ABCM Symposium Series in Mechatronics - Vol. 2 - pp.691-698

A PETRI NET BASED PLATFORM FOR DISTRIBUTED MODELING AND SIMULATION OF PRODUCTIVE SYSTEMS

Fabrício Junqueira
University of São Paulo, Escola Politécnica, BRAZIL
fabri@usp.br

Emília Villani
Instituto Tecnológico de Aeronáutica, BRAZIL
evillani@ita.br

Paulo Eigi Miyagi
University of São Paulo, Escola Politécnica, BRAZIL
pemiyagi@usp.br

Abstract. *Currently with the increasing complexity of productive systems associated with the geographical dispersion of industries there are new requiriments for design tools. In this context the purpose of this work is to introduce a new platform for distributed modeling and analysis of productive systems. The platform is based on Petri net as a modeling formalism and on the label-ring protocol for managing the distributed simulation. The focus of this paper is on the platform communication algorithm. An example is presented in order to illustrate the proposal.*

Keywords*: Petri Net, Distributed Simulation, Productive Systems.*

1. Introduction

Global markets are more and more demanding the industrial organization evolution into decentralized units. However, the different industry units should collaborate and coordinate efforts, coping with distributed customers, operations, and suppliers. According to Zhan et al. (2003) this new pattern of relationships can be identified as a new production paradigm, named Dispersed Network Manufacturing (DNM). It consists of enterprises that are geographically dispersed but need to communicate and work concurrently with large amount of data of their factories, sub-contractors and suppliers. The structure of a DNM motivates the use of a distributed design approach, where teams geographically distributed in different locations collaborate to system specification and design of the production processes.

Furthermore, in a DNM, the material and information interchange among the enterprises requires high flexibility, which contributes for increasing system complexity. In order to avoid failure, faults and errors, the validation of the system behavior is imperative. However, the validation of these systems as a whole is not trivial and may be unfeasible from a computational point of view. A possible solution is to adopt a distributed validation approach, where the computational effort is divided among a set of computers connected through a network.

In this context this work proposes a new software platform for supporting the distributed design and validation of productive systems. The platform is organized as a set of modules that are geographically distributed and that communicate using a new high speed Internet (Internet 2). This work is being developed as part of the Brazilian Government Program TIDIA/KyaTera (FAPESP, 2005) which is connecting a number of research laboratories through an advanced high-speed optical network (a tera-bps network). The TIDIA/KyaTera network is been used as a testbed for research in different areas including the distributed design and simulation of manufacturing systems.

The proposed platform adopts Petri net as a modeling formalism. The object-oriented paradigm is incorporated into the modeling approach in order to deal with the complex and distributed nature of the system. The next section discusses the distributed simulation of Petri net. Following, Section 3 presents the modeling of productive systems using the proposed platform. Section 4 introduces the platform architecture, the distributed simulation algorithm and an example. Finally, Section 5 presents some conclusions.

2. Petri net and distributed simulation

According to Ho and Cao (1991), productive systems can be considered as a Discrete Event Dynamic System (DEDS). The dynamics of a DEDS is characterized by discrete state variables whose values are switched, asynchronously in time, according to the occurrence of events that are considered instantaneous (Miyagi, 1996; Moore and Brennan, 1996).

Among the DEDS modeling formalisms, Petri net are particularly suited for representing the inherent characteristics of productive systems such as concurrency, synchronism and parallelism. In addition, the Petri net formalism is known for its solid theoretical base, clear syntax and semantics and intuitive graphic representation (Murata, 1989).

In the field of Petri net distributed simulation, two main solutions are commonly adopted: parallel computers and computer network. In the first case, the computation effort for simulating a complex model is distributed in a multi-processor environment. However, the distributed nature of the system is transparent to users. From the user point of view, the simulation is performed as in a non-distributed system. The main drawbacks of this solution are higher costs and a centralized modeling approach. Examples of works that adopt this solution are Fujimoto (1990), Nicol and Roy (1991), Chiola and Ferscha (1993), Kumar and Kohli (1997), and Beraldi and Nigro (1999). Chiola and Ferscha (1993) and Djemame et al. (1998) propose the creation and maintenance of a global list of enabled ***transitions***[1]. The list is sorted by its enabling time and divided among the different processors. Each processor then manages its own local list. The model division may result in conflicts, such as when a ***place*** is the pre or post condition of two or more ***transitions*** allocated in different processors. In this case the processors exchange a set of messages to solve the conflict preserving the causality among events. A different approach is proposed in Fujimoto (1990). In this case, the simulation algorithm is optimized according to the characteristics of the model. However, it cannot be successfully applied to flexible productive systems because the diversity of the models that will be simulated in the platform is not known beforehand.

With the advances in the computing and network technologies, the second solution became increasingly attractive. It is used by Nketsa and Vallete (2001), among others. One of its main advantages is the possibility of using computers geographically dispersed. This is the solution adopted in this work.

One of the main problems of Petri net distributed simulation is how to manage the simulation in order to guarantee consistent and coherent results. The solutions proposed in the literature can be grouped in two classes: conservative approaches and optimistic approaches.

The conservative approaches prevent causality errors by blocking the ***transition firings*** that are considered unsafe, i.e., ***firings*** that can lead to a situation where causality constrains are violated (Beraldi and Nigro, 1999). As a result all the nodes (processor or computer) must share a common simulation clock. The events are sorted by date and the simulation process become sequential.

On the other hand, the optimistic approach allows a certain degree of unsafe ***firings*** and each node has a ´Local Virtual Clock´. When a causality constrain is violated, the simulation must rollback the last operations until a safe state is reached. The simulation then restarts from this point. One of the implications of this approach is that each node must maintain a register of the last operations. In order to manage the size of the register, a ´Global Virtual Clock´ indicates the time of last safe state and limits the memory used by the algorithm. This approach is usually used in parallel computers with shared memory, which exploit the parallelism inherent to the models when compared to the conservative protocol. However, a great part of the computational effort may be lost in the rollback operations.

When a network of computers is employed, it results in no memory sharing. Moreover, the possibility of having rollback operations associated with the messages that are normally exchanged among computers can result in network traffic overload.

Another important drawback of optimistic approaches is that users cannot visually follow the evolution of the net marking during the simulation because of the possibility of rolling back. They can only see the final result. This is a hard limitation for system validation, as it makes difficult the understanding of system dynamics and makes impossible error debugging.

Based on these characteristics a conservative approach has been chosen.

3. Modeling approach

The global list of ***transitions*** previously described and proposed by Chiola and Ferscha (1993) and Djemame et al. (1998) can be used in parallel computers with shared memory. However, in a distributed environment of simulation, or either, an environment where a set of computers is connected through a network to act as a single one, it is not possible to have shared memory. In this case, the implementation of algorithms that divide the model and distributes it among the processors is not a good choice from a computational point of view.

Moreover, considering that previous knowledge of the system components by the modeler is important and in order to achieve modularity, the proposed platform incorporates the Object Oriented (OO) paradigm (Booch et al., 1999) into the Petri net formalism.

The OO enables the model to be built using sub models. Another advantage of the OO is the identification of sub models interactions. Thus, the interfaces among sub models are identified as well as the format of the messages changed between them. In other words, the use of the OO approach solves two problems in distributed simulation through computer networks: (1) how the model is divided among the computers – one sub model must be associated to only one computer while a computer can simulate one or more sub models; and (2) the rationalization of the information flow through the network because this flow is specified through interfaces previously defined by the modeler.

According to the proposed approach, the model of a distributed system is composed by a set of Objects. Following the OO paradigm, an Object is the instantiation of a Class. Classes represent generic elements of productive systems and can be organized into libraries, providing the re-use of models. A Petri net models the behavior of a Class.

[1] In this text, terms related with structural elements of Petri net is written in bold italic type.

Two Objects can communicate by method calls. In a method call, the exchange of messages between the two Objects is modeled by ***transition*** fusions (Sibertin-Blanc, 1993). An example is illustrated in Figure 1. The fusion of ***transitions*** $t_{A.1}$ and $t_{B.1}$ represents Object A calling a method provided by Object B. The fusion of ***transitions*** $t_{B.2}$ and $t_{A.2}$ models the answer to the method call provided by Object B to Object A.

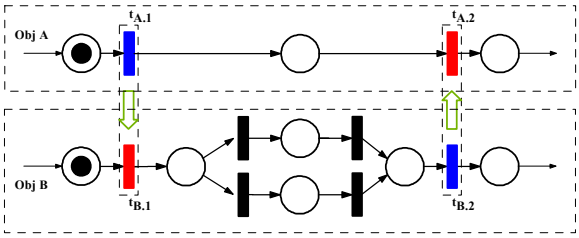
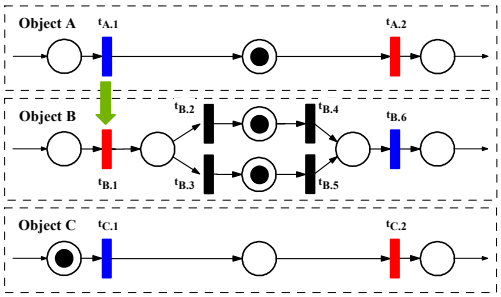
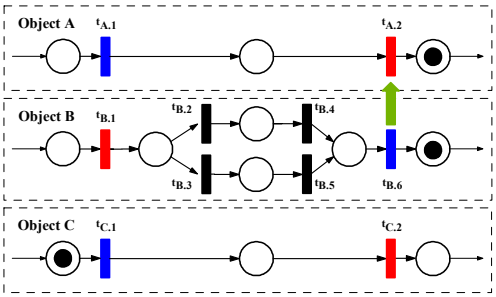


Figure 1. Method call between two objects.

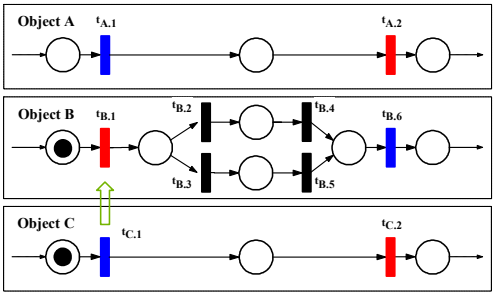
The method calls must obey the following rule. Once an Object made a method call to a second Object, it must wait for the answer. If the second Object is currently executing the method call of a third Object, it will add the request to a request list and execute it as soon as possible. An example is illustrated in Figure 2. In a) Object C sends a method call to Object B, but Object B is currently executing the method requested by Object A. Following, b) shows the three Objects after the answer of Object B to Object A. In c) the method of Object B is available again and in d) Object B is executing the method of Object C.



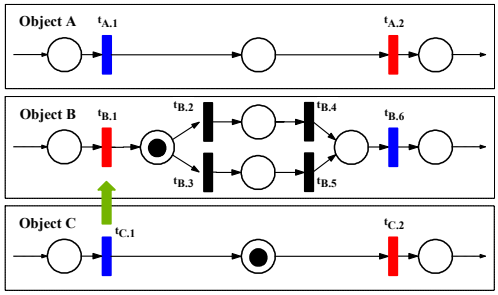
(a) Object B is executing a method required by Object A, through the fusion of transitions $t_{B.1}$ and $t_{A.1}$, while Object C is waiting for the availability of Object B



(b) Object B answers the method called through the fusion of transitions $t_{A.2}$ and $t_{B.6}$



(c) The method of Object B ($t_{B.1}$) is available again

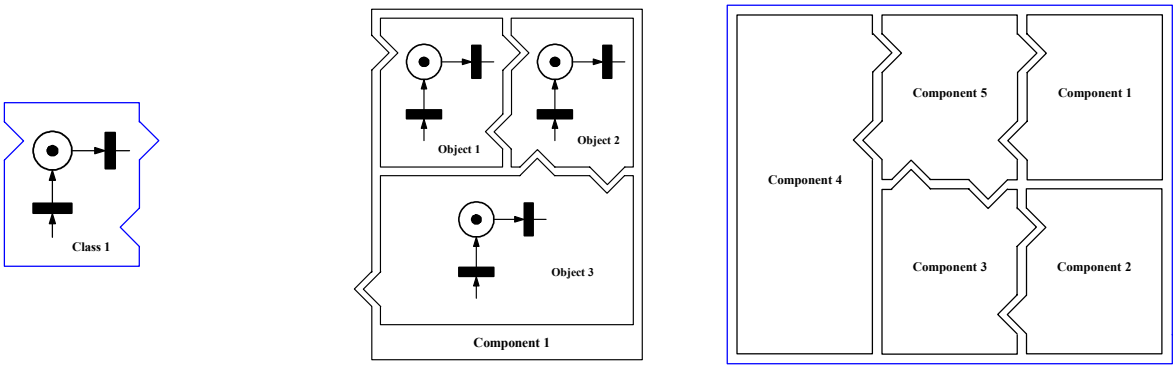


(d) Object C calls a method provided by Object B through the fusion of transitions $t_{C.1}$ and $t_{B.1}$

Figure 2. Two concurrent method calls.

In the Petri net this rule implies that the ***transitions*** that represent method calls cannot be in conflict with other ***transitions***. A second rule also determines that ***transitions*** associated to method calls are always instantaneous.

Models can be hierarchically organized in three levels: Objects can be grouped into Components, which can be organized in Applications (Figure 3).



(a) Class (b) Component (c) Application

Figure 3. A hierarchical way to organize models.

3.1. The use of a hierarchical approach – an example

In the level of Class, a Petri net represents general productive system elements such as AGVs and machines. Figure 4 presents a simplified model of an AGV. The ***place*** p_1 indicates the AGV in movement, while p_2 the (un) load process, and p_3 the waiting state. t_1 is an interface ***transition*** that signs the AGV to stop and to start the (un) load process. t_3 signs the (un) load process time, and t_2 the interface ***transition*** that signs the end of the process.

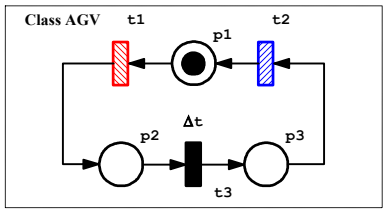


Figure 4. A simplified model of an AGV.

Based in nomenclature XML (W3C, 2005), Figure 5 shows a textual description of the model presented in Figure 4 (some tags had been suppressed to simplify the presentation and to detach the main points).

The tag <Var> used with t_3 textual description enables the Component "ManufactureCell" (Figure 6) to overwrite the t_3 default value (5 time units) with another value (for example, 10 time units). This feature assures more flexibility to reuse models.

```
<Class>AGV
   <PN>Place-Transition</PN>
   <Token>
      <Name>State
         <Attr>IP
            <Type>String</Type>
         </Attr>
         <Attr>IDCaller
            <Type>String</Type>
         </Attr>
         <Attr>Return
            <Type>String</Type>
         </Attr>
      </Name>
   </Token>
   <Place>
      <Name>L1
         <Desc>AGV running</Desc>
         <Tok>1
            <TName>State
               <Attr>IP
                  <Value><Var>Null</Var></Value>
               </Attr>
               <Attr>IDCaller
                  <Value><Var>Null</Var></Value>
               </Attr>
               <Attr>Return
                  <Value><Var>Null</Var></Value>
               </Attr>
            </TName>
         </Tok>
      </Name>
      ...
   </Place>

   <Transition>
      <Name>T1
         <Desc>Input transition</Desc>
         <Type>input</Type>
         <Rule>IP=param[1]</Rule>
         <Rule>IDCaller=param[2]</Rule>
         <Rule>Return=param[3]</Rule>
      </Name>
      <Name>T2
         <Desc>Outpu interface</Desc>
         <Type>output</Type>
      </Name>
      <Name>T3
         <Desc>Load or unload time</Desc>
         <Type>timed</Type>
         <Time><Var>5</Var></Time>
      </Name>
   </Transition>

   <Arc>
      ...
   </Arc>
</Class>
```

Figure 5. AGV model textual description.

```
<Component>ManufactureCell
   <Obj>
      ...
      <Name>Transp
         <Class>AGV</Class>
         <Transition>T3
            <Time>10</Time>
         </Transition>
            </Name>
         </Obj>

         <Arc>
            ...
         </Arc>
      </Component>
```

Figure 6. Component ManufactureCell textual description.

4. Proposed platform for distributed simulation

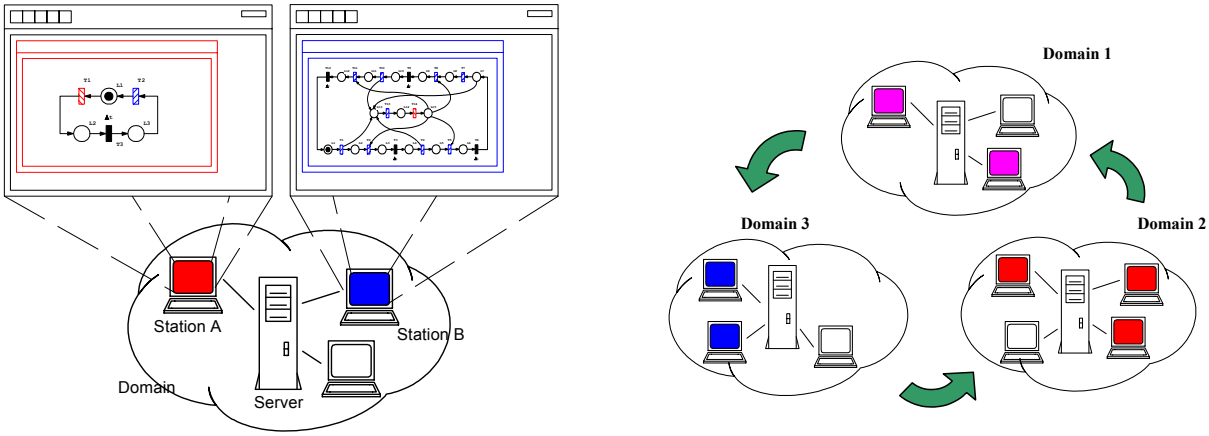
In order to support the distributed design of distributed systems, the platform must be itself a distributed system. From the software point of view, the platform architecture is based on the definition of Stations, Domains and Federations.

A Station is the modeling and simulation environment of a node (a computer) of the platform. A Station can edit and simulate models of one or more Components organized in one or more Applications.

A Domain is a group of one or more Stations (Figure 7(a)). One of these Stations is the Domain Server, which is responsible for managing Domain users and administrators, maintaining a list of on-line Stations, setting parameters of the communication protocol, among other tasks.

A Federation is a set of two or more Domains that can share Components. In a Federation, the communication among Stations of two different Domains is established through the Server of each Domain (Figure 7(b)).

It is important to observe that the distributed nature of the platform support not only distributed simulation but also distributed modeling. Among the results of the system distributed nature are (1) more complex models can be handled by the user and by the platform, (2) different groups of designers can work simultaneously on the same system, (3) different types of Petri net can be used by different designers according to the experience and preference of each one – the only restriction is upon the object interface.



(a) Objects of two different Stations interacting in a Domain

(b) Domains interacting in a Federation

Figure 7. Distributed simulation platform

4.1. Communication algorithm

This algorithm is base on the Label-Ring protocol (Göhring and Kauffels, 1994). In a Label-Ring network Stations are connected in a ring topology. However, in the proposed platform, Stations are not necessarily connected in a ring topology, but a logical ring can be built. Basically, each station knows the identity (IP address) of the next Station, and a label is sent across the Stations in a predefined order. Simultaneously messages associated to method calls can be sent across the net (Figure 8).

The management of the simulation is distributed among the Stations. The label synchronizes the evolution of simulation time in all the Stations, according to a conservative approach. The label frame is composed by 5 fields that can be modified by any of the Stations:

1. *Station identity field* – this field indicates the last Station to change the value of the other label fields.
2. *Future time field* – this field contains the simulation time required by the Station indicated in the Station identity field.
3. *Status field* – this field indicates the current status of the Station indicated in the Station identity field. Table 1 indicates its possible values.

4. *Instruction field* – this field carries instructions to all the Stations, such as start, stop and pause the simulation.

5. *Error field* – this field is used to manage simulation errors, such as when one of the Stations is disconnect because of a communication problem in the network.

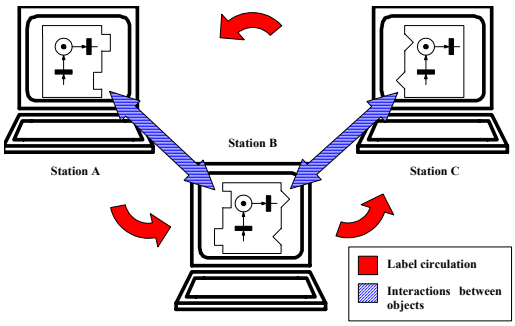


Figure 8. Communication among Objects.

Table 1. Status field

Value of the Status field	Meaning
0	--- (no station is using the label)
1	Station is checking the current status of the other Stations
2	Station is sending an order to all the Station to update the global time according to the Future time field.
3	Station is currently in a deadlock state.

The rules presented in Table 2 manage the simulation across the network.

Table 2. Management rules

1. When the simulation starts the label fields are reset (all the field are empty or zero).
2. The Server sets the *instruction field* to 'start' and send the label across the ring.
3. When the Server receives the label back, it makes the following changes:
   - *Instruction field* = empty.
4. After receiving the label with *Instruction field* = start, each station fires its enabled ***instantaneous transitions*** and exchanging method calls until:
   a. No more ***transitions*** are enabled.
   b. Only timed-***transitions*** are enabled.
5. If a Station is in the situation a) and receives the label with the reset values, it makes the following changes:
   - *Station identity field* = name of the station.
   - *Status field* = 3.
6. If a Station is in the situation b) and receives the label with the reset values, it makes the following changes:
   - *Station identity field* = name of the station.
   - *Status field* = 1.
   - *Future time field* = smallest time of ***firing*** of its ***enabled timed-transitions***.
7. If a Station is currently ***firing instantaneous transitions*** and receives the label with Status field = 1 or 3, it makes the following changes:
   - *Station identity field* = empty.
   - *Status field* = 0.
   - *Future time field* = current time of the object.
8. If a Station is in the situation a) and receives the label with *Status field* = 3, it does not change the label values.
9. If a Station is in the situation b) and receives the label with *Status field* = 3, it makes the following changes:
   - *Station identity field* = name of the Station.
   - *Status field* = 1.
   - *Future time field* = smallest time of ***firing*** of its ***enabled timed-transitions***.
10. If a Station is in the situation a) and receives the label with *Status field* = 1, it does not change the label values.
11. If a Station is in the situation b) and receives the label with *Status field* = 1, and the smallest time of ***firing*** of its ***enabled timed-transitions*** is bigger or equal to the *Future time field*, it does not change the label values.
12. If a Station is in the situation b) and receives the label with *Status field* = 1, and the smallest time of ***firing*** of its ***enabled timed-transitions*** is smaller than the *Future time field*, it makes the following changes:
   - *Station identity field* = name of the station.
   - *Status field* = 1.
   - *Future time value* = smallest time of ***firing*** of its ***enabled timed transitions***.
13. If, after setting *Status field*=3 the same Station receives the label without any changes, it means that all the Stations are in a deadlock state. The Station then sends a message to the Server to stop the simulation.
14. If, after setting *Status field*=1 the same Station receives the label without any change, it means that the Station has the ***timed-transition*** with the smallest time of ***firing***. The Station then makes the following changes:
   - *Station identity field* = name of the station.
   - *Status field* = 2.
15. If a Station receives the label with *Status field*=2, it sets its simulation clock according to the value of the *Future time field*. It does not change the label and it does not ***fire*** any of its ***enabled transitions***.
16. If, after setting *Status field* =2 the same Station receives the label without any changes, it means that all the Stations have update its simulation clock to the same global time. The Station then makes the following changes:
   - *Station identity field* = empty.
   - *Status field* = 0.
17. When a Station receives the label with *Status field*=0, it fires any ***timed-transition*** that has the current simulation clock as enabling time. It then ***fires*** its ***enabled instantaneous transitions*** and exchanging method calls until:
   a. No more ***transitions*** are ***enabled***.
   b. Only ***timed-transitions*** are ***enabled***.

4.2. Example

The model of Figure 9 is used as an example to illustrate the simulation algorithm. Despite its simplicity, it shows some of the key points of the algorithm such as the label passage and method calls.

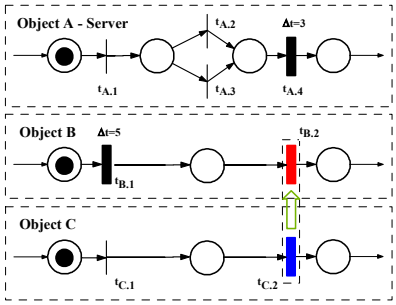
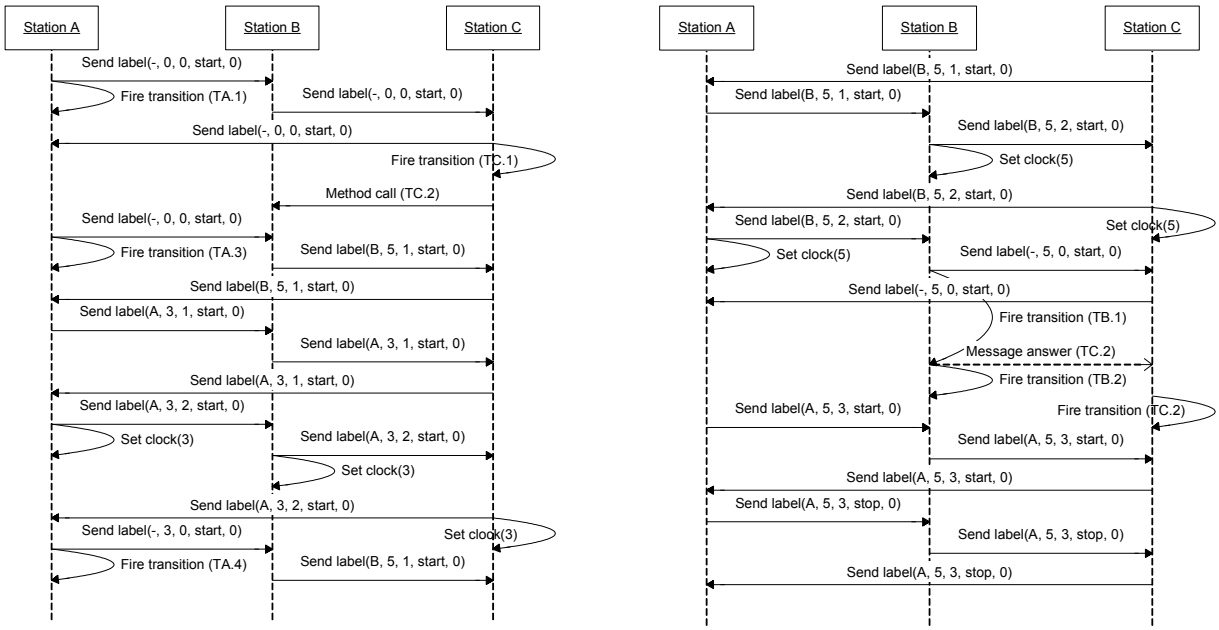


Figure 9. Petri net of the example.

In the example the system to be simulated is composed by three Objects (A, B and C), simulated in 3 different Station.

The main events of a possible simulation are illustrated in the UML Sequence Diagram of Figure 10. It is important to observe that the preliminary activities to compose the ring are not shown in the example, but it is also a task of the Server. Furthermore, time intervals such as the sending and receiving of the label, or the execution of ***transition firing*** subroutines are randomly chosen. As these time intervals depend on the computers capacity and on the availability of the communication media, they can vary from case to case.



(a) first part

(b) second part

Figure 10. UML Sequence Diagram for the example.

The simulation starts when the Server sends the label through the ring with ***Instruction Field*** = Start. After receiving the label each Station can ***fire*** its ***instantaneous transitions***, which are $t_{A.1}$ and $t_{A.2}$ (or $t_{A.3}$) for Station A, and $t_{C.1}$ for Station C. ***Transition*** $t_{C.2}$ is a method call to Object B, but this method is not yet available so Object C must wait.

Following, Object B makes a request to update the simulation clock to 5, however, before it receives the label back, Object A replaces the label fields with a request to time 3. When the label goes across the ring, all the Stations have already ***fired*** all their ***instantaneous enabled transitions***, so Station A confirm the update to 3 (***Status field*** =2).

After ***fire*** $t_{A.3}$, Station A reaches a deadlock state and sends the label with ***Status field*** = 3. This information is replaced by Station B that again requests to update the simulation clock to 5. This last request is not replaced, so the simulation clock is set to 5. Station B can then ***fire*** $t_{B.1}$ and then answer the method call, which results in the ***fire*** of $t_{B.2}$ and $t_{C.2}$.

This example, although simple, it illustrates the effectiveness of the communication algorithm, responsible for synchronizing the simulation in the diverse stations, exploring some of the rules presented in Table 2.

5. Conclusions

This paper proposes a new platform for the distributed modeling and simulation of productive systems based on the Petri net formalism. The main purpose of the platform is to support the distributed design of productive systems. The platform provides a design environment where teams of designers, engineers and other people involved in the design process can model and simulate the system from different geographical locations, using the high-speed internet as communication media.

The focus of this paper was on the simulation algorithm. Its main characteristics are the conservative approach and the label-ring based procedure for managing the simulation clocks.

The simplicity of the algorithm makes easy its extension to include high-level Petri net, which is the next step on the platform development. Another point that the authors are also considering is the adaptation of the platform to model hybrid systems, which is possible mainly because a conservative approach is adopted. The conservative approach imposes the same simulation time to all the Stations, allowing continuous variables sharing.

6. Acknowledgement

The authors would like to thank the partial financial support of the Brazilian governmental agencies CNPq, CAPES, and FAPESP, specially the TIDIA/KyaTera program.

7. References

Beraldi, R.; Nigro, L., 1999, "Distributed Simulation of Timed Petri Nets: A Modular Approach Using Actors and Time Warp" IEEE Concurrency, 7, no.4, 52-62.

Booch, G.; Rumbaugh, J.; Jacobson, I., 1999, The Unified Modeling Language User Guide. Addison Wesley Longman, Inc.

Chiola, G.; Ferscha, A., 1993, "A Distributed Discrete Event Simulation Framework for Timed Petri Net Models" Technical Report Series of the Austrian Center for Parallel Computation, ACPC/TR 93-21.

Djemame, K.; et al., 1998, "Performance comparison of high-level algebraic nets distributed simulation protocols", J. of Systems Architecture, No.44, 457-472.

Fujimoto, R., 1990, "Parallel Discrete Event Simulation" Communications of the ACM, 33, no.10, 30-53.

Göhring, H-G.; Kauffels, F-J.., 1994, Token Ring: principles, perspectives and strategies, Addison-Wesley Pub. Co.

Ho, Y.C.; Cao, X.R., 1991, Pertubation Analysis of Discrete Event Dynamic Systems, Kluwer Ac. Publishers.

Kumar, D.; Kohli, A., 1997, "Faster Simulation of Timed Petri Nets Via Distributed Simulation" In Proc. of the 21st Int. Computer Software and Applications Conf., 149-152.

Miyagi, P. E., 1996, Controle Programável - Fundamentos do Controle de Sistemas a Eventos Discretos, Ed. Edgard Blücher, São Paulo.

Moore, K. E.; Brennan, J. E., 1996, "Alpha/SIM Simulation Software Tutorial" In Proc. of the 1996 Winter Simulation Conference, 632-639.

Murata, T., 1989, "Petri Nets - Properties, Analysis and Applications" In Proc. of the IEEE, 77, no.4, 541-580.

Nicol, D. M.; Roy, S., 1991, "Parallel Simulation of Timed Petri-Nets" In Proc. of the 1991 Winter Simulation Conference, 574-583.

Nketsa, A., Valette, R., 2001, "Rapid and modular prototyping-based Petri nets and distributed simulation for manufacturing systems" Applied Mathematics and Computation, 120, 265-278.

Sibertin-Blanc, C., 1993, "A Client-Server Protocol for the Composition of Petri Nets" In Proc. 14th Int. Conf. on Application and Theory of Petri Nets, 377-396. Springer-Verlag.

FAPESP, 2005, "TIDIA-KyaTera – Advanced Internet Program", FAPESP, Brazil. http://www.kyatera.fapesp.br (access in 04/2005)

W3C, 2005, "Extensible Markup Language (XML)", World Wide Web Consortium, http://www.w3.org/XML/ (access in 05/2005)

Zhan, H. F. et al., 2003, "A web-based collaborative product design platform for dispersed network manufacturing" J. of Materials Processing Tech., no.138, 600–604.

8. Responsibility notice

The authors are the only responsible for the printed material included in this paper.