A COMPUTATIONAL TOOL TO MODEL INTELLIGENT SYSTEMS

José A. S. Guerrero[a] Antônio S. R. Gomes[b] Ricardo R. Gudwin[c]
DCA-FEEC-UNICAMP
Caixa Postal 6101-13.083-970-Campinas, SP-Brasil
{jasg[a],asrgomes[b],gudwin[c]}@dca.fee.unicamp.br

Abstract: This paper presents the implementation of a computational tool for modeling intelligent systems under the paradigms of computational semiotics. It uses, as a modeling artifact, the framework given by object networks and mathematical objects (Gudwin 1996). Several aspects of the system dynamics, characteristic topology and selection mechanisms are tackled in detail, aiming at a robust architecture for object network simulation in real-world applications.

Keywords: Computational Semiotics, Modeling Intelligent System, Object Network.

1 INTRODUCTION

The role of Semiotics within intelligent systems development and implementation is being recently studied as an important foundation paradigm able to provide a future basis for a general theory of intelligent systems (Albus 1997).

Within many possible interactions between Semiotics and Intelligent Systems, we detach Computational Semiotics, i.e., the attempt of emulating the semiosis cycle within a digital computer. The main paradigm behind computational semiotics is "semiotic synthesis", i.e. the ability to use sign-processes and semiosis cycles from an engineering point of view, as a design platform for development of intelligent systems. Among other things, using computational semiotics paradigms we are targeting the construction of autonomous intelligent systems able to perform intelligent behavior, including perception, world modeling, value judgement, and behavior generation (Albus 1991). There is a claim that most part of intelligent behavior should be due to semiotic processing within autonomous systems, in the sense that an intelligent system should be comparable to a semiotic system. Mathematically modeling such semiotic systems is currently being the target for a group of researchers studying the interactions encountered between semiotics and intelligent systems.

The key issue in this study is the implementation of a computational tool in order to enable us to model intelligent systems and to study semiotic processes on computers. In this work we adopted a theoretic tool known as *object networks*.

Object networks were first introduced by Gudwin (Gudwin 1996), and have been used as formal background for the development of Computational Semiotics (Gudwin and Gomide 1997a, Gudwin and Gomide 1997b, Gudwin and Gomide Sep. 1997) in computer systems. More recently, object networks were also used within the context of computational intelligence, soft computing (Gudwin and Gomide Oct. 1997, Gudwin and Gomide 1998a), and computing with words (Gudwin and Gomide 1998b).

2 A COMPUTATIONAL MODEL FOR OBJECT NETWORKS

An object network is a mathematical tool structured in several sub-elements, such as objects, places, ports, relationships between places and the network operation mechanism itself. Its formal specifications and comparisons with other modeling tools can be found in (Gudwin, 1996). Other modeling tools using objects in Petri-nets-like structure can be seen e.g. in (Ceska & Janousek 1996, 1997), or in (Newman et. al. 1998)

2.1 Objects

Objects are the elements that define the dynamic behavior of the network. They are the fundamental actors in the process of creation, elimination and modification of information that is encoded through existing objects. There are two types of objects. The first, acts solely as information containers (figure 1), exhibiting *passive* behavior. The second type, known as *active*, also contains information as a passive object, but it also includes a set of internal transforming functions that operate over its internal data (figure 2). Actually, active objects are responsible for generation, destructive or non-destructive assimilation, and modification of its own state.

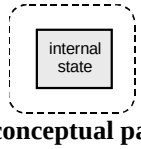

Figure 1. The conceptual passive object.

Interactions between objects are regulated by a mechanism called triggering, and are performed by active objects. This mechanism, its behavior, and its formal definition are described in depth in (Gudwin 1996, Gudwin and Gomide 1998a, Gudwin and Gomide 1998b, Gudwin and Gomide May 1998).

Every object is instantiated from a predefined class. A class represents a general description of how each object of this class should behave, what kind of internal data it has, and how its private ports are connected to its internal data.

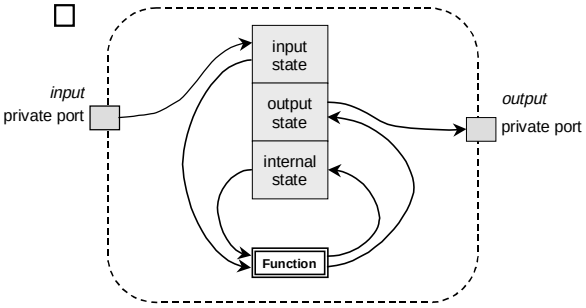

Figure 2. The conceptual active object.

2.2 Object Networks

An object network is a special type of object system (Gudwin 1996) in which additional constraints concerning interactions are included. To distinguish an object network from an object system let us assume places and arcs (links) whose roles are similar to those used in Petri nets context (Murata 1989). An object, in a given place, can only interact with objects in places connected through arcs.

An object network can be put in a graphical form, with places being represented by circles, arcs by arrows and instances of objects by tokens, as in figure 3.

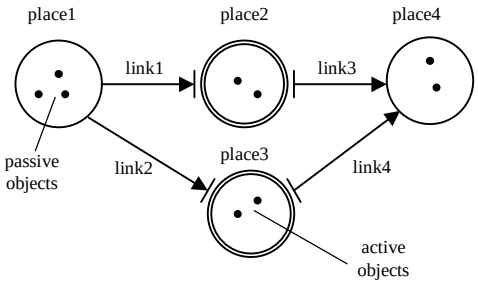

Figure 3. Example of an object network.

As for an objects system, the basic behavior in an object network is the triggering of active objects. The reader is referred to the work of Gudwin (Gudwin 1996, Gudwin and Gomide 1997a, Gudwin and Gomide 1997b, Gudwin and Gomide 1998a, Gudwin and Gomide 1998b, Gudwin and Gomide May 1998) for details about the formal definitions.

2.2.1 Places

Places are the components of an object network where objects are located. Each place is registered to a unique class, and every stored object should belong to this class. In this sense, there are passive and active places, named after their class type, as described in section 2.1.

Places are represented graphically by circles, where passive places are depicted by single line circles, and active places by double line circles, as illustrated in figure 4. Each place can have one or more ports, which allow the interchange of objects with adjacent places.

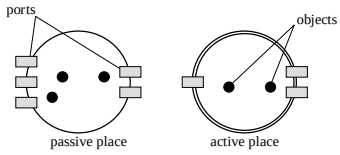

Figure 4. Place types.

2.2.2 Ports and arcs

Ports are interfaces of communication between places in object networks. According to the topology of the network these ports can be classified in *input ports*, when another place sends objects for this place through this port, or *output ports*, when this place sends objects to another place through this port, as shown in figure 5.

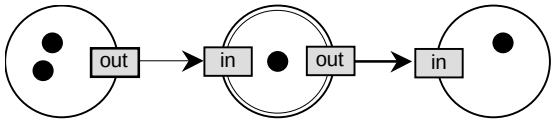

Figure 5. Classification of ports according to the topology of the object network.

According to the semantics involved there are two different types of ports (figure 6):

- *private ports*: these ports have different behavior depending on whether they are being input or output ports. Input private ports are used to feed active objects staying within an active place, to allow their triggering. Output private ports are used to deploy generated or transformed objects to other places. So, private ports are connected either to a domain or to a co-domain of an internal transforming function. These ports are part of the internal class definition and can exist solely on active classes, e.g., active places.

- *public ports*: these ports are used to put objects to, or get objects from a place. So, the use of this port is managed by a place container, instead of by an internal object. Public ports are a part of the place definition, independently of the class type it may host.

The number of private ports a place may have will depend on the internal functions their objects carry on. A place can have any number of public ports.

Arcs (figure 7) are used to create links between places, connecting input ports to output ports. Links are unaware of the kind of ports they connect, but some combinations are not considered valid.

- *private to private*, because it implies a racing condition between the supplier and the consumer.

- *public to public*, because this is meaningless, due to the fact that there is no transforming function, just an inert pipe. The correct way to do this is by an intermediate active place that takes the object from the source place and throws it into the destination place.

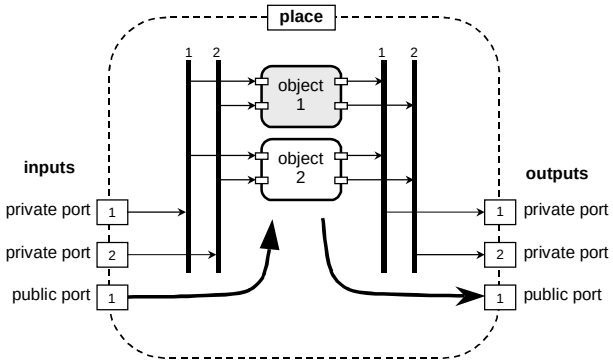

Figure 6. Classification of ports, according to the semantic involved.

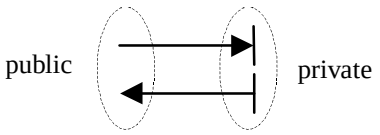

Figure 7. Graphical symbology adopted for arcs

Active places must have, by its own definition, at least one private port. They may or may not have public ports. Passive places, on the other hand, must have at least one public port and no private ports.

3 COMPUTATIONAL MODEL

The work described in this section is based on the current development of a computational tool, or a set of tools, that can supply researchers with an easy-to-use interface to model intelligent systems. This set of tools is a possible implementation of the concepts described in section 2. Our aim here is to construct a software tool[1] with the following features:

- **Robustness,** through an independent specification language that isolates the object network model from the host programming language.
- **Processing performance**, through automatic generation of stubs for each user defined class. These stubs are compiled together with the user code. External classes (in this case Java classes) can be incorporated by the network engine and manipulated as typical object network classes.
- **Scalability**, with support to SMP[2] capable computers through multi-thread architecture.
- **Easy-to-use interface**, provided by a graphical user interface and a specification language for each component on the object network.
- **Portability,** because it is totally implemented in Java (Sun Microsystems, 1998, 1999a, 1999b), it runs on any hardware/operating system that supports the Java Virtual Machine (Gosling et.al., 1999).

3.1 Design issues

This section will describe implementation aspects. Figure 8 shows how software objects interact with each other into the Java Virtual Machine. Each box represents a distinct thread.

The object network controller (Net Manager) is the first software object instantiated on the Java VM. Its main functions are:

- To define classes;
- To instantiate places;
- To control the iteration sequence;
- To provide the external communication interface with the client thread.

Each network place has a special management unit, named *place manager*, responsible for every transaction performed, including the communication with adjacent places and the controller.

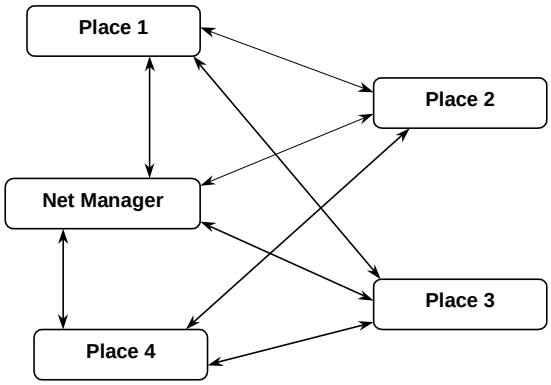

Figure 8. Diagram for the network from figure 3. Each box runs on its own thread. Arrows indicate communicating messages between distinct units.

From the user viewpoint, the interface with the external client, such as the graphical user interface, is also implemented in a message passing protocol (figure 9). This protocol links the client thread with the object network controller.

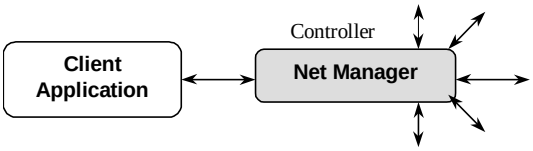

Figure 9. External communication between client and the MTON subsystem.

3.2 Object Network Specification Language

The design requirements for a robust specification and independence of platform specific issues led us to a definition of an Object Network Specification Language (ONSL). This language provides basic building blocks for each component of an object network:

classes	**network**
• variables	• places
• functions	• arcs
• private input/output ports	• kernel

Basically, there are two fundamental class types. The first is named *internal class* and contains, in its internal definition, only ONSL classes. The second type works as a shell, importing an external native language class (in this case a Java class) to the MTON environment. This type of class is called *external class*, and is extremely useful because it reuses previously available Java classes.

Figure 10 represents the network definition sample file corresponding to the object network shown in figure 3.

3.3 Object Network Compiler (ONC)

The Object Network Compiler (ONC) is the tool that processes ONSL input files, performing syntactic and semantic analysis, generating the specific Java stub for each user defined class.

The abstract Java class NetObject is the basic ancestor for all network objects, and provides the common behavior for all of them (figure 11). Stubs are generated based on the ONSL definition. They connect the standard internal interface of the NetObject base class with the user defined class particularities

[1] The main simulation engine module is named MTON (Multi-Threaded Object Network).
[2] SMP stands for Symmetric Multi Processing (Tanenbaum 1992).

by providing abstract methods, that are overridden in user classes.

```
import Samplenet.xFloat;

class C1 is xFloat;

class C2 {
   var v1 type C1;
   var v2 type C2;
   function f1 from v1 to v2 match from v1;
   input 1 to v1;
   output 1 to v2;
}

network Samplenet {
   place Place1 type C1 ports (0,2);
   place Place2 type C2;
   place Place3 type C2;
   place Place4 type C1 ports (2,0);
   arc Link1 from Place1(public 1)
             to Place2(private 1);
   arc Link2 from Place1(public 2)
             to Place3(private 1);
   arc Link3 from Place2(private 1)
             to Place4(public 1);
   arc Link4 from Place3(private 1)
             to Place4(public 2);
   kernel Place1 o1,o2;
   kernel Place2 o3;
   kernel Place3 o4;
}
```

Figure 10. Sample ONSL definition file.

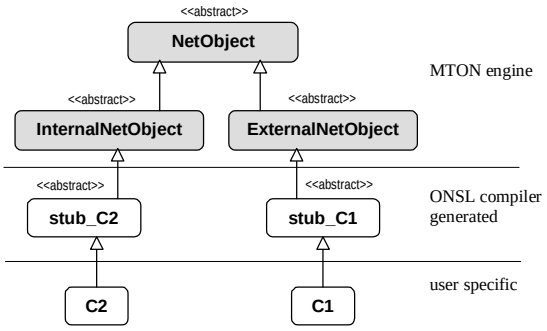

Figure 11. Java class structure.

3.4 Network Dynamics

One of the most crucial features of an object network is its parallelism capability, in which every object may execute concurrently with the rest of the network. The MTON architecture was designed with the assignment of a running thread to each place on the network. To do so, it implements a message passing sub-system to support its internal communication protocol.

Every object on the network may be triggered by any input combination that satisfies its input domain requirements, but this is a great problem because there are some dependency implications, when an object is activated. The first question is: from the many available functions within an object, which internal function should be selected? Only one internal function can be activated at the same time (Gudwin 1996). The second problem is more complex: which objects available on neighbor places, connected through incoming arcs, should be used to activate the chosen internal function? This problem can either be split into two other problems : the selection of the best objects, according to some goals, and a policy to avoid conflicts when the selected objects are desired also by other active objects. To implement this policy, we assign the following properties to available objects:

- *shareable+leave(s/l)*: several objects can read this object simultaneously. It will remain the same on its place.
- *shareable+clean(s/c)*: like a shareable, but the last consumer will automatically destroy it.
- *exclusive+leave(e/l)*: just one object can read this object. It will remain the same on its place.
- *exclusive+clean(e/c)*:just one object (the consumer) can read this object, after which it will be destroyed.

Actually, this problem is a little bit harder than explained, because the algorithm has to provide: (a) the objects to be activated, (b) the internal functions to be triggered, and (c) the objects to be consumed. This process needs to avoid any kind of conflicts that may arise between different objects. Notice that there is no sense in choosing a function when there is no object available to feed it, or if the same object is requested at the same time by different objects. In order to solve the whole problem we devised an algorithm to evaluate all dependencies among each object, searching for valid actions.

We use the following concepts:

- **access mode**: rules of use, and the consequences of them, when an object is used by another object;
- **combination**: a combination of possible incoming objects that satisfies the type requirements for a specific internal transforming function of a given class;
- **local combination**: This is a combination with the addition of a matching interest and access modes. Note that it does not take into account the rest of the network to avoid conflicts;
- **global (or valid) combination**: it is just like a local combination, but it takes into account all external relationships this decision may involve. Actually, this is a valid combination that can occur in a given context. In a given network situation we argue that the set of global combinations is a subset of the set of local combinations;

We start by building local combinations and evaluating their feasibility in order to reach global combinations. This process is performed by a *matching procedure*, in which each internal function of each object expresses its *interest* in using a given local combination. This interest is expressed either by a fitness value or a more elaborated solution, e.g. using possibility and necessity measures.

The proposed algorithm is based on an initial set of hypotheses:

1. Every internal function has one internal matching sub-function;
2. There are several possible sets of global combinations, but just one of them, with the best performance, must be found;
3. The searching for global combinations can be done on computing every internal matching interest of every function of every object on every place for each of their local combinations;

Best Matching Search Algorithm (BMSA-1):

1. Determine the set of local combinations for each place, including its objects.

2. Eval the matching interest of every previous local combination.

3. Group all previous sets from each place in one global set.

4. Create a list containing all consumable objects on the network, with no repetition (This is just the union of all objects used on the local combinations from step 1).

5. While *number of remaining objects* ≠ 0 and *number of remaining local combinations* ≠ 0 do:

   - From the list of local combinations get the highest matching value, and move its local combination to the list of global combinations.
   - For each input object in this local combination do:
   - if the access mode is exclusive remove it from the list of consumable objects and remove all local combinations that are using it from the respective list.
   - if the access mode is shareable remove all local combinations that use this object in exclusive mode.
   - if the access mode is self clean mark this object.

After the creation of the list of global combinations every object is contacted to perform its scheduled task.

3.5 Graphical User Interface

The MTON architecture is supported by a graphic user interface, which offers to the user tools for design and simulation, providing an easier way to work on object networks issues. The editor module is directly attached to the ONSL, e.g., every element in an object network can be designed on it. The simulator module enables the user to verify the dynamic behavior of the network in different situations. A screenshot of the editor module is shown in figure 12.

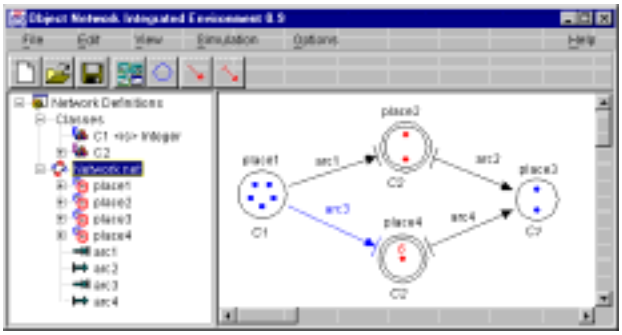

Figure 12. Sample screenshot of the graphical user interface.

4 CONCLUSIONS

With the implementation of the proposed software architecture, we upgrade the status of object networks from a purely conceptual tool to a useful, easy-to-use application tool, suited for the development of intelligent systems. With it, it becomes easier to use object networks as a modeling tool, in the development of intelligent systems.

Up to the time of writing this article, we were not aware of any other computational tool that could be compared with the work presented in this paper. This is the first computational implementation of a generic tool to design and build fully workable object networks. So, it is hard to assess specific features of the proposed system against other works. Object Networks were designed with the specific purpose to serve as a tool to help designing and building intelligent systems using the Computational Semiotics paradigm as a background. Until now, the use of object networks was restricted either to theoretical examples or specific illustration examples, hardcoded into standard computer languages. Now, with the introduction of MTON, we are able to develop, build and test general applications using the object network paradigm. Due to space limitations, though, we are not able to show here the set of tutorial examples we developped to illustrate the potential use of object networks. We have developped reference examples regarding fuzzy systems, neural networks, genetic algorithms and hybrid systems mixing them, together with a lot of other examples that are available in order to guide new users with the first steps in using MTON. Most of these examples, with the full workable system are publicly available. Just contact one of the authors. In the near future our research group will be providing the software for free in a dedicated web site.

Despite we are not able to compare MTON with other software tools implementing object networks (because it is actually the first one to do it), we are aware that other modeling tools do exist and are actually used in order to model some kinds of systems. One of these other modeling tools, that would be (in thesis) compared with object networks is Coloured Petri Nets (CPN) (Jensen, 1990). There is a computational tool to develop CPN's, called design/CPN (Meta Software Corporation 1993), which is widely used by the Coloured Petri Nets community. We understand, though, that a comparison among design/CPN and MTON would be not fair to design/CPN, because the object network paradigm enhances significatively over the CPN model (Gudwin, 1996).

At the same time MTON is the consolidation of object networks theory, it is also being used as a prototype for the enhancement of the own object network idea. As soon as new real applications are constructed by its means, we are able to analyze the demand for new capabilities, which can be incorporated, both to the mathematical/conceptual tool and to the application tool we developed. More than this, it is a testbed for computational semiotics, where its models and structures can be instantiated, tested and evaluated. So, more than a simple application tool, MTON is actually a research tool, where the steps to the future are going to be traced.

We see a lot of improvements that are required, though, from what has already been done. Basically, there are two categories of improvements. The first is related to the software system being deployed, considering new technological solutions, in order to achieve efficiency and reliability.

Among other things we suggest:

- Extension of the simulation engine from multiprocessor systems to a network distributed engine, where each place may run on a different machine. The way to that will be crossed by CORBA (Common Object Request Broker Architecture) and RMI (Remote Method Invocation) distributed computing architectures.
- New selection functions for scheduling the triggering objects. The concept of matching may be extended to a more general one.

- Introduction of software engineering analysis and design concepts to bring well-known tools to the arena of object networks.

The second is represented by the enhancements that can be made to the mathematical/conceptual model of object networks. These include enhancements like:

- Hierarchical modeling, in which each object within an object network can be modeled by an object network itself.
- A connection of object networks with semantic networks, allowing a better representation of the types of objects used and their relation to each other.
- The use of fielded and fuzzy objects (Gudwin, 1996) in its structure.

As we see it, the current implementation is just a step in the ambitious goal of creating a general theory of intelligent systems.

5 REFERENCES

Albus, J.S. (1991). "Outline for a Theory of Intelligence", *IEEE Transactions on Systems, Man, and Cybernetics*, vol 21, no.3.

Albus, J.S. (1997). *"Why Now Semiotics? - From Real-Time Control to Signs and Symbols"*- Proceedings of the 1997 International Conference on Intelligent System and Semiotics, pp. 3-7.

Ceska M.; Janousek V. *"Object Orientation in Petri Nets. Object Oriented Modeling and Simulation"* – Proceeding of the 22nd Conference of the ASU, University Blaise pascal, Clermont-Ferrand, france 1996, pp. 69-80.

Ceska M.; Janousek V. *"A Formal Model for Object Oriented Petri Nets Modeling"*. Advances in Systems Science and Applications, 1997.

Gosling, J.; Joy B.; Steele G., 1999, *Java™ Language Specification*, http://java.sun.com/docs/books/jls/html/index.html

Gudwin, R.R. (1996). "*A Contribution to the Mathematical Study of Intelligent Systems*" – PhD Thesis – DCA-FEEC-UNICAMP. (in portuguese)

Gudwin, R.R. and Gomide, F.A.C. (1997a), *Computational Semiotic: An Approach for the Study of Intelligent Systems – Part I: Foundations* – (Technical Report RT – DCA 09 – DCA-FEEC-UNICAMP, 1997).

Gudwin, R.R. and Gomide, F.A.C (1997b)., *Computational Semiotic: An Approach for the Study of Intelligent Systems – Part II: Theory and Application* – (Technical Report RT – DCA 09 – DCA-FEEC-UNICAMP, 1997).

Gudwin, R.R. and Gomide, F.A.C., Sep 1997, *"An Approach to Computational Semiotics" – (Proceedings of the ISAS´97 – Intelligent Systems and Semiotics: A Learning Perspective* – International Conference – Gaithersburg, MD).

Gudwin, R.R. and Gomide, F.A.C., Oct. 1997, *A Computational Semiotic Approach for Soft Computing* – (Proceedings of the IEEE SMC´97 – IEEE International Conference on Systems, Man and Cybernetics – Orlando, FL, USA.)

Gudwin, R.R. and Gomide, F.A.C.,1998a, *"Object Network: A Formal Model to Develop Intelligent Systems" in Computational Intelligence and Software Engineering*, J. Peters and W. Pedrycz (eds.) – World Scientific.

Gudwin, R.R. and Gomide, F.A.C., 1998b, *"Object Network: A Computational Framework to Compute with Words"* in Computing with words in Information/Intelligent Systems, L.A. Zadeh and J. Kacprzyk (Eds.) – Springer-Verlag.

Gudwin, R.R. and Gomide, F.A.C., May 1998, *"Object Networks – A Modeling Tool"* – Proceedings of FUZZY-IEEE98, WCCI'98 - IEEE World Congress on Computational Intelligence, Anchorage, Alaska, USA, pp. 77-82.

Jensen, K. (1990). *"Coloured Petri Nets: A High Level Language for System Design and Analysis"* – Lecture Notes in Computer Science 483 – Advances in Petri Nets, pp. 342-416.

Meta Software Corporation, 1993, Design/CPN Tutorial for X-Windows, http://www.daimi.aau.dk/designCPN

Murata, T. (1989) *"Petri Nets: Properties, Analysis and Applications"* – Proceedings of the IEEE, vol. 77, n. 4.

Newman A.; Shatz, S.M.; Xie, X. (1998). *"An Approach to Object System Modeling by State-Based Object Petri Nets"* – International Journal of Circuits, Systems and Computer.

Sun Microsystems, 1998, *JavaCC Documentation*, http://www.suntest.com/JavaCC/DOC/

Sun Microsystems, 1999a, *Java™ Plataform 1.2 API Specification*, http://java.sun.com/products/jdk/1.2/api/index.html

Sun Microsystems, 1999b, *Java™ Development Kit 1.2, Documentation*, http://java.sun.com/products/jdk/1.2/docs/index.html

Tanenbaum, A.S., 1992, *Modern Operating Systems* – Prentice Hall, Inc.