ABCM Symposium Series in Mechatronics - Vol. 1 - pp.525-531
Copyright © 2004 by ABCM

THE DEVELOPMENT OF A PLATFORM TO SIMULATE PRODUCTIVE SYSTEMS IN A DISTRIBUTED ENVIRONMENT

Fabrício Junqueira
Departamento de Engenharia Mecatrônica e Sistemas Mecânicos – Escola Politécnica da USP – Av. Prof. Mello Moraes, 2231, CEP 05508-900 – São Paulo - SP
fabri@usp.br

Emília Villani
Departamento de Engenharia Mecatrônica e Sistemas Mecânicos – Escola Politécnica da USP – Av. Prof. Mello Moraes, 2231, CEP 05508-900 – São Paulo - SP
evillani@usp.br

Paulo Eigi Miyagi
Departamento de Engenharia Mecatrônica e Sistemas Mecânicos – Escola Politécnica da USP – Av. Prof. Mello Moraes, 2231, CEP 05508-900 – São Paulo - SP
pemiyagi@usp.br

Abstract. *Pushed by the need of enlarging productive systems, the use of open distributed automation concept is spreading in industry. In distributed systems, some activities, such as the network data transmission may be seen as a succession of discrete states and instantaneous events. In this case, Petri Nets may be used to model this kind of system because it is known as a powerful tool used to model and to analysis concurrent, distributed, asynchronous, and parallel systems and it has a useful graphic representation and mathematic formalism. This work proposes the development of a platform to model and simulate productive systems, in a distributed computers environment. The adoption of middleware systems in this platform guaranties more portability, interoperability, scalability, and interconnectivity, allowing a future integration of this platform with real machines/resources. To help on the design of this platform, both ODP (Open and Distributed Processing) and UML (Unified Modeling Language) diagrams are used.*

Keywords: *Modeling, Simulation, Open-distributed Systems, Petri Net, Middleware System.*

1. Introduction

In productive industrial systems, equipments from different vendors are proprietary solutions, which implies in a low interoperability. Taking as an example the automotive industry, robots (as well as other resources) of different vendors have different communication protocols and programming languages. As a consequence it is necessary to develop special interfaces in order to be able to communicate and integrate all the parts, and coordinate their activities. Another characteristic of these systems is that they are composed by different components and equipments (ex.: automatic guide vehicles, machines, robots, etc.), whose behavior is partially independent. Each component or equipment was its own tasks and interacts with the other in order to achieve a global purpose. Therefore, productive industrial systems can be generally considered as a distributed system.

Furthermore, in other to cope with competition and globalization, productive industries must deal with complex strategies that involve the management of large automated systems, performing a number of simultaneous processes. The global production system adopted by FIAT as well as HP-COMPAQ is an example where the parts of their products are produced not only in different plants but also in different countries. These parts must be then exchanged in order to assembly the final product.

On the other hand, researches of computer area have introduced the concept of *open-distributed systems*, which are characterized by the possibility of using different operational systems, networks and computational equipments in heterogeneous applications (Katchabaw et al., 1999), and by the use of international norms to database operations, communication protocols, graphic interfaces, etc. (Kokai et al., 1998).

The design, development, and management of distributed applications present many difficult challenges that do not exist in traditional sequential systems (Katchabaw et al., 1999). Some of them, such as process distribution and heterogeneity (of processes, systems, equipments, etc.), can be handled by using access and location transparency offered by today's middleware systems, as well as distributed transaction processing and reliable messaging (Giese & Wirtz, 2001).

Based on the points cited before, a methodology and an environment to help the design and analysis (modeling, verification, and validation) of distributed systems is a need identified by all professionals in the area. In this context, this work proposes the development of a platform to model and simulate distributed productive systems, in a distributed environment. This platform is composed by a set of computers working in parallel. It can analysis and validate the models by simulation. The word "platform" is adopted here in order to distinguish the proposed "system" from other means that the word "system" could assume in the text.

The modeling of the productive systems is based on the Discrete Event Dynamic Systems (DEDS) theory. It includes not only the description of equipment behavior but also that of the industrial processes and control systems. The platform uses Petri Net as the modeling language. This choice is based on the Petri Net ability to represent concurrency, distribution, asynchronism and parallelism, and on its easily interpreted graphic representation (Moore & Brennan, 1996).

The concept of middleware is incorporated into the platform in order to guarantee portability, interoperability, scalability, and interconnectivity (Kokai et al., 1998). In the design of this platform, both ODP (Open and Distributed Processing) and UML (Unified Modeling Language) are used.

ODP (Open and Distributed Processing) is an approach for the development of distributed computational systems. It encompasses five distinct viewpoints, which allow different participants of the system development to observe it from different perspectives and levels of abstraction (Kandé et al., 1998; Boiten et al., 2000). These viewpoints guides the management of the information generated during the distributed system specification (Marte, 2000).

The UML diagrams are used in order to highlight different aspects of the system. Each diagram may be considered as a projection of the system or a part of it (Booch et al., 1999).

This paper is organized as following. Section 2 presents the platform specification and section 3 draws some conclusions.

2. The platform specification

The ODP viewpoints are: enterprise viewpoint, information viewpoint, computational viewpoint, engineering viewpoint and technologic viewpoint. The platform is presented in the sequence, through the specification of each of these viewpoints.

2.1. “Enterprise viewpoint”

The ODP “enterprise viewpoint” is directed to the needs of the system users. The system is modeled in terms of the required functionality, the domains involved, the actors and their roles. For the specification of the platform “enterprise viewpoint” it is necessary:

?? To define the object community that satisfies the enterprise aim. In this case, there are two communities: (1) ADMINISTRATORS and (2) USERS;

?? To define the platform functionalities. The platform is for the modeling, performance analysis through simulation and control of distributed productive systems. In the case of simulation, a global clock synchronizes the activities. When the platform is used for control it runs in real time. For this purpose, it is desired that models of equipment and resources can be replaced by the correspondent real entities (as an example, substitute the robot model by the real entity robot) in a transparent way.

?? To identify the functions performed by the objects. In the ADMINISTRATOR community, the administrator is responsible for managing the platform, creating new user accounts, and giving permission or not to access models. In the USERS community, the user may build models, analyze them, design a control system and perform some basic operations such as open and save models (Fig. (1a)).

?? To define the domains and federations and their rules. In the platform specification, the domain is considered as a set of communities of ADMINISTRATORS and USERS whose main characteristic is the mutual utilization of models. The ODP considers federation as a community of domains. In our case, the information exchange between the domains of a federation (Fig. (1b)) is restricted (more details in the “information viewpoint”).

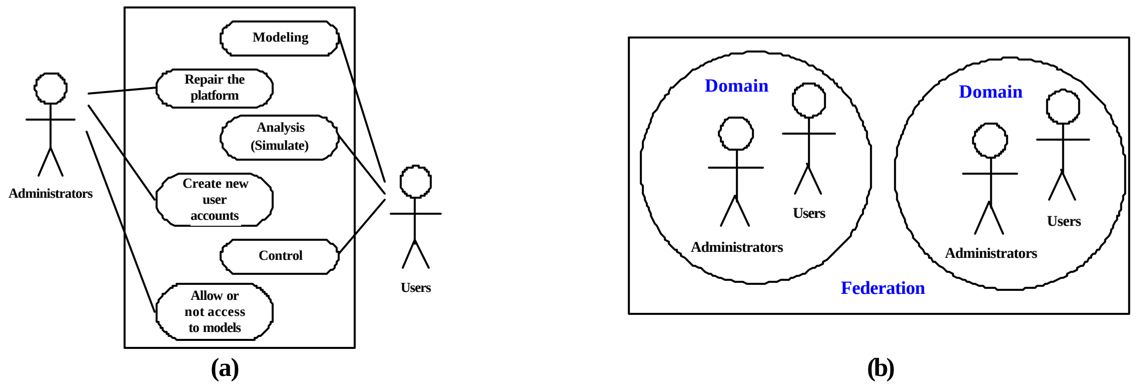


Figure 1 – (a) UML use case diagram of the proposed platform; (b) Illustration of the relationship between Federation and Domains.

2.2. “Information viewpoint”

The ODP “information viewpoint” describes a consistent, and common view of information resources, which supports information requirements of the “enterprise viewpoint”. It also defines relationships between information elements and information processes. For the platform “information viewpoint” specification it is necessary:

?? To identify the information objects and the associations among them. Using some UML terms, we define: (1) OBJECT – model of a real entity, with attributes, behaviors, and states that can accessed by other objects; (2) COMPONENT – the combination of one or more objects and/or components that possess some characteristics in common, and that interact among themselves; and (3) APPLICATIONS (it is not a UML term) – the combination of one or more components (Fig. 2).

?? To define association rules among the information objects: (1) a COMPONENT should be composed by one or more OBJECTS from a same domain; (2) a COMPONENT encapsulates objects that use the COMPONENT interfaces to communicate with other elements; and (3) an APPLICATION is composed by one or more COMPONENTS from the same or from different domains, but that belong to the same federation. This rule is based on UML concepts, and consequently on objects, where every interactions are performed by interfaces. Thus the COMPONENT implementation is hide. This rule is adopted as a way to: (1) limit the traffic of information though the platform; and (2) protect intellectual properties, and consequently have more people (enterprises, entities, and users) adopting this platform.

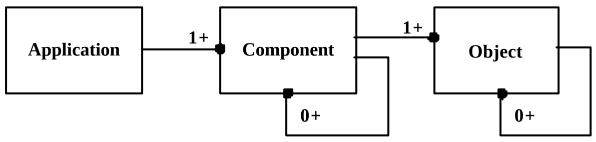


Figure 2 – UML class diagram for the association of information objects.

2.3. "Computational viewpoint"

The ODP "computational viewpoint" consists in the functional decomposition of a system into computational objects (CO), the definition of the object interfaces and the interactions among them.

The platform is composed by four computational objects: (1) a modeling and simulation environment (MSE); (2) a users manager (UM) that controls the access to the platform and models; (3) a data base (DB) where the models (object, component, and application) are stored; and (4) a communication manager (CM) that manages the communication between different MSE from the same domain, and changes information with UM from other domains. Thus, a domain is composed by a UM, a DB, a CM, and one or more MSE (Fig. (3)).

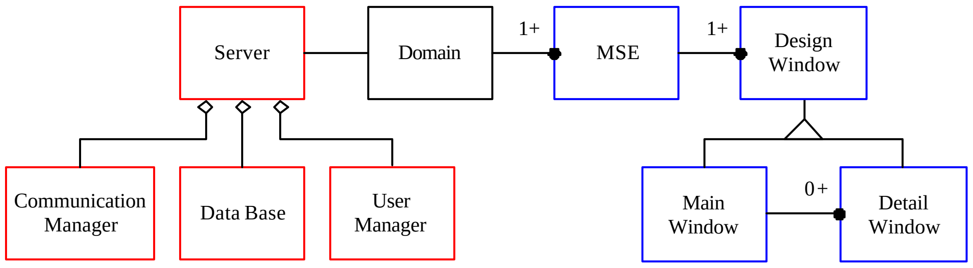


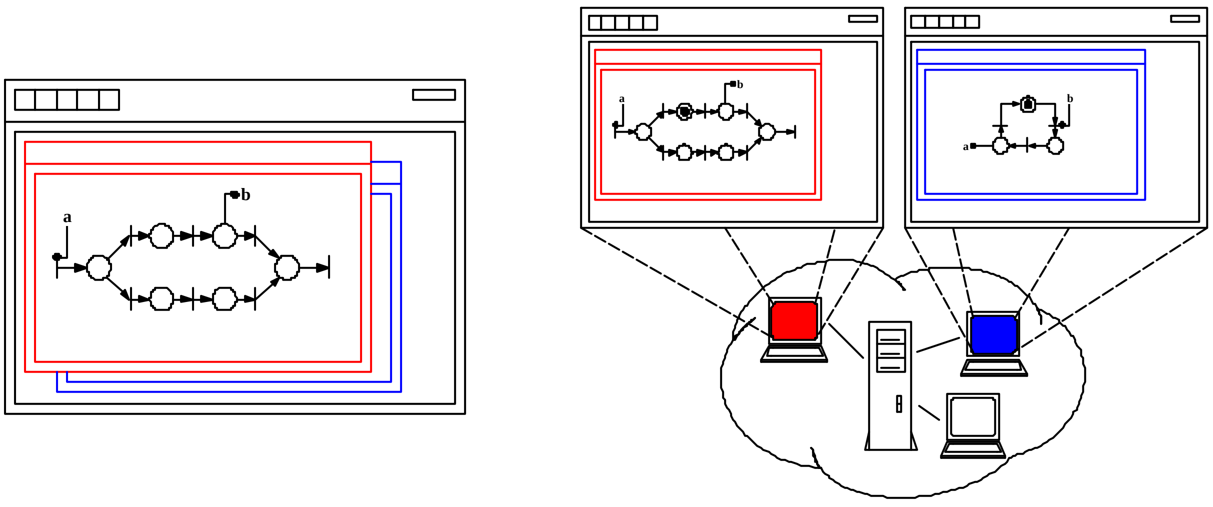
Figure 3 –UML class diagram of the proposed platform.

Each MSE has one or more "design windows", which uses a Petri Net as a modeling language. Each "design window" can be either a "main window" or a "detail window". This approach provides a top-down refinement as presented in Miyagi (1996). An object is designed in a "main window" and can be detailed in one or more "detail windows". During the simulation, the Petri Net model of each "main window" and its "detail windows" is treated as a single one because they belong to the same object.

The MSE, without others CO, can be used for designing of small models distributed in a set of MSE design windows (Fig. (4a)) – in this case, the MSE function coordinates the information exchanges among its windows. In the first case, during the simulation, the MSE manages the simulation clock. An example is when the red computer in Fig. (5) (and its models) is isolated from the other COs. Based on the dynamic of token-ring nets (Göhring & Kauffels, 1994; Bird, 1995), the MSE checks its "design window" scheduling and fires the transitions.

Combining the MSE with the other COs (UM, DB, and CM), the application can be distributed in more than one computer, dividing the computational effort (Fig. (4b)). In this case, the server synchronizes the simulation clocks of all the MSE that belong to the same simulation process (exemplified by the purple arrow in Fig. 5).

In order to synchronize the simulation, the server uses a "message" composed by two attributes: the "current step" and the "next step". When the "message" pass through a MSE, it uses the "current step" to fire Petri Net transitions. Each MSE writes its "next transition fire time" in the "next step" attribute when "next step" is null or "next transition fire time" is lower than "next step". When the "message" pass through the server, it copy the "next step" attribute to "current step" and reset "next step". Other attributes can be added to monitor network or simulation problems.



(a) (b)

Figure 4 – (a) An example of two (Petri Net) models interacting in the same MSE; (b) An example of two (Petri Net) models interacting each other in different MSE through a CM.

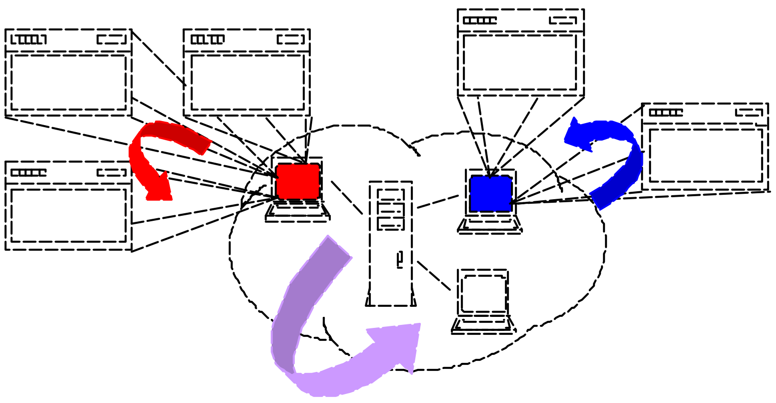


Figure 5 – An example of simulation clock synchronization using a solution based on token ring nets.

In the proposed platform, the computational model and its interfaces are defined as: (1) a Petri Net model decomposed in different "design windows" which has its interactions managed by a MSE; (2) a MSE interacts with another MSE (OBJECTS that belong to the same COMPONENT); (3) a MSE interacts with a CM (in interactions among COMPONENTS); and (4) a CM interacting with another CM (COMPONENTS from different domains interacting each other) (Fig. (6) and Fig. (7)).

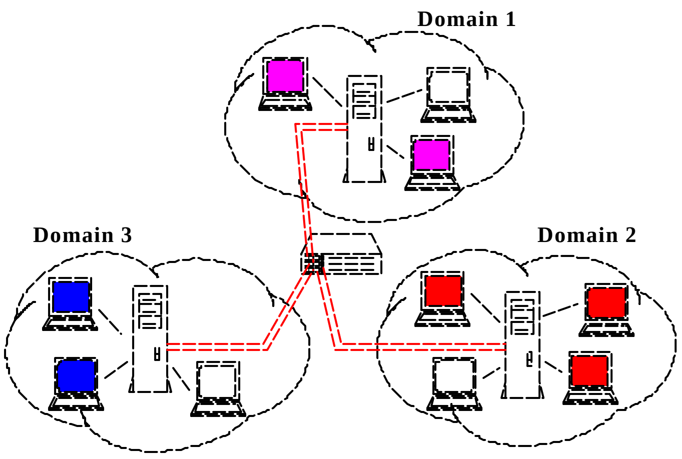


Figure 6 – An example of a project using components from three different domains, all of them from the same federation.

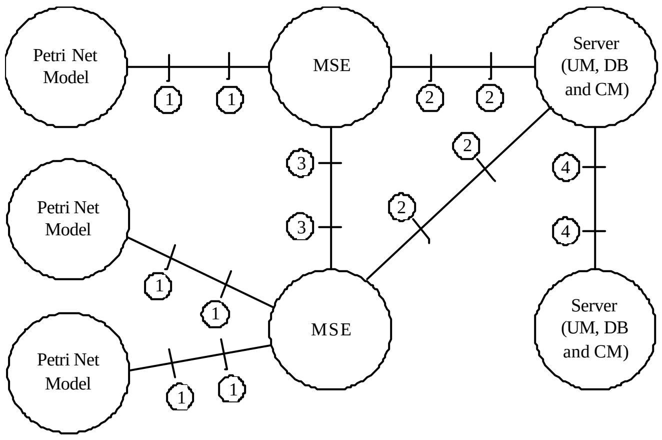


Figure 7 – Model of the platform computational objects (COs) and their interfaces.

The model of Fig. 7 presents the four different kinds of interfaces in the platform: (1) is responsible for starting and finishing the transactions, and for information exchange with other models and components; (2) is responsible for registering users at the domain, providing a list of models/components, and connecting with other models/components; (3) is responsible for exchanging data with other MSE; and (4) is responsible for registering users of other domains, providing a list of components and its interfaces, and connecting itself with other domains.

2.4. "Engineering viewpoint"

The ODP "engineering viewpoint" describes the infrastructure needed to support distribution, providing the transparencies. It defines the necessary communications and the deployment of functionalities. It complements the "computational viewpoint" setting some mechanisms and functions to support CO distributions.

The model of Fig. 8 represents the computational model of Fig. 7 from the engineering viewpoint. The notations 1 to 4 are the same from Fig. 7. The nucleus at Fig. 8 is the operational system. S, B, and P are respectively, Stubs, Binders, and Protocol objects that compose a channel. In a channel, stubs provide the conversion of data carried by interactions. The binders in a channel manage the end-to-end integrity of that channel. Protocol objects provide communication functions.

2.5. "Technology viewpoint"

The ODP "technology viewpoint" focuses on the details of the components (hardware and software) that will be used on this platform.

Middleware solutions such as CORBA (Common Object Request Broker Architecture) have been researched. CORBA is a standard adopted by OMG (Object Management Group), which specifies a system that provides interoperability between objects in a heterogeneous distributed environment (Saleh et al., 1999).

The object model introduced by CORBA is described as a group of cooperating objects that reside in the same machine or in different machines. The client objects can invoke methods of the server objects using its object reference, as it invokes the method of any other local object. Any request will be intercepted and executed by the ORB, which is responsible for locating the remote object and delivering the request to it. The server object processes the request and sends the result to the client again via the ORB.

In this object model, the client object does not need to know about the implementation of the server, what language the objects are written in, what operational system/hardware platform they run on, or what communication protocols and networks are used to interconnect distributed objects (Schmidt et al., 1998). The server provides an interface which allows the access to their functionalities. As a consequence the separation of interface and object implementation is evident.

DCOM and Javabeans (Chung et al., 1997; Hoffman, 2003; Payton, 2003; Raj, 2003) work as CORBA, but the first one is a proprietary solution from Microsoft Co, distributed with Windows operational system, but a portable version for other operational systems such as Linux is already under development. Javabeans is a solution developed by Sun Microsystems Co. that works with Java applications. Future versions of Javabeans will adopt some CORBA components to work with applications developed in other program languages.

The use of middleware architecture contributes to define a transparent way to exchange information among different systems such as other Petri Net simulators or real systems/resources (Fig. 9).

Other specifications such as programming languages and operational systems should be also considered once the aim is to adopt a programming language that can be used in different operational systems such as Delphi (and Kylix), C, and C++.

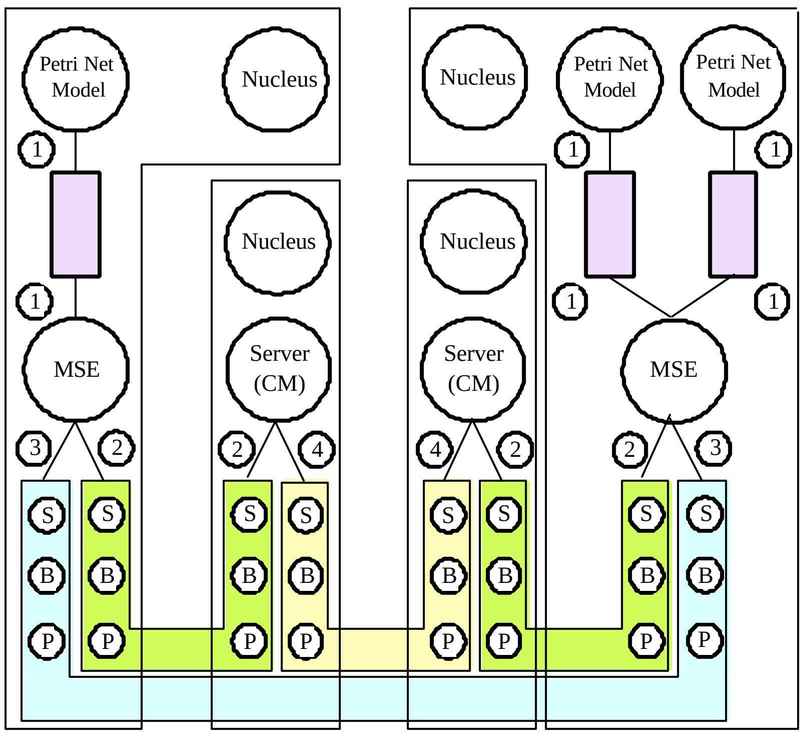


Figure 8 – Platform engineering model.

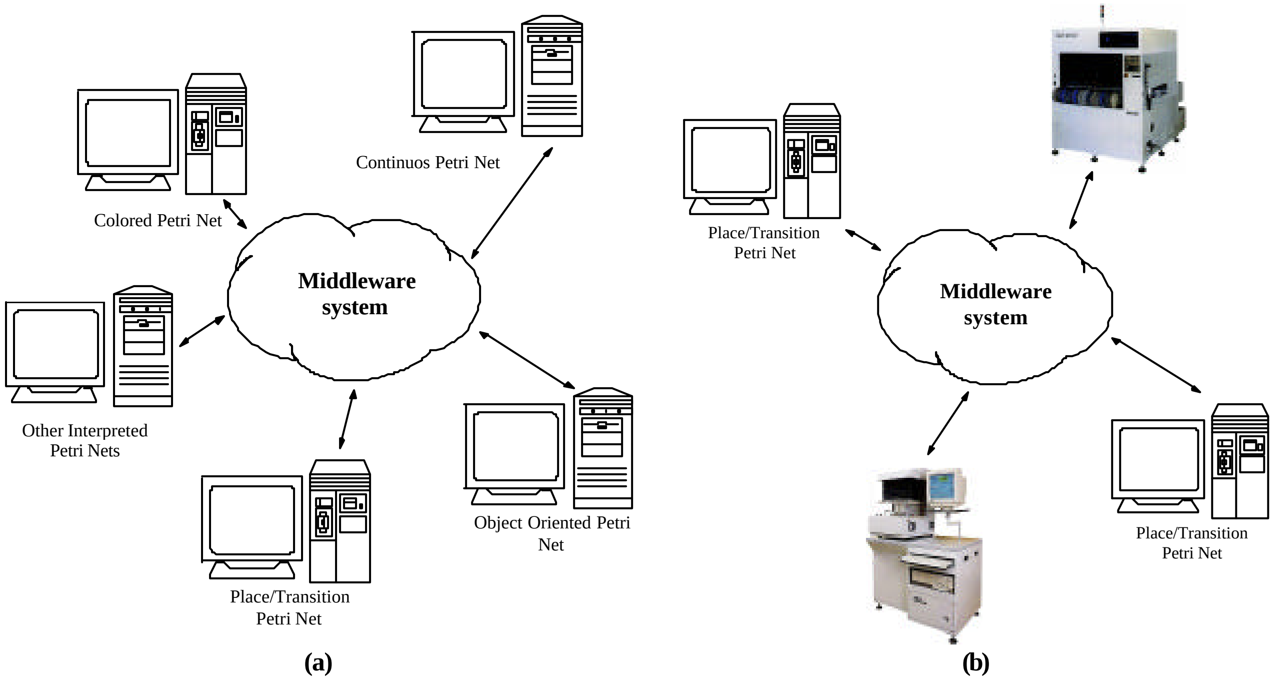

Colored Petri Net
Continuos Petri Net
Middleware system
Other Interpreted Petri Nets
Object Oriented Petri Net
Place/Transition Petri Net
(a)
Place/Transition Petri Net
Middleware system
Place/Transition Petri Net
(b)

Figure 9 – (a) A middleware system integrating different Petri Net simulators; (b) An automatized system integration using middleware system and Petri Net.

3. Conclusion

This work proposes the design and implementation of a platform to model and simulate distributed productive systems. Each part of the system can be modeled as an object and then grouped in components to form an application. Furthermore, the platform simulates the system in a computer-distributed environment that divides the computational effort.

An advantage of the use of ODP is that the system specification becomes independent from the technology solution. In other words, all the preview specifications (enterprise, information, computational and engineering viewpoint), in a great number of cases, do not need to be changed when a different technology is chosen.

The use of middleware system assures the computer distribution as well as the integration with other systems (commercial or not), improving the platform interoperability and interconnectivity. In manufacture systems, it transfers some hardware integration problems (e.g.: private communication protocols) to the software domain.

Petri Net is chosen as the modeling formalism of this platform due to its well-know advantages for representing discrete event dynamic systems. Furthermore, Petri Net based descriptions are used by different PLC (Programmable Logic Controller) vendors and in academic area, making possible the use of the platform to integrate systems of different vendors.

Currently, the platform is under development and the focus is on the implementation of the COs. Future works may include: (1) the development of a compiler that translates Petri Nets to a program language such as C; (2) the implementation of an agent to help on the load balance among the workstations; and (3) the implementation of routines for faults detection and treatment in the platform.

4. Acknowledgements

The authors would like to thank the partial financial support of the governmental agencies FAPESP, CNPq, and CAPES.

5. References

Bird, D., 1995, "Token Ring Network Design", Addison-Wesley Publishing Co.

Boiten, E. et al., 2000, "Viewpoint consistency in ODP", Computer Networks, No. 34, pp. 503-537.

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

Cardoso, J.; Valette, R., 1997, "Redes de Petri", Editora da UFSC, Florianópolis.

Cassandras, C. G., Strickland, S. G., 1992, "Sample Path Properties of Timed Discrete Event Systems in Discrete Event Dynamic Systems – Analyzing Complexity and Performance in the Modern World". New York, IEEE Press.

Chung, P. E. et al, 1997, "DCOM and CORBA Side by Side, Step by Step, and Layer by Layer", in: http://www.research.microsoft.com/~ymwagn/papers/HTML/DCOMnCORBA/S.html

Giese, H., Wirtz, G., 2001, "Visual Modeling of Object-Oriented Distributed Systems", Journal of Visual Languages and Computing, No. 12, pp. 183-202.

Göhring, Hans-Georg; Kauffels, Franz-Joachim, 1994, "Token Ring: principles, perspectives and strategies", Addison-Wesley Publishing Co.

Ho, Y.C.; Cao, X.R., 1991, "Perturbation Analysis of Discrete Event Dynamic Systems", Kluwer Academic Publishers.

Hoffman, R., 1999, "Sneaking up on CORBA: the race for the ideal distributed object model", in: http://www.networkcomputing.com/shared/printArticle.jhtml?article=/1009/1009f2full.html&pub=nwc

Inamasu, R.Y., 1995, "Modelo de FMS: Uma Plataforma para Simulação e Planejamento", Tese de Doutorado, Escola de Engenharia de São Carlos da USP, São Carlos.

Kandé, M. M. et al., 1998, "Applying UML to Design an Inter-Domain Service Management Application", UML'98: Beyond the Notation - International Workshop.

Katchabaw, M. J. et al., 1999, "Making distributed applications manageable through instrumentation", The Journal of Systems and Software, No. 45, pp. 81-97.

Kokai, Y. et al., 1998, "Recent development in open systems for EMS/SCADA", Electrical Power & Energy Systems, Vol.20, No. 2, pp. 111-123.

Marte, C. L., 2000, "Sistemas computacionais distribuídos aplicados em automação dos transportes", Tese de Doutorado, Escola Politécnica da USP, São Paulo.

Miyagi, P. E., 1996, "Controle Programável – Fundamentos do Controle de Sistemas a Eventos Discretos", Editora Edgard Blücher, Ltda, São Paulo.

Moore, K. E., Brennan, J. E., 1996, "Alpha/SIM Simulation Software Tutorial", Proceedings of the 1996 Winter Simulation Conference.

Payton, M., 2003, "CORBA vs. DCOM: a Comparison", in: http://www.cs.colorado.edu/~getrich/Classes/csci5817/Term_Papers/payton/

Raj, G. S., 2003, "A Detailed Comparison of CORBA, DCOM and Java/RMI, in: http://my.execpc.com/~gopalan/misc/compare.html

Reisig, W., 1992, "A Primer in Petri Design", Springer-Verlag, Berlin.

Saleh, K, Probert, R., Khanafer, H., 1999, "The distributed object computing paradigm: concepts and applications", The Journal of Systems and Software, No. 47, pp. 125-131.

Schmidt, D. C., Levine, D. L., Mungee, S., 1998, "The design of the TAO real-time object request broker", Computer Communications, No. 21, pp. 294-324.