. Parameterization of the macro-actions of the Mill


SAHY - Environnement de Simulation [Le Laminoir]

EVOLUTION DES INTERACTIONS: Temps= 49.0 Tr = 125.0 * T_f = 500.0 * Tm = 490.8421805556 Tcd = 365.842 Ticd = 0.0 Xref = 50.0 Or = 62.5 EndCycle VRAI

Fig. 14. Simulation of a Reversible Lamination with 5 passages

7 Conclusion

An Object-oriented construction, description and simulation platform for Grafects has been presented. The platform is designed in UML and implemented in Java.

We conceived and achieved an algorithm taking in account the problem of the research of a consistency verification process, and we arrived to formulas and a "algorithmic method" of construction and the simulation of the Grafcet in an interactive way. The algorithms, the techniques, the models and programs presented have been applied to some sequential and hybrid system examples in order to put in evidence their efficiency and the advantages that one pulls of it. The "software" result of this project is a platform possessing various environments of functional "visual display" of hybrid "components", and takes counts the main constraints of quality defined by the standards of the software engineering ([22] [23] [24]) of it.

Further research will focus on two aspects of the project:
- An Extension of the simulator to include interactive compilation of the Grafcet during model construction.
- A survey more retailed of the semantics of the Grafcet construct.

REFERENCES

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

[2] M. Nkenlifack, "Modélisation objet et développement d'un atelier de simulation des automatismes industriels hybrides", Thèse de Doctorat de l'Ecole Polytechnique (Université de Yaoundé 1), Cameroun, 2004.

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

[4] E. Tanyi and M. Nkenlifack, "Une Adaptation d'UML à la Modélisation des Systèmes Hybrides", Revue des Sciences et Technologies de l'Automatique, Volume 7, N°2 - 2ème semestre 2010, pp 46-57, ISSN 1954-3522.

[5] M. Nkenlifack, E. Tanyi and F. Fokou, "Amelioration of the HAD Metamodel for the Modelling of Complex Hybrid Systems", International Journal of Advances Research in Computer Science, Volume 2, No. 1, Jan-Feb 2011, pp 370-380, available online at http://www.ijarcs.info, ISSN 0976 - 5697.

[6] M. Nkenlifack, E. Tanyi and F. Fokou, "Establishing bridges between UML, HAD and GRAFCET Metamodels for the Modelling of Dynamic Systems", International Journal of Scientific and Engineering Research, Volume 2, Issue 3, March 2011, pp 1 - 12, available online at http://www.ijser.org, ISSN 2229-5518.

[7] E. Tanyi and M. Nkenlifack, "Modélisation Unifiée Hybride et Simulation des Systèmes de Contrôle", Revue des Sciences et Technologies de l'Automatique, Volume 8, N°1 - Premier semestre 2011, pp 31-43, ISSN 1954-3522

[8] H. Brenier, Les spécifications fonctionnelles : automatismes industriels et temps réel, Dunod, France, 2001.

[9] M. Nkenlifack and E. Tanyi, "HAD: Extending UML for the Modeling of Hybrid Control Systems", Poster in ECOOP, 17th European Conference on Object-Oriented Programming, July 21-25, 2003, Darmstadt University of Technology, Germany, http://www.st.informatik.tu-darmstadt.de:8080/ecoop/posters/index.phtml

[10] E. Tanyi and M. Nkenlifack, "An object oriented simulation platform for hybrid control systems", Analysis and Design of Hybrid Systems (ADHS), 2003, Elsevier IFAC Publications, Edited by S. Engell, H. Gueguen & J. Zaytoon, ISBN 0-08-044094-0.

[11] E. Tanyi, T. Noulamo, M. Nkenlifack and J. Tsochounie, "A Multi-Agent Design and Implementation of An Internet Based Platform for the Remote Monitoring and Control of the Cameroon Power Network", Special Issue on Advances in Information Engineering, Engineering Letters, 13:2_18, ISSN: 1816-0948 (online version), 2006 http://www.engineeringletters.com/issues_v13/issue_2/index.html.

[12] E. Tanyi and M. Nkenlifack, "An extended UML for the modeling of hybrid control systems", in Burnham K. J., Haas O. C. L. (Editors), Proc. of the sixteenth International Conference on Systems Engineering (ICSE2003), Coventry, UK, 9-11 September 2003, Vol. 2, pp. 681 - 686, ISBN 0-905949-91.

[13] F. Ayres, Théorie et applications du calcul différentiel et intégral, série Schaum, McGraw-Hill Inc., New York, 1979.

[14] H. Gueguen and M. Lefebvre, "A comparison of mixed specification formalisms", 4ème Conférence Internationale sur l'Automatisation des Processus Mixtes (ADPM'00), 2000, Dortmund, Allemagne.

[15] R. David and H. Alla, Du Grafcet aux réseaux de Pétri, Paris, Hermès, France, 1997.

[16] CEI-IEC (Commission Electrotechnique Internationale), "Grafcet specification language for sequential function charts", Norme Internationale IEC 60848, 2002.

[17] J. Zaytoon and V. Carré-Ménétrier, "Grafcet et graphe d'états : comportement, raffinement, vérification et validation", JESA Vol. 33 N° 7, PP 751-782, 1999.

[18] A. Manuel, P. Remelhe and S. Engell, "Structuring Discrete-Event Models in Modelica", 4ème Conférence Internationale sur l'Automatisation des Processus Mixtes (ADPM'00), 2000, Dortmund, Allemagne.

[19] OMG-Web, www.omg.org, Site de l'OMG contenant le manuel de référence UML 2.0, consulté en 2008.

[20] C. Delannoy, Programmer en Java, Eyrolles, France, 2001.

[21] M. Nkenlifack, "Spécifications orientées objets de circuits électroniques", Mémoire de fin de DEA, ENSP, Université de Yaoundé I, Cameroun, 1999.

[22] S. Schach, Practical Software Engineering, IRWIN, Boston, 1992.

[23] M. Otter, Qualité des logiciels, Les techniques de l'ingénieur, Doc H4028.

[24] P. Mosterman, « An Overview of Hybrid Simulation phenomena and their support by Simulation Packages », F.W. Vaandrager & J. H. van Schuppen (Eds.), Hybrid Systems: Computation and Control, Lecture Notes in Computer Science 1569, pp 165-177, 1999.