Object Networks – A Modeling Tool

Ricardo Gudwin and Fernando Gomide
DCA-FEEC-UNICAMP
Caixa Postal 6101 – 13.083-970 – Campinas, SP – Brasil
e-mails: gudwin@dca.fee.unicamp.br
gomide@dca.fee.unicamp.br

Abstract

This paper presents the introductory concepts related to the modeling tool called object networks. Inspired by both (Fuzzy) Petri nets and object oriented technology, this modeling tool has been recently addressed as a formal framework for Computational Semiotics, and also as a meta-theoretical approach to the integration of different paradigms within Computing with Words and Computational Intelligence. In a general sense, it is proposed as a tool for modeling, analysis and design of intelligent systems.

1. Introduction

Petri nets are a general-purpose mathematical tool for describing relations existing between conditions and events. They are a very suitable tool for modeling and analysis of information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, fuzzy and/or stochastic [1]. As a graphical tool, Petri nets can be used as a visual-communication aid similar to flow charts, block diagrams and networks. Tokens are used in these nets to simulate the dynamic/concurrent activities of systems. As a formal tool, state equations, algebraic equations and other mathematical models are used to analyze properties and characteristics of the modeled systems.

To extend its representation power and enhance its functionality, lots of different models were derived from the main definition of a Petri Net. Examples of those extensions are the predicate-transition [2] nets and the coloured Petri Nets [3,4], also known as high-level Petri Nets. More recently, object-oriented versions of Petri-nets [5,6,7,8,9,10] were developed. But none of them disrupt with the main idea in a Petri Net structure that is the existence of a set of places, a set of transitions and a set of arcs connecting places-transitions and transitions-places. Object networks introduce two distinguished characteristics when compared to Petri Nets. In the object net approach, there is no set of transitions. There are only a set of places and a set of arcs connecting them. The second difference (that was already suggested by some object-oriented versions of Petri Nets) is that tokens are individualized by objects. So, the tokens are not all equal. But, more than this, some objects, called active objects, i.e., objects that have a positive number of methods, perform the job of transitions. In this sense, objects within an object network can act both as a token and/or as a transition. And, as new objects can be created, object networks can have a variable number of transitions, what is not allowed in the definitions of Petri Nets. This is why we don't call our model an extended Petri Net, but only that it is "inspired" in a Petri Net.

Other authors tried to define Petri Nets with a variable structure, as the case of the adaptive design Petri nets [11] and the self-modifying Petri nets [12,13]. In the first case a sequence of Petri nets is generated, giving the illusion (as in the movies) of a network that adapts itself. In the second one, the arc parameters depend on the number of tokens in other places. These were earlier tries, but none of them give up with the idea of having a fixed set of places and transitions. They were intrinsic static structures. And the problem of using static structures is the inability to model systems with learning and adaptive characteristics.

In the study of intelligent systems, the field of Fuzzy Petri Nets emerged as a way of providing an interesting way of knowledge representation [14] and rule-based decisionmaking [15] . The generalized fuzzy Petri net [16], made also a connection to the field of neural networks using a type of logic based neuron. This suggests that the type of computations that can be performed by Petri nets-like structures is a fundamental issue in finding a modeling tool for intelligent systems, that could embrace aspects of different paradigms within computational intelligence [17] and computing with words [18].

Object networks were first introduced by Gudwin [19], in the context of intelligent systems modeling, analysis and design. They were used as a formal background for the development of Computational Semiotics [20,21,22], an emulation of the semiose loop (the elementary semiotic processing) in computer systems. More recently, object networks were also used within the context of computational intelligence and soft computing [23,24], and computing with words [25].

In this paper, we review the fundamental concepts concerning object networks. In section 2, we elaborate on the concept of object, and in section 3 we introduce the object network. We focus on the main conceptual aspects, giving only a concise formal representation. For the reader interested in the complete mathematical definitions, we suggest the work of Gudwin [19,20,21,24,25].

2. Objects

The main element in the modeling tool to be addressed here is the object, and its extensions, generic object and fuzzy object. We address objects in two different views. In the first, we present the conceptual object, i.e., the conceptual specification of objects. Next, we present a resumed formal model that implements this specification, and analyze the interaction among objects to compose object systems. In the next section, we address a particular kind of object system, the object network.

2.1 The Conceptual Object

Our concept of object [26] is closely related to its intuitive physical meaning. Ontologically, an object is an entity of the real world and is characterized by its properties. The properties are its attributes [27] . Based on a frame of reference, it is possible to find attributes distinguishing different objects. Thus attributes describe the objects. This view of objects does not consider that, in addition to its existence, the objects also “act” in real world. Therefore, a mathematical concept of object must model its active aspect.

The conceptualization of object cannot, in principle, be made in an independent way. Although we can imagine the existence of an object by itself, we should also consider its capability to interact with different objects. In other words, to introduce the main concepts about objects, we have to discuss object systems. An object system is a set of interacting entities. The object components, which allow interaction, are shown in figure 1.

Each active object is assumed to have two types of interfaces: input and output interfaces, as in figure 1. The input interface is composed by a collection of gates (input gates). Within an object we find its internal states. These states are divided in four regions. The first is a copy of the input interface whereas the second comprises internal variables. The third region is a copy of the output interface whereas the fourth region is a set of transformation (internal) functions. The output interface is composed by a collection of output gates.

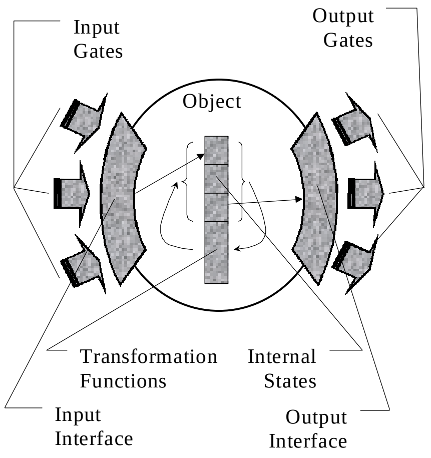

Figure 1 – The Conceptual Object

The interaction among objects is regulated by a mechanism called triggering, and is performed by active objects. In this mechanism, some objects are first bound to the active object, through the input gates, starting what is called the assimilation phase. In this phase, the active object copies the internal states of binding objects to its internal states. After assimilation, the bounded objects can be destroyed or released back to the system. If they are destroyed, then we have a destructive assimilation (or consumption). Otherwise, we have a non-destructive assimilation. In the second phase of triggering, the active object uses one of the transformation functions to change its internal states. Both, input and output, are parts of the internal states. This is called the transformation phase. After the transformation phase, some of the internal states of the active object are copied into the output interface. Next, another set of objects is bound to the output gates, and their internal states are changed to those present in the output interface. This last phase is called either generation phase or regeneration phase, depending on the objects that are bound to output gates. If the bounded objects already exist, then this process is called regeneration because it alters the internal states of bounded objects. However, this last phase can also create a new object, not part of the object system. In this case, the last phase creates this new object, fills its internal states with the information of the output interface, and releases it to the system. This process is called generation.

The triggering mechanism may allow different kinds of behavior, as illustrated in figure 2. In this example, object o_6 is the active object performing the triggering process.

Objects o_1, o_2 and o_3 are the objects to be assimilated in the triggering. Objects o_1 and o_4 are regenerated, and o_5 is generated. Note that o_1 is, at the same time, assimilated and regenerated. Object o_2, after assimilation, is released back to the system but o_3 is destroyed.

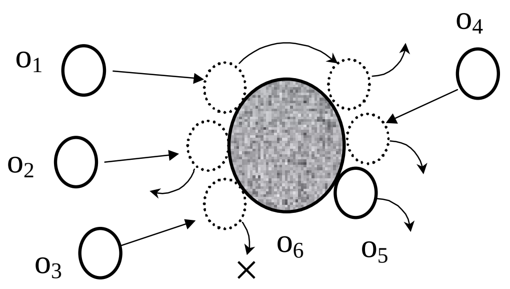

Figure 2 – Object Interactions

To control the triggering process, there is a special function associated with each object called the selection function. This function decides which objects are to be bound to input gates, which objects are to be bound to output gates, and which internal function is to be used in the triggering process. The control strategy of an object system is dictated by the selection functions.

Note, however, that the selection functions do have some restrictions. These restrictions concern the transformation functions requirements, as well as some problems involving synchronization. Each transformation (internal) function requires a minimum set of objects to start the triggering procedure. Therefore, the selection function must consider the simultaneous availability of all objects needed to enable a transformation function. The synchronization problems that may appear are related to multiple active objects binding the same object. For assimilation bindings, there should be a guarantee that only one active object is performing a destructive assimilation. If some assimilated object is also being regenerated, it must be regenerated by only one active object. And cannot be destructively assimilated in this case. In this sense, there should be a global policy for the selection functions, assuring that those constraints are satisfied.

With an appropriate implementation of selection functions, objects can become autonomous entities, i.e., independent of an external synchronization mechanism. Synchronism, despite being useful sometimes, is not a requirement in object systems. The behavior of real objects, with asynchronous and parallel activities can be modeled. Note that both assimilated and (re)generated objects are not necessarily passive. This allows adaptive and self-organizing systems to be modeled by object systems.

2.2 The Formal Object

This subsection introduces a resumed formal definition for objects and related concepts. The focus here is on the main issues and definitions only. For a more in depth coverage the reader is referred to the work of Gudwin [19,20,21,24,25].

Definition 1 – **Variable:** Let T be a countable set (a "time" set) with a generic element t, and $X \subseteq U$. A variable x of type X is a function $x : T \to X$. Note that a function is also a relation and hence it can be expressed by a set. Thus, $x \subset T \times X$.

Definition 2 – **Class:** A class C is a set whose elements c_i are tuples of the type:

$$(v_1, v_2, \dots, v_n, f_1, f_2, \dots, f_m), \ n \geq 0, m \geq 0$$

where $v_i \in V_i$, and f_j are functions

$$f_j : \bigtimes_{p \in P_j} V_p \to \bigtimes_{q \in Q_j} V_q .$$

Here $\bigtimes$ means the Cartesian product, $P_j \subseteq \{1, \dots, n\}$ and $Q_j \subseteq \{1, \dots, n\}$.

Definition 3 – **Object:** Let C be an non-empty class and c be a variable of type C. Thus c is an object of class C.

Observe that, by definition, an object is a multi-temporal entity, i.e., the definition of an object comprises all its time history. A more intuitive definition concerns an **instance** of an object, viewed as the value of an object at a particular instant of time, i.e., a tuple:

$$(v_1, v_2, \dots, v_n, f_1, f_2, \dots, f_m).$$

Definition 4 - **Set Variable:** Let N be an enumerable set, with a generic element n and $X \subseteq U$ a subset of U. We define a **set variable** x of type X as a function $x:N \to 2^X$.

Definition 5 - **Generic Object:** Let C be a non-empty class. Let c be a set variable of type C. The set variable c is called a **generic object** of class C.

Definition 6 - **Case of a Generic Object:** Let c be a generic object of class C. An object c' of type C is said to be a case of generic object c if $\forall n \in N$, $c'(n) \in c(n)$.

Definition 7 - **Fuzzy Object:** Let N be an enumerable set with a generic element n, X a class, $\tilde{X}$ a fuzzy set defined onto X and $2^{\tilde{X}}$ the set of all fuzzy sets onto X. We define a **fuzzy object** x of type X as a function $x : N \to 2^{\tilde{X}}$.

Definition 8 - **Object System**

A set of objects (or generic objects, or fuzzy objects) c_i is an object system if the c_i's are related to each other in the sense that each instance of such objects, at a given instant, is a function of the instances of all objects at the previous time instant:

$$c_k\,(t+1) = f\,(c_1(t), \ldots , c_n(t)\,).$$

This is only a concise definition of an object system. The complete definition is far more involved. The reader is referred to the work of Gudwin [19,20,21,24,25] for details.

3. Object Network

An object network is a special type of object system in which additional restrictions concerning interactions are included. To distinguish object network and object system let us assume places and arcs whose roles are similar to those used in Petri nets context. Objects in places can only interact with objects in places connected through arcs. Thus, at each instant, the objects defined should be at one place. For each place there is a set of places connected with, through input arcs. These places are called the input gates of the place. Analogously, each place has a set of places connected with it by means of output arcs, called output gates. For each field of output interface of objects in this place there should exist one corresponding output gate. With those conditions we can see that, for each place, there should be only objects of the same class. Remember that objects can be of two types: passive and active. Passive objects do not have functions in its tuples and are only used to store information. Active objects do have functions in its tuples, and perform the task of transitions in the object network. Each place can only have objects of the same class. In this sense, we can say that there are passive and active places if the objects that can be put in a place are passive or active, respectively. Apart of those special characteristics, an object network is similar to an object system.

Object networks can be put in a graphical form, with places being represented by circles and arcs by lines. Passive places are indicated by circles. Active places are indicated by double circles, and instances of objects by black tokens, as in figure 3.

Observe that, differently than in Petri nets, the tokens are instances of objects that have individuality, i.e., they are not a marking on the place, but are related to objects with attributes and eventually transforming functions. Again, active objects, which perform the role of transitions, are also mobile and changeable. This gives an object net great power of representation, allowing modeling of systems that are not suitable to be modeled by Petri nets, e.g., adaptive systems.

As for an object system, the basic behavior in an object network is the triggering of active objects. Triggering an active object corresponds to the generation of new instances of objects in places directly connected to the place where the active object is through output arcs.

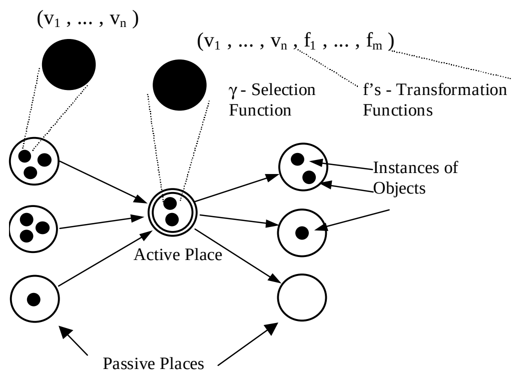

Figure 3 : Example of an Object Network

To be triggered, an object must first have an enabling scope, that is, a set of object instances put in the input gates, enabling one of the object functions. To select an enabling scope, there is a selection function that selects, from the object instances available, those that are to be used for triggering. After triggering, object instances may be put in one ore more output gates of the place where the active object is. This is also determined by the selection function. The object instances used as an enabling scope may (or not) be destroyed for the next time instant.

3.1 The Formal Object Network

An object network $\Re$ is a tuple

$$\Re = (\Sigma, \Pi, \Xi, A, \eta, fpi, fpo, \mathcal{C}, \xi, \gamma\,),$$

such that:

1)- an objects system $\Omega = \{\,(c_i\,, \gamma_i\,)\,\}$ is determined by choosing $c_i \in \mathcal{C}$ and $\gamma_i \in \gamma$, $0 \le i \le \delta$,

2)-for each object $c_i \in \mathcal{C}$ with a function f_j being triggered at n, being this object at n at a place $\pi = \xi(n,c_i\,)$, the objects s_i^k belonging to the generative scope S_i indicated by γ_i (n) should have a localization function defined by:

$$\xi(n+1,s_i^k\,) = \pi^k$$

where π^k should be such that $\eta\,(\,fpo_\pi\,(k')\,) = (\pi,\pi^k\,)$ and k' is the index of the k-th field of the input interface specific to function f_i of c_i referred at the output interface of c_i .

In this definition we have that:

a) $\Sigma = \{C_i\}$ is a set of classes

b) $\Pi = \{\pi_i\}$ is a set of places π_i

c) $\Xi = \Xi(\pi)$ is a mapping of classes $\Pi \rightarrow \Sigma$, such that $\forall \pi \in \Pi$ for each field v_i of input interface of objects from class $\Xi(\pi)$, being v_i an object from class C, $\exists \pi_k$, $\pi_k \in F(\pi)$, such that $\Xi(\pi_k) = C$, and for each field v_i of output interface of objects from class $\Xi(\pi)$, being v_i an object from class C, $\exists \pi_k$, $\pi_k \in V(\pi)$, such that $\Xi(\pi_k) = C$.

d) A is a set of arcs $A = \{a_i\}$

e)η is a node function $\eta : A \rightarrow \Pi \times \Pi$

f) fpi = $\{fpi_i\}$ is a set of attribute functions for input gates, where each $fpi_i : \{1, \ldots, \partial_i\} \rightarrow A$ is the attribute function for input gates of objects which are at a place π_i .

g) fpo = $\{fpo_i\}$is a set of attribute functions for output gates, where each $fpo_i = \{1, \ldots, \rho_i\} \rightarrow A$ is the attribute function for output gates of objects that are at a place π_i .

h) $\mathcal{C} = \{c_i\}$, is a set of objects where c_i are objects from a class C_i , $C_i \in \Sigma$, $0 \leq i \leq \delta$, $\delta > 0$.

i) ξ is the localization function $\xi : N \times \mathcal{C} \rightarrow \Pi$, which relates to each object $c \in \mathcal{C}$, for each instant n, a place π.

j) $\gamma = \{\gamma_i\}$, $0 \leq i \leq \delta$, $\delta > 0$, is a set of selection functions, where each element γ_i is a selection function $\gamma_i : N \rightarrow 2^{\mathcal{C} \times B} \times 2^{\mathcal{C}} \times \Theta_i$ for an object c_i . These functions selects, for each instant n, a set of objects to be assimilated by object c_i (its enabling scope H_i), a set of objects to be generated by object c_i (its generative scope S_i) and the index for the internal function to be executed by the object. These selections have as restrictions that:

$\forall (c,b) \in H_i$,
 $\xi(n,c) = \pi$, $\pi \in F(\xi(n,c_i))$,
 if b = 1, $(\forall k \neq i)((c,1) \notin H_k)$,
$\forall c \in S_i$,
 $\xi(n+1,c) = \pi$, $\pi \in V(\xi(n,c_i))$,
 $(\forall k \neq i)(c \notin S_k)$ and
 $(\forall k)((c,1) \notin H_k)$.

If c_i is a passive object or, for a given n, $\nexists H_i \neq \emptyset$ or $\nexists S_i \neq \emptyset$ then $\gamma_i(n) = (\emptyset, \emptyset, 0)$. The third index being 0 does mean that no internal function is going to be executed, i.e., the object is not going to trigger.

As additional definitions we have:

k) $F(\pi)$ is a mapping $\Pi \rightarrow 2^{\Pi}$, defined by $F(\pi) = \cup \pi_k$ where $k \in K$, $K = \{k \mid \exists a_j \in A$ such that $\eta(a_j) = (\pi_k,\pi)\}$.

l) $V(\pi)$ is a mapping $\Pi \rightarrow 2^{\Pi}$, defined by $V(\pi) = \cup \pi_k$ where $k \in K$, $K = \{k \mid \exists a_j \in A$ such that $\eta(a_j) = (\pi,\pi_k)\}$.

m) $X(\pi)$ is a mapping of connections $\Pi \rightarrow 2^{\Pi}$, such that $X(\pi) = F(\pi) \cup V(\pi)$.

n) i^i is the input interface of an object from class $\Xi(\pi_i)$.

o) o^i is the output interface of an object from class $\Xi(\pi_i)$.

p) ∂_i is the number of fields in i^i and ρ_i the number of fields in o^i .

q) $\Theta_i = \{0, \ldots, m_i\}$, where m_i is the number of function for object c_i

4. Conclusions

In this paper, we presented the main issues of a tool aimed to be used in modeling, analysis and design of intelligent systems. We covered the tool description and main concepts only, referring the interested reader for other publications providing a more in depth coverage. This is a very promising tool for the task of building a unified mathematical representation for intelligent systems, considering the preliminary results obtained so far [19-25]. Along with the theory of Computational Semiotics, it may provide, in the future, a substrate for a theory of intelligent systems.

It is important to say, however, that there is still a lot to do to achieve this goal. The tool is defined only for discrete time. The extension for continuous time would require a lot of changes in the model, which are not trivial. The same must be said about object's internal functions. In the current development, we assume instantaneous executing times for such functions, what is not necessarily the best approach (despite being the easiest). We have also a problem with hierarchical objects, a problem that is not totally solved within object oriented systems [28] theory. Other missing point is the lack for formal analysis tools. Analysis parameters similar to those found in Petri nets (e.g. boundedness, liveness, reachability, coverability, safeness, non-interference, etc) are still to be developed. Other analysis methods, like the invariant method [29], may have to be ported to be used in object networks.

We hope to address such matters in the future.

5. References

[1] Murata, T. "Petri Nets : Properties, Analysis and Applications" - Proceedings of the IEEE, vol. 77, n. 4, April 1989.

[2] Genrich, H.J.; Lautenbach, K. "System Modelling with High Level Petri Nets" - Theoretical Computer Science 13, pp. 109-136, 1981.

[3] Jensen, K. "Coloured Petri Nets and the Invariant Method" - Theoretical Computer Science 14 pp. 317-336, 1981

[4] Jensen, K. "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, 1990.

[5] Baldassari, M.; Bruno, G. - “An Enviroment for Object Oriented Conceptual Programming Based in PROT Nets” - Lecture Notes in Computer Science 340 - Advances in Petri Nets, pp. 1-19, 1988.

[6] Battiston, E. ; De Cindio, F. Mauri, G. - “OBJSA Nets : A Class of High Level Nets having Objects as Domains” - Lecture Notes in Computer Science 340 - Advances in Petri Nets, pp. 20-43, 1988.

[7] Cardoso, J. ; Valette, R. ; Dubois, D. - “Petri Nets with Uncertain Markings” - Lecture Notes in Computer Science 483 - Advances in Petri Nets, pp. 64-78, 1990.

[8] Guerrero, D.; Figueiredo, J.; Perkusich, A. – “Object-Based High-Level Petri Nets as a Formal Approach to Distributed Information Systems” - SMC’97 - IEEE International Conference on Systems, Man and Cybernetics, pp. 3383-3388, Orlando, FL, USA, October, 12-15, 1997.

[9] Adamou, M.; Bourjault, A. – “Assembly Systems Control Models Translation in an Object Oriented Language Environment using Object Oriented Petri Nets Hierarchical Formalism” - SMC’97 - IEEE International Conference on Systems, Man and Cybernetics, pp. 3924-3929, Orlando, FL, USA, October, 12-15, 1997.

[10] Zha, X.; Lim, S.; Fok, S. – “Concurrent Intelligent Design and Assembly Planning (CIDAP): Object-Oriented Intelligent Petri Nets (OOIPNs) Approach” - SMC’97 - IEEE International Conference on Systems, Man and Cybernetics, pp. 3930-3935, Orlando, FL, USA, October, 12-15, 1997.

[11] Zhou, M.C. ; Dicesare, F. - “Adaptive Design of Petri Net Controllers for Error Recovery in Automated Manufacturing Systems” - IEEE Transactions on System, Man and Cybernetics, vol. 19, n.5 September/October, pp. 963-973, 1989.

[12] Valk, R. - “Self-Modifying Nets - A Natural Extension of Petri Nets” - Lecture Notes in Computer Science 62 - Automata, Languages and Programming, pp. 464-477, 1978.

[13] Valk, R. - “Generalizations of Petri Nets” - Lecture Notes in Computer Science 118 - Mathematical Foundations of Computer Science, pp. 140-156, 1981.

[14] Chen, S.; Ke, J.; Chang, J. – “Knowledge Representation using Fuzzy Petri Nets” – IEEE Transactions on Knowledge and Data Engineering, vol. 2, no.3, pp. 311-319, September, 1990.

[15] Looney, C.G. – “Fuzzy Petri Nets for Rule-Based Decisionmaking” – IEEE Transactions on Systems, Man and Cybernetics, vol. 18, no. 1, pp. 178-183, Jan/Feb 1988.

[16] Pedrycz, W.; Gomide, F. – “A Generalized Fuzzy Petri Net Model” – IEEE Transactions on Fuzzy Systems, vol. 2, no. 4, pp. 295-301, Nov. 1994.

[17] J.Zurada, R.J.Marks II, and C.J.Robinson, - *Computational Intelligence - Imitating Life* – (IEEE Press, USA, 1994).

[18] L.Zadeh, “Fuzzy Logic = Computing with Words” – *IEEE Transactions on Fuzzy Systems*, pp. 103-111, vol. 4, no. 2, May (1996).

[19] Gudwin, R.R. – “A Contribution to the Mathematical Study of Intelligent Systems” – PhD Thesis – DCA-FEEC-UNICAMP, May, 1996. (in portuguese)

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

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

[22] Gudwin, R.R. and Gomide, F.A.C., *An Approach to Computational Semiotics* – (Proceedings of the ISAS’97 – Intelligent Systems and Semiotics : A Learning Perspective – International Conference – Gaithersburg, MD, USA - 22-25 September, 1997).

[23] Gudwin, R.R. and Gomide, F.A.C., *A Computational Semiotics Approach for Soft Computing* – (Proceedings of the IEEE SMC’97 – IEEE International Conference on Systems, Man and Cybernetics - Orlando, FL, USA - 12-15 October, 1997).

[24] Gudwin, R.R. and Gomide, F.A.C., “Object Network : A Formal Model to Develop Intelligent Systems” in *Computational Intelligence and Software Engineering*, J.Peters and W. Pedrycz (Eds.) – World Scientific, 1998.

[25] Gudwin, R.R. and Gomide, F.A.C., “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, 1998.

[26] Snyder, A. - “The Essence of Objects : Concepts and Terms” - IEEE Software, Jan. 1993, pp. 31-42.

[27] Wand, Y. - “A Proposal for a Formal Model of Objects” - in Object-Oriented Concepts, Databases and Applications, W. Kim and F. Lochovsky, eds., ACM Press, New York, 1989, pp. 537-559

[28] Wegner, P. - “Interactive Foundations of Object-Based Programming” - in Object Technology - A Virtual Roundtable, H. El-Rewini, S. Hamilton, IEEE Computer, vol. 28 n. 10, October 1995, pp. 70-72.

[29] Jensen, K. - “How to Find Invariants for Coloured Petri Nets” - Lecture Notes in Computer Science 118 - Mathematical Foundations of Computer Science, pp. 327-338, 1981.

Acknowledgements: The first author acknowledges CNPq for a fellowship. The second author acknowledges CNPq for grant #300729/86-3 and FAPESP, for their support.