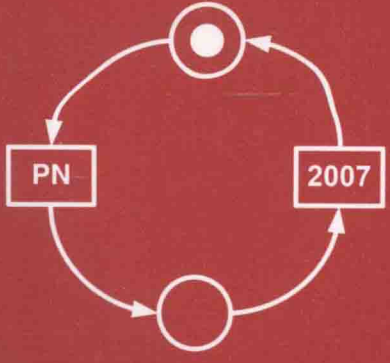
Jetty Kleijn
Alex Yakovlev (Eds.)

Petri Nets and Other Models of Concurrency – ICATPN 2007

LNCS 4546

28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN 2007 Siedlce, Poland, June 2007, Proceedings



Springer

Jetty Kleijn Alex Yakovlev (Eds.)

Petri Nets and Other Models of Concurrency – ICATPN 2007

28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN 2007
Siedlce, Poland, June 25-29, 2007
Proceedings

Springer

Volume Editors

Jetty Kleijn
Leiden University
Leiden Institute of Advanced Computer Science (LIACS)
P.O. Box 9512, 2300 RA Leiden, The Netherlands
E-mail: kleijn@liacs.nl

Alex Yakovlev
Newcastle University
School of Electrical, Electronic and Computer Engineering
Newcastle upon Tyne, NE1 7RU, UK
E-mail: Alex.Yakovlev@ncl.ac.uk

Library of Congress Control Number: 2007928856

CR Subject Classification (1998): F.1-3, C.1-2, G.2.2, D.2, D.4, J.4

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-540-73093-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-73093-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12077830 06/3180 5 4 3 2 1 0

Preface

This volume consists of the proceedings of the 28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007). The Petri Net conferences serve as annual meeting places to discuss the progress in the field of Petri nets and related models of concurrency. They provide a forum for researchers to present and discuss both applications and theoretical developments in this area. Novel tools and substantial enhancements to existing tools can also be presented. In addition, the conferences always welcome a range of invited talks that survey related domains, as well as satellite events such as tutorials and workshops. The 2007 conference had five invited speakers, two advanced tutorials, and five workshops. Detailed information about ICATPN 2007 and the related events can be found at http://atpn2007.ap.siedlce.pl/.

The ICATPN 2007 conference was organized by the Institute of Computer Science at the University of Podlasie and the Institute of Computer Science of the Polish Academy of Sciences. It took place in Siedlce, Poland, during June 25–29, 2007. We would like to express our deep thanks to the Organizing Committee, chaired by Wojciech Penczek, for the time and effort invested in the conference, and to Andrzej Barczak, for all his help with local organization. We are also grateful to the Offices of the Siedlce County Governor and Siedlce City Mayor for their suppport of the local organization.

This year we received 70 submissions from authors from 29 different countries. We thank all the authors who submitted papers. Each paper was reviewed by at least four referees. The Program Committee meeting took place in Leiden, The Netherlands, and was attended by 20 Program Committee members. At the meeting, 25 papers were selected, classified as: theory papers (17 accepted), application papers (5 accepted), and tool papers (3 accepted). We wish to thank the Program Committee members and other reviewers for their careful and timely evaluation of the submissions before the meeting. Special thanks are due to Martin Karusseit, University of Dortmund, for his friendly attitude and technical support with the Online Conference Service. Finally, we wish to express our gratitude to the five invited speakers, Samson Abramsky, Sadatoshi Kumagai, Antoni Mazurkiewicz, Andrzej Tarlecki, and Karsten Wolf, for their contribution to this volume. As usual, the Springer LNCS team provided high-quality support in the preparation of this volume.

April 2007

Jetty Kleijn
Alex Yakovlev

Organization

Steering Committee

Wil van der Aalst, The Netherlands
Jonathan Billington, Australia
Jörg Desel, Germany
Susanna Donatelli, Italy
Serge Haddad, France
Kurt Jensen, Denmark (Chair)
Jetty Kleijn, The Netherlands
Maciej Koutny, UK
Sadatoshi Kumagai, Japan
Tadao Murata, USA
Carl Adam Petri (Honorary Member)
Lucia Pomello, Italy
Wolfgang Reisig, Germany
Grzegorz Rozenberg, The Netherlands
Manuel Silva, Spain
Alex Yakovlev, UK

Organizing Committee

Wojciech Penczek (Chair)
Andrzej Barczak
Stefan Skulimowski
Artur Niewiadomski
Wojciech Nabiałek

Tool Demonstration

Wojciech Nabiałek (Chair)

Program Committee

Jonathan Billington, Australia

Didier Buchs, Switzerland
José-Manuel Colom, Spain
Raymond Devillers, Belgium
Susanna Donatelli, Italy
Jorge de Figueiredo, Brazil
Giuliana Franceschinis, Italy
Luis Gomes, Portugal
Boudewijn Haverkort, The Netherlands
Xudong He, USA
Kees van Hee, The Netherlands
Monika Heiner, Germany
Kunihiko Hiraishi, Japan
Claude Jard, France
Gabriel Juhás, Slovak Republic
Peter Kemper, USA
Victor Khomenko, UK
Jetty Kleijn,
The Netherlands (Co-chair)
Lars Kristensen, Denmark
Johan Lilius, Finland
Chuang Lin, China
Robert Lorenz, Germany
Patrice Moreaux, France
Wojciech Penczek, Poland
Laure Petrucci, France
Michele Pinna, Italy
Lucia Pomello, Italy
Laura Recalde, Spain
Toshimitsu Ushio, Japan
Rudiger Valk, Germany
François Vernadat, France
Karsten Wolf, Germany
Alex Yakovlev, UK (Co-chair)

Referees

Samy Abbes
Gonzalo Argote
Unai Arronategui
Eric Badouel
João Paulo Barros
Marco Beccuti
Marek A. Bednarczyk
Eric Benoit
Robin Bergenthum
Giuseppe Berio
Luca Bernardinello
Henrik Bohnenkamp
Andrzej Borzyszkowski
Anne Bouillard
Roberto Bruni
Nadia Busi
Lawrence Cabac
Javier Campos
Thomas Chatain
Ang Chen
Gianfranco Ciardo
Robert Clarisó
Lucia Cloth
Philippe Darondeau
Massimiliano De Pierro
Jörg Desel
Zhijiang Dong
Boudewijn van Dongen
Till Dörges
Michael Duvigneau
Dirk Fahland
Jean Fanchon
Carlo Ferigato
João M. Fernandes
Mamoun Filali
Paul Fleischer
Jana Flochova
Yujian Fu
Guy Gallasch
Pierre Ganty
Fernando Garcia-Vallés
Gilles Geeraerts
Attilio Giordana
Daniele Gunetti
Serge Haddad
Keijo Heljanko
Jarle Hulaas
David Hurzeler
Agata Janowska
Pawel Janowski
Jens Jørgensen
Michael Köhler
Kathrin Kaschner
Kii Katsu
Kais Klai
Nicolas Knaak
Maciej Koutny
Marta Koutny
Matthias Kuntz
Juan Pablo López-Grao
Linas Laibinis
Charles Lakos
Kai Lampka
Kristian Lassen
Fedor Lehocki
Nimrod Lilith
Niels Lohmann
Levi Lucio
Ricardo Machado
Kolja Markwardt
Peter Massuthe
Sebastian Mauser
Agathe Merceron
Toshiyuki Miyamoto
Daniel Moldt
Salmi Nabila
Apostolos Niaouris
Artur Niewiadomski
Olivia Oanea
Edward Ochmanski
Atsushi Ohta
Ribeiro Oscar
Chun Ouyang
Wieslaw Pawłowski
Luis Pedro
Florent Peres
Denis Poitrenaud
Agata Półrola
Heiko Rölke
Sylvain Rampacek
Jean-François Raskin
Ronny Richter
Matteo Risoldi
Nabila Salmi
Mark Schaefer
Carla Seatzu
Alexander Serebrenik
Dalton Serey
Frederic Servais
Natalia Sidorova
Markus Siegle
Christian Stahl
Martin Schwarick
Maciej Szreter
Shigemasa Takai
Satoshi Taoka
Yann Thierry-Mieg
Simon Tjell
Kohkichi Tsuji
Antti Valmari
Lionel Valet
Robert Valette
Laurent Van Begin
Somsak Vanit-Anunchai
Eric Verbeek
Valeria Vittorini
Daniela Weinberg
Lisa Wells
Matthias Wester-Ebbinghaus
Michael Westergaard
Dianxiang Xu
Shingo Yamaguchi
Satoshi Yamane
Huiqun Yu
Cong Yuan

Table of Contents

Invited Papers

Full Papers

Tool Papers

Petri Nets, Discrete Physics, and Distributed Quantum Computation

Samson Abramsky

Oxford University Computing Laboratory

Abstract. The genius, the success, and the limitation of process calculi is their *linguistic character*. This provides an ingenious way of studying processes, information flow, etc. without quite knowing, independently of the particular linguistic setting, what any of these notions are. One could try to say that they are implicitly defined by the calculus. But then the fact that there are so many calculi, potential and actual, does not leave us on very firm ground.

An important quality of Petri's conception of concurrency is that it *does* seek to determine fundamental concepts: causality, concurrency, process, etc. in a syntax-independent fashion. Another important point, which may originally have seemed merely eccentric, but now looks rather ahead of its time, is the extent to which Petri's thinking was explicitly influenced by physics (see e.g. [7]. As one example, note that K-density comes from one of Carnap's axiomatizations of relativity). To a large extent, and by design, Net Theory can be seen as a kind of *discrete physics*: **lines** are time-like causal flows, **cuts** are space-like regions, **process unfoldings** of a **marked net** are like the solution trajectories of a differential equation.

This acquires new significance today, when the consequences of the idea that "Information is physical" are being explored in the rapidly developing field of quantum informatics. (One feature conspicuously *lacking* in Petri Net theory is an account of the non-local information flows arising from entangled states, which play a key role in quantum informatics. Locality is so plausible to us — and yet, at a fundamental physical level, apparently so wrong!). Meanwhile, there are now some matching developments on the physics side, and a greatly increased interest in discrete models. As one example, the causal sets approach to discrete spacetime of Sorkin et al. [8] is very close in spirit to event structures.

My own recent work with Bob Coecke on a categorical axiomatics for Quantum Mechanics [4,5], adequate for modelling and reasoning about quantum information and computation, is strikingly close in the formal structures used to my earlier work on Interaction Categories [6] — which represented an attempt to find a more intrinsic, syntax-free formulation of concurrency theory; and on Geometry of Interaction [1], which can be seen as capturing a notion of interactive behaviour, in a mathematically rather robust form, which can be used to model the dynamics of logical proof theory and functional computation.

The categorical formulation of Quantum Mechanics admits a striking (and very useful) diagrammatic presentation, which suggests a link to

J. Kleijn and A. Yakovlev (Eds.): ICATPN 2007, LNCS 4546, pp. 1–2, 2007.

geometry — and indeed there are solid connections with some of the central ideas relating geometry and physics which have been so prominent in the mathematics of the past 20 years [3].

References

1. Abramsky, S.: Retracing some paths in process algebra. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 1–17. Springer, Heidelberg (1996)
2. Abramsky, S.: What are the fundamental structures of concurrency? We still don't know! In: Proceedings of the Workshop Essays on Algebraic Process Calculi (APC 25). Electronic Notes in Theoretical Computer Science, vol. 162, pp. 37–41 (2006)
3. Abramsky, S.: Temperley-Lieb algebra: from knot theory to logic and computation via quantum mechanics. To appear in Mathematics of Quantum Computation and Quantum Technology. Chen, G., Kauffman, L., Lomonaco, S. (eds.) Taylor and Francis, pp. 523–566 (2007)
4. Abramsky, S., Coecke, B.: A categorical semantics of quantum protocols. In: Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, pp. 415–425 (2004) arXiv:quant-ph/0402130.
5. Abramsky, S., Coecke, B.: Abstract physical traces. Theory and Applications of Categories 14, 111–124 (2005)
6. Abramsky, S., Gay, S.J., Nagarajan, R.: Interaction categories and foundations of typed concurrent programming. In: Deductive Program Design: Proceedings of the 1994 Marktoberdorf International Summer School. NATO ASI Series F, pp. 35–113. Springer, Heidelberg (1995)
7. Petri, C.-A.: State-Transition Structures in Physics and in Computation. International Journal of Theoretical Physics 21(12), 979–993 (1982)
8. Sorkin, R.: First Steps in Causal Sets. Available at `http://physics.syr.edu/~sorkin/some.papers/`

Autonomous Distributed System and Its Realization by Multi Agent Nets

Sadatoshi Kumagai and Toshiyuki Miyamoto

Department of Electrical, Electronic and Information Technologies
Osaka University
Suita, Osaka 565-0871, Japan
kumagai@eei.eng.osaka-u.ac.jp

Abstract. Autonomous Distributed Systems (ADS) concept plays a central role for designing, operating, and maintaining complex systems in these ubiquitously networked society. Design objectives such as optimality, reliability, and efficiency are renovated according to this system paradigm. Petri nets and its related models provide one of the most concrete basis for realizing ADS to cope with their nondeterministic, concurrent, and asynchronous behavioral features. On the other hand, autonomous decisions by distributed units based on their own interests should be coordinated with total system objectives. Multi Agent Nets are Object-Oriented Colored Petri Nets for implementing autonomous intelligent units and collaborating actions among distributed units. Here in this paper, the realization of ADS by Multi Agent Nets are described through several industrial applications and prototyping that shows paramount versatility of the approach to hardware-software distributed systems encountered in wide variety of engineering problems.

1 Introduction

The history of human beings can be seen as a struggle how to organize the society where individuals tend to enjoy freedom as much as possible. The innovation of systems for organizing such human society has been rather trial and error and there could not find any unilateral evolution in the process. It has been observed that the modern society had based its strength and efficiency on centralized governmental systems. However, from the beginning of 70's since the invention of micro processors, we can realize that all aspects of the society have been changed drastically such as globalization on one hand and diversity of individuals the other. The characteristics of post modern industrial society changed accordingly where agility, flexibility, and robustness are key of core competence. We can say these era as the third industrial revolution as Alvin Toffler correctly cited in his book, "The Third Wave", in 1980. The most significant impact of this innovation, especially on the industrial society, is the enforcement of individual operating units equipped with sufficient information acquisition and processing abilities. On the other hand, the organization extends its scale and complexity without bounds through information and communication

J. Kleijn and A. Yakovlev (Eds.): ICATPN 2007, LNCS 4546, pp. 3–19, 2007.

technologies (ICT) and rigid centralized control of the total system has been more and more difficult or even obstacle. In Japan, reflecting these tendencies, the intensive research project, called "Autonomous Distribution", funded by Ministry of Education began in late 80's. The author was one of the participants and took a role in prototyping the communication architecture and formal analysis of Autonomous Distributed Systems (ADS). Theoretical results such as structural characterization and reachability for live and bounded free-choice nets obtained in [1,2] are by-products of this project. Consecutively to this project, the international research project, called "Intelligent Manufacturing Systems", funded by Ministry of Trade and Commerce of Japan, has begun in 1992. In this project, we advocated Autonomous Distributed Manufacturing Systems (ADMS). ADMS are composed of variety of autonomous production units and these units can collaborate with each other through communication links to fulfill overall system tasks without any centralized control mechanism. The key features of ADS such as agility, flexibility, and robustness are emphasized in ADMS in addition to its scalable maintainability to cope with product variations of short life cycle in smaller quantities. As a modeling framework, we proposed Multi Agent Nets (MAN) which are implemented among a distributed environment for various applications. MAN is a nested Colored Petri Nets embedded in a Object-Oriented programming similar to other 0bject-0riented nets such as in [3,4]. Comparison of these modeling approach were made in [5]. Formal approach for nested Petri nets was originated by Valk and Moldt [6]. Here in this paper, the key concepts of ADS is summarized in Section 2. In Section 3, main features of MAN is explained. In Section 4, the software environment for MAN to be executed in distributed computers is described. In Section 5, we introduce several industrial applications of MAN to show the versatility of the approach to ADS realization. Section 6 is the conclusion.

2 Autonomous Distributed System Paradigm

ADS is a system composed of distributed autonomous agents (units) where each agent is provided with at least following six capabilities required in the sequence of decision and action process.

1. recognition of whole or partial state,
2. intent proposal based on self interest,
3. total evaluation of the expected results of the intention considering the proposals by other agents,
4. action based on the final decision,
5. timely communication with other agents,
6. learning for improving decision and collaboration strategies.

According to target applications, above functions are specified more precisely, but the fundamental feature of ADS is an iteration of step (2) and step (3) to reach the final consensus and decision that leads to the real action taken by individual agent in step (4) at each time point. Comparing with a centralized system, a drawback of ADS exists in this iteration process, and the efficiency of ADS depends on how quickly the consensus can be obtained among the participated agents.

In the evaluation step (3), each agent must take into account the total optimality and not just an individual interest. In some cases, the decision must be made against own interest. In this sense, we can call the ability in step (3) as a collaboration capability. In many applications, one or two iteration is enough to reach the final decision at each time point because of the simplicity of the negotiation mechanism. In other words, we only need collaborative agents with the common "world view or value" for the objectives of the problem. The learning functions in step (6) speed up the decision process avoiding negotiating iteration steps. Notwithstanding the above drawback, ADS has the obvious advantages such as (1) Flexibility, (2) Agility, and (3) Robustness. Flexibility includes the maintainability and scalability both in design and operation of the system. Frequent changes of the specifications or requirements of the system do not affect the basic architecture of ADS but only require the modification of the function, or simply the number of agent. We do not need to redesign the total system but just need to redesign the function of agent, or just to increase or decrease the number of agents according to the scale change. Behavior of ADS is inherently asynchronous and concurrent. Each agent can act based on its own decision, so agility is a key feature of ADS whereas the time controlling centralized systems cannot realize such autonomous behavior. Like as the scalability of ADS, plug-in and plug-out of agents are flexible so that a damage of a part of agents does not necessarily result in the total system down. The performability is maintained more easily by the remaining agents' efforts. It implies the Robustness of ADS whereas the centralized counterpart is usually quite vulnerable.

3 Multi Agent Nets

A multi agent net model was shown as a description model for ADS's in [7]. The objective of the model is to design a system, to simulate on the model, to analyze properties of the system, e.g., performance or dead-lock freeness, on the model, and to control the system by the nets. The model can be used from a design phase through the real time control with application oriented modifications. In ADS's, each system module has its own controller to decide own behavior. In the multi agent net model, a system is realized by a set of agent nets, and each agent net represents one component of the system. We call the component an agent, and its net representation as an agent net. Behavior of each agent is represented by the net structure of an agent net. Each agent net is an extended colored Petri net (CPN). Figure 1 is an example of the multi agent net. The figure shows a simple communication protocol. A sender sends a message and a receiver replies an acknowledge. When there are one sender and one receiver, the system is represented by a multi agent net with three agent nets. The agent net with CLASS protocol on their left shoulder is the system, and manage the number of senders and receivers and their interactions.

In the multi agent net, interaction between agents is activated by a communication among agent nets, i.e., a rendezvous of agent nets. The agent net with CLASS sender and receiver describe the behavior of sender and receiver, respectively. Agent nets

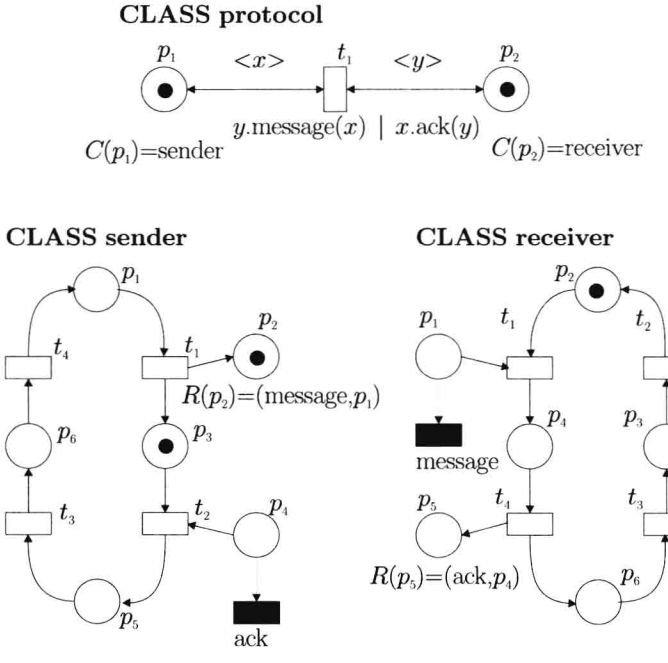


Fig. 1. A Multi Agent Net

belonging to the same agent class have the same net structure, and an agent net is called an instance of the agent class. Each instance has its own net and state, and instances are differentiated by their own ID number.

In a multi agent system, it is undesirable that a change in an agent needs changes in other agents to keep consistency among agents. In order to avoid the propagation, the multi agent net should be based on the object-oriented methodology like [3,4]. Thus, each agent net has two parts: an interface part and an implementation part, and other agent nets are allowed to access only the interface part [7]. This access restriction technique is well known as encapsulation. In the interface part, methods and their input places and output ports are specified on the agent net. Such a place is called an input port (of the method). The implementation part is a net structure without any methods and arcs to methods. A method represented by a bold bar is a fictitious node only expressing an action provided for the requirement of other agent nets. To execute the action, the agent must get tokens in the input places of the method. In Figure 1, a method is represented by a black-filled box, and arcs from input places are drown by dashed arrows. For example, the net "sender" provides a method ack, and the net "receiver" send an acknowledge message via this method. Note that the implementation part is encapsulated from outside. That is, designers of other agent nets can only see that sender provides a method ack and its input place is p4. By using this encapsulation technique, any side effects of changing agent net structure can be reduced or neglected. Note that ordering relations with respect to method call should

carefully be examined, since disregards of ordering of method call may cause the deadlock of the system when an agent net has plural methods. In the multi agent net model, each agent class has its own color, and it is allowed that an agent net resides in a place of another agent net as a token [7]. This extension allows us a hierarchical structures in Petri nets. We call it a recursive structure. In Figure 1, the agent net "sender" has its color "sender", and the color specified to place p1 in the agent net "protocol" is also "sender" specified by C(p1)=sender. That is the token in the place p1 in the net "protocol" is the net "sender". Actually, the token has a reference to the net. In this case, the net "protocol" is called an upper-level net, and "sender" is called a lower-level net. Note that there exists an agent net for each token, where the map need not be a bijection. Thus we can consider that the net structure of the lower-level net is one of attributes of the upper-level net, and its marking is the value of the attribute. One of the advantages of this extension is that treating a net as a token enables us to change the number of agents easily. That is, by changing the number of tokens in the agent net "protocol", we can change the number of senders and receivers arbitrarily. A multi agent net can provide reuses of design resources by inheritances and aggregations of classes [7]. When class B inherits class A, class A is called a super-class of class B, and class B is a sub-class of class A. An inheritance of a class implies the inheritance of its color, namely we can say that class B is class A. Therefore, when the classes have colors A and B respectively, a token is specified in a place p1 with C(p1)=A, whether the token implies a net in class A or class B. The reverse, however, is not true. That is, when C(p2)=B, a token that implies a net in class B can exist in p2, but a token that implies a net in class A can not exist. An inheritance and an aggregation of a class are done by copying nets of the super-class. Designers can change a part of the net when the part is copied from the super-class, so long as methods and ordering relations with respect to method calls are kept unchanged. Besides of these extensions including object-oriented methodology, we need to provide intelligence on agent nets by putting a program code on a place or a transition in addition to guards or arc expressions. Such a program code is called an action on a place or a transition. An action on a place is executed when a token is put into the place, and an action on a transition is executed when the transition fires. In order to make program more flexible, agent nets can hold attribute variables. For example the ID number of each agent net, which we mentioned in the previous paragraph, is one of such variables. We can use these variables anywhere in an arc expression, a guard on a transition or in an action. Intelligent decision of each agent can be expressed in these program codes. For the use of real time applications, two kinds of notions of time are introduced into the multi agent net model: a transition invalid time and a token invalid time. Each value is given by an integer number. The transition invalid time inhibits transition firing. A transition with this invalid time cannot fire during that period, once it fires. The token invalid time inhibits use of tokens. A transition can put this invalid time to its output token when it fires. The token cannot be used during that period. We assume that there is unique clock in the system, each transition fires according to the clock. When any invalid time is not given, a transition can fire with zero delay and it can fire again in the same clock if it is enabled, and the succeeding transition can fire with the token in the same clock.

4 Software Environment for the Multi Agent Nets

A software environment for the multi agent nets is shown in Fig. 2 [8]. It consists of three tools and class libraries as follows:

Net Editor: GUI for editing agent nets
Analyzer: It analyzes consistency between agent nets.
Composer: It composes a simulation system of agent nets. Its output is initial state of agent nets.
Class Libraries: Java class libraries for simulation

Java class libraries are improved considering following points:

- actions,
- invalid times, and
- distributed simulation.

The multi agent net must run on distributed computers which are connected on a network. Fig. 3 shows structure of the Class Libraries. In the figure, the dark box is Java class libraries. They are given by vendors. The gray boxes are libraries for the multi agent nets. The lower library is called Multi Agent Environment (MAE) library. It provides a multi agents platform and communication ability of agent nets on distributed computers. In this level, creating and destroying agents, distributing agents to hosts, naming service of agents and communications between agents are supported. The MAE consists of three major components: Agent, AgentBase, and Base-Master. Agent provides a form of agents. The Timer in Fig. 3 is an extension of Agent, and it counts clocks and distributes clocks to agents. Agent-Base provides server service for managing agents. There must be a unique base on each host. When an agent is created, it must be registered in the base on the same host. Base-Master provides server service for naming service of agents and managing bases. There must be a unique master on each simulation. It provides the same service of CORBA name server.

Currently communication between computers is realized by using Java RMI (Remote Method Invocation) on Agent-Base and Agent-Master levels. The higher library is called Multi Agent Net library. It provides component libraries for agent nets, e.g., transitions, places, arcs, tokens, and so on. Agents on the MAE can be agent nets by using these libraries. When some special work on transitions, places or arcs are required, we can put programs to them as "guards", "actions" or "arc expressions". The guard and the arc expression must be a boolean function. The action is executed when the transition fires or a token was putted into the place. In the program, we can use attributes of the net and attributes of the relating token. No other attributes can be used, and the program is written in Java. By calling native method in the action by JNI (Java Native Interface), we can control a real system. There may be a case where we want to use general agents in the multi agent system for which a net structure is not necessary to be defined. For example, if an agent only calculates a complex function, no agent nets need to correspond to the agent. These agents may be regular Java agents