OBJECT NETWORK : A FORMAL MODEL TO DEVELOP INTELLIGENT SYSTEMS

R.R. GUDWIN, F.A.C. 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: The aim of this work is to introduce an approach based on a generalization of the object-oriented paradigm for system modeling and implementation. The key concept is object network, a set of connected places containing objects modeling the characteristics and properties of a given system. Object network has a naturally distributed structure, is derived from a general and formal approach for a theory of objects, and provides a framework particularly suitable for computational intelligence. An object network has both, formal and representational power needed to develop and to implement intelligent systems.

1 Introduction

Attempts to develop artificially intelligent systems have a long and rich history. The last quarter of century has revealed a number of fundamental insights, from approaches to machine reproduction behavior, control and communication in the animal and machine, to artificial intelligence and, more recently, computational intelligence. In particular, computational intelligence has come up as a field embracing neural networks, evolutionary computation, fuzzy systems, artificial life, probabilistic reasoning.

Intelligent systems may be viewed as a class of systems in which intelligence arises as a consequence of some kind of embedded knowledge, learning and adaptation. From the computational point of view, intelligent systems depend on architectures for information processing, that is, architectures to determine how they perceive an environment, to recognize and interpret the objects in it, and to specify subsequent actions based on objects recognition. Clearly, a central issue in computational intelligence is the very basic notion of object.

Generally speaking, an object is an identifiable entity that plays a role in an environment. In computational terms, an object is an identifiable entity that plays a visible role in providing a service to a user or program[1] under request. Therefore, it is intuitively reasonable to think about a computational object as a means to model physical and, more abstractly, non-physical objects of an environment. A computational object explicitly embodies an abstraction characterized by the behavior of certain requests. Services may access or modify information associated with objects. Services are described independently of the form of data and algorithms used to implement the services. Behavior, in particular, can have several implementations.

Although a considerable effort in object-oriented programming, there is still no commonly accepted formal definition of the object-oriented approach. For instance, Wand[2] suggests a formal model in which the object-oriented paradigm is shifted from implementation-driven to modeling-driven. Wolczko[3] suggests a formal, meta-language specification in which object-oriented programming languages should adhere to.

Despite the numerous approaches and languages for object-oriented programming available today, they still lack formal and representational power which is mandatory within intelligent systems framework. For instance, any intelligent system should be able to dynamically create and destroy objects; to associate objects into a higher abstraction levels to assemble time variant cognitive concepts; to cooperate, to compete and to change its own structure to achieve adaptation and learning capabilities.

In this work, we introduce the notion of object network aiming at not only to provide a more general and formal approach for a theory of objects, but also as a computational framework to develop intelligent systems. After this brief introduction, the paper proceeds presenting a mathematical theory of objects. Next, the idea of an object system is developed, followed by the central concept of object network. An object network has both, formal and representational power needed to develop and to implement intelligent systems, a feature that is not available in most current approaches.

2 Foundations for a Mathematical Theory of Objects

In this section we build the formal body of the theory that is addressed here. It starts by introducing some preliminary definitions used as a base for further developments. After, we discuss conceptual objects, followed by its formal definition. This discussion is first related to the object as an individual, and next by the study of its interaction with other objects, to compose object systems. At the end we define the object network, and its extensions developed to enhance the representation power of classes of knowledge.

2.1 Preliminary Definitions

Here, we introduce the concepts and definitions used as a formal background for further developments. 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[4,5,6,7] . The definitions assume a discrete set N associated with time instants, or algorithm steps when convenient. Extensions to the continuous case may be possible, but it will not be considered here. Usually N is the set of natural numbers, but in general it can be any countable set.

A remark concerning notation: no distinction between a function and its graph will be made. Therefore, both $f : A \rightarrow B$ and $f \subset A \times B$ will be used to express a function f.

Definition 1 – **Tuples :** Let $q_1 , q_2 , \ldots , q_n$ be generic elements of the sets $Q_1 , Q_2 , \ldots , Q_n$ respectively. A tuple is a structure joining $q_1 , q_2 , \ldots , q_n$ into a single element denoted by $q = (q_1 , q_2 , \ldots , q_n)$.

A tuple with n elements is an n-tuple, or tuple for short. The elements of a tuple are called components. They can be identified by the indices associated with the order in which they appear. Note that a tuple component can be itself a tuple. In this case they are called complex tuples. For example $q = (q_1 , (q_{21} , q_{22} , q_{23}), q_3 , q_4 , q_5)$ is a complex tuple. To simplify notation we may assign $q_2 = (q_{21} , q_{22} , q_{23})$ turning the original tuple into $q = (q_1 , q_2 , q_3 , q_4 , q_5)$.

Definition 2 - **Tuple Arity:** Let $q = (q_1 , q_2 , \ldots , q_n)$ be a tuple. The **arity** of q, Ar(q), is the number of tuple's components: Ar(q) = n.

Examples: q = (a,b,c), Ar(q) = 3; q = (a,(b,c),d), Ar(q) = 3; q = ((a,b,c),(d,(e,f),g)), Ar(q) = 2.

Definition 3 - **Reference Index:** Let q be tuple. To identify an individual component of q we associate a reference index to each of its element. For a simple tuple the reference index will be a number i, $1 \leq i \leq Ar(q)$. For a complex tuple the reference index will be a simple tuple i with each element i_k corresponding to a sub-index of level k. The value of the sub-indices ranges between one and the arity of the tuple at level k. The reference index can also be used to identify the domain of the components. For example, let $s \in S$ be simple, and $c \in C$ complex tuples. Thus the reference indices and the component domains are found as follows:

$s = (a,b,c)$, $S = S_A \times S_B \times S_C$
$i=1 \rightarrow s_i = a$, $S_i = S_A$
$i=2 \rightarrow s_i = b$, $S_i = S_B$
$i=3 \rightarrow s_i = c$, $S_i = S_C$
$c = (a,(b,d))$, $C = C_A \times (C_B \times C_C)$
$i=1 \rightarrow c_i = a$, $C_i = C_A$
$i=2 \rightarrow c_i = (b,d)$, $C_i = C_B \times C_C$
$i=(2,1) \rightarrow c_i = b$, $C_i = C_B$
$i=(2,2) \rightarrow c_i = d$, $C_i = C_C$
$c = (a,(b,(s,d,(e,f),g),h))$, $C = C_A \times (C_B \times (C_C \times C_D \times (C_E \times C_F) \times C_G) \times C_H)$
$i=(2,1) \rightarrow c_i = b$, $C_i = C_B$
$i=(2,2,3) \rightarrow c_i = (e,f)$, $C_i = C_E \times C_F$
$i=(2,2,3,2) \rightarrow c_i = f$, $C_i = C_F$
$i=(2,3) \rightarrow c_i = h$, $C_i = C_H$
$i=2 \rightarrow c_i = (b,(s,d,(e,f),g),h)$, $C_i = C_B \times (C_C \times C_D \times (C_E \times C_F) \times C_G) \times C_H$

Definition 4 **- Induction Formula :** Let $q = (q_1 , q_2 , \ldots , q_n)$ be a tuple and k be an expression defined by the following syntax

$k \leftarrow [\, i \,]$
$i \leftarrow i , i$
$i \leftarrow [\, i , i \,]$

where i is a reference index of q. The expression k is called an induction formula.

Examples: $k = [\, i_1 , [\, i_2 , i_3 , i_4 \,] , i_5 \,]$
$k = [\, [i_1 , i_2], [i_3 , [i_4 , i_5]] \,]$
$k = [i_1 , i_2 , i_3]$
where i_j are reference indices of q.

Definition 5 **- Induction of a tuple:** Let $q = (q_1 , q_2 , \ldots , q_n)$ be a tuple in $Q = Q_1 \times \ldots \times Q_n$ and k be an induction formula. The induction of q according k is defined as the new tuple $q_{(k)}$ induced by the induction formula. The induced tuple $q_{(k)}$ is found from k by changing brackets and each reference index i_j into parenthesis and q_{i_j} of the original tuple q. The domain $Q_{(k)}$ of $q_{(k)}$ is found similarly.

Examples: $q = (a,b,c,d)$, $Q = Q_1 \times Q_2 \times Q_3 \times Q_4$, $k = [1,3,4,2\,]$,
$q_{(k)} = (a,c,d,b)$, $Q_{(k)} = Q_1 \times Q_3 \times Q_4 \times Q_2$

$q = (a,b,c,d)$, $Q = Q_1 \times Q_2 \times Q_3 \times Q_4$, $k = [4,1]$,
$q_{(k)} = (d,a)$, $Q_{(k)} = Q_4 \times Q_1$

$q = (a,b,c,d)$, $k = [\ 1, [2, 3]\ , 4]$,
$q_{(k)} = (a, (b,c), d)$, $Q_{(k)} = Q_1 \times (Q_2 \times Q_3) \times Q_4$

$q = (a,(b,c),d)$, $Q = Q_1 \times (Q_2 \times Q_3) \times Q_4$,$k = [1,(2,1),(2,2),3]$,
$q_{(k)} = (a,b,c,d)$, $Q_{(k)} = Q_1 \times Q_2 \times Q_3 \times Q_4$

$q = (a, (b,c), d)$, $Q = Q_1 \times (Q_2 \times Q_3) \times Q_4$, $k = [3,2]$,
$q_{(k)} = (d,(b,c))$, $Q_{(k)} = Q_4 \times (Q_2 \times Q_3)$

$q = (a, (b,c), d)$, $Q = Q_1 \times (Q_2 \times Q_3) \times Q_4$, $k = [3,2,(2,1)]$,
$q_{(k)} = (d,(b,c),b)$, $Q_{(k)} = Q_4 \times (Q_2 \times Q_3) \times Q_2$

Definition 6 - **Sub-tuple:** A tuple $q_{(k)}$ is called a sub-tuple of q if k has only one pair of brackets, and each reference index in k is unary and appears only once in k.

Definition 7 – **Relation:** If $R_1 , \ldots , R_n$ are sets and $R = \{(r_{i1} , \ldots , r_{in})\}$, $i = 1, \ldots , M$, is a set of M tuples with arity $n > 1$ such that $\forall i \in \{1, \ldots ,M\}$, $\forall k \in \{1, \ldots , n\}$, $r_{ik} \in R_k$, then the set R, $R \subseteq R_1 \times \ldots \times R_n$ is a relation in $R_1 \times \ldots \times R_n$,

Definition 8 – **Projection:** Let $R = \{r_i \}$, $r_i = (r_{i1} , \ldots , r_{in})$ be an n-ary relation in $R_1 \times \ldots \times R_n$ and k be an induction formula with unary indices $k = [k_1 , k_2 , \ldots , k_m]$, $k_i \in \{1, \ldots , n\}$, $k_i \neq k_j$, if $i \neq j$, $i = 1, \ldots , m$, $j = 1, \ldots , m$, $m \leq n$. The projection of R on $R_{k_1} \times \ldots \times R_{k_m}$,denoted by $R\downarrow R_{k_1} \times \ldots \times R_{k_m}$ (alternatively, $R_{(k)}$) is the relation obtained by the union of all sub-tuples $r_{i(k)} = (r_{ik_1}, \ldots, r_{ik_m})$ of R originated from the induction of R's tuples according to k, $R_{(k)} = \cup\, r_{i(k)}$.

Examples: $A = \{1, 2\}$ $B = \{a,b,c\}$ $C = \{\alpha, \beta, \gamma)$. $R=\{(1,a,\beta), (2,c,\alpha), (2,b,\beta), (2,c,\beta)\}$
$R \downarrow A \times C = \{ (1,\beta), (2,\alpha), (2, \beta) \}$
$R \downarrow C \times B = \{ (\beta,a), (\alpha,c), (\beta,b) , (\beta,c)\}$

Definition 9 - **Free Projection:** Let $R = \{r_i \}$, $r_i = (r_{i1} , \ldots , r_{in})$ be an n-ary relation defined in $U = R_1 \times \ldots \times R_n$ and k be an induction formula. The free projection of R in $U_{(k)}$, $R \downarrow U_{(k)}$ (alternatively, $R_{(k)}$) is the relation obtained by the union of all sub-tuples $r_{i(k)}$ originated by the induction of the tuples from R according to k: $R_{(k)} = \cup\, r_{i(k)}$.

NOTE: Free projection is a generalization of projection. Recall that in a projection, the induction formula has unary indices only. This implies in tuples defined only over the main dimensions of the original tuple. In free projection, any element, in whatever level of a tuple, can be used to define the inducted tuple. Clearly, with the proper induction formula free projection becomes standard projection.

Definition 10 - **Cylindrical Extension:** Let $R = \{ (r_{i1}, r_{i2}, \dots, r_{in}) \}$ be an n-ary relation in $R_1 \times \dots \times R_n$. The cylindrical extension P of R in $P_1 \times \dots \times P_m$, denoted by $P = R \uparrow P_1 \times \dots \times P_m$, where $\forall k \in \{1, \dots, n\}\ \exists P_j = R_k$, $1 \le j \le m$, is the greatest (in the sense of the greatest number of elements) relation $P \subseteq P_1 \times \dots \times P_m$ such that $P \downarrow R_1 \times \dots \times R_n = R$.

Example: $A = \{1, 2\}$ $B = \{a,b,c\}$ $C = \{\alpha, \beta, \gamma)$. $R = \{ (1,a), (2,c) \}$
$R \uparrow A \times B \times C = \{ (1,a,\alpha), (2,c,\alpha), (1,a,\beta), (2,c,\beta), (1,a,\gamma), (2,c,\gamma) \}$
$R \uparrow C \times A \times B = \{ (\alpha,1,a), (\alpha,2,c), (\beta,1,a), (\beta,2,c,), (\gamma,1,a,), (\gamma,2,c,).$

NOTE: As in projection, the order of elements in tuples of the cylindrical extension it is not the same as in the original tuples.

Definition 11 – **Junction:** Let R and S be two relations in $R_1 \times \dots \times R_n$ and $S_1 \times \dots \times S_m$, respectively, and $P = P_1 \times \dots \times P_o$ an universe where $\forall i \in \{1, \dots, n\}\ \exists P_k = R_i$, and $\forall j \in \{1, \dots, m\}\ \exists P_h = S_j$, $o \le n + m$. The junction of R and S under P, denoted by $R * S |_P$, is $R * S |_P = R \uparrow P \cap S \uparrow P$.

NOTE: If there is a $R_i = S_j$ then there may be only one set P_k with elements in tuples of R*S. In this case, for the tuples to be included in junction, the value of such element in the tuple in R and S should be the same (see first example).
NOTE: If $\forall i,j$, $R_i \neq S_j$, then $R * S |_P \downarrow R_1 \times \dots \times R_n = R$ and $R * S |_P \downarrow S_1 \times \dots \times S_m = S$.
Example: $A = \{1, 2\}$ $B = \{a,b,c\}$ $C = \{\alpha, \beta, \gamma)$. $R = \{ (1,a), (2,c) \}$ $S = \{(a,\alpha), (b,\beta)\}$
$R * S |_{A \times B \times C} = \{ (1,a,\alpha) \}$
$R * S |_{A \times B \times B \times C} = \{(1,a,a,\alpha), (1,a,b,\beta), (2,c,a,\alpha), (2,c,b,\beta) \}$

Definition 12 – **Variable:** Let N be a countable set with a generic element n (comprising some type of time measure), and $X \subseteq U$. A variable x of type X is a function $x : N \rightarrow X$. Note that a function is also a relation and hence it can be expressed as a set. Thus, $x \subset N \times X$.

Examples: $N = \{1, 2, 3\}$, $X = \{a, b, c\}$, $x(1) = a$, $x(2) = b$, $x(3) = c$ or
$x = \{ (1, a), (2, b), (3, c) \}$

Definition 13 - **Composed Variable:** Let x be a variable of type X. If the elements of X are n-tuples with $n > 1$, then x is called a composed variable (or structure).

The value of a composed variable, in a particular instant of time, will always be a tuple. The individual value of each sub-element of this tuple can be obtained by its reference index, the field of the variable. If $X = X_1 \times \ldots \times X_n$, then each field of x can be viewed as a free projection on $N \times X_i$, i.e., it is a standard variable of type X_i.

Example: $N=\{1, 2, 3\}$, $X_1 = \{a,b\}$, $X_2 = \{c,d\}$ $X = X_1 \times X_2 = \{ (a,c),(a,d),(b,c),(b,d)\}$

$$x = \{ (1,(a,c)) , (2,(a,d)), (3, (a,d)) \}$$

$$x \downarrow N \times X_1 = \{ (1,a) , (2,a), (3, a) \}$$

$$x \downarrow N \times X_2 = \{ (1,c) , (2,d), (3, d) \}$$

2.2 The Conceptual Object

Before to mathematically define an object, we briefly describe the conceptual object. Our concept of object 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[2] . 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 that 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 4 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, and the fourth region is a set of transformation (internal) functions. The output interface, similarly to the input one, is composed by a collection of output gates.

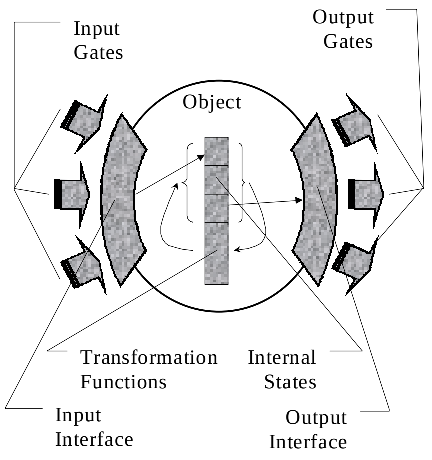

Figure 1 – The Conceptual Object

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, we have a destructive assimilation (or consumption). Otherwise, 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 in the internal states. This is called the transformation phase. After the transformation phase, some of the active object internal states 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 are existing objects, 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.

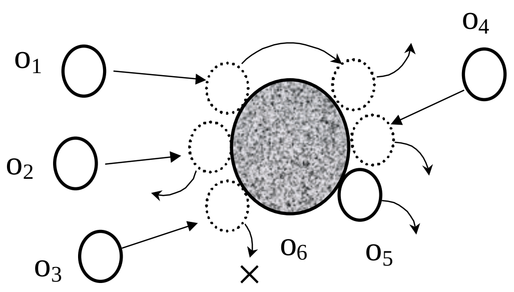

Figure 2 – Object Interactions

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.

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.

2.3 The Mathematical Object

Using the preliminary definitions, we can now proceed to define the mathematical concepts involving objects, turning the conceptual object into a mathematical one.

Definition 14 – **Class:** A class C is a set whose elements c_i are tuples of the type:

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

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, $P_j \subseteq \{1, \ldots, n\}$ and $Q_j \subseteq \{1, \ldots, n\}$.

Definition 15 – **Object:** Let C be an non-empty class and c be a variable of type C. Thus c is an object of class C.

It is worth noting that an object, as a variable, supports objects composed by parts which are objects themselves. In addition, if n=1 and m=0 then the tuple reduces to a single element. Hence a standard variable (a primitive object) is an object. For an empty class n=m=0 and there is no object. Clearly structures are also object. As it will be seen later, a structure is a passive object.

Definition 16 - **Instance of an Object:** Let c be an object of class C. The instance c(n) is the value of c at n.

C is a set of tuples. Therefore the instance of an object is an element of C, i.e. a tuple.

Definition 17 - **Superclass and Subclass:** Let C be a class. The set D whose elements are sub-tuples of the elements of C belonging to the same universe, and each element in C is associated with one element of D and D is itself a class, is called a superclass of C. In this case C is a subclass of D.

Note that a class can be defined from primitive classes. Since class is a relation, another class can be generated by the cylindrical extension of a class, by the junction of two or more classes, or by both junction and cylindrical extensions. In all cases, the primitive classes are superclasses of the newly generated class. Moreover, for a given a class its cylindrical extension is a subclass of itself. The junction of two classes is a subclass of both of them. Any class is a subclass of empty class. Therefore a hierarchy of classes is induced by projections, junctions and cylindrical extensions (figure 3).

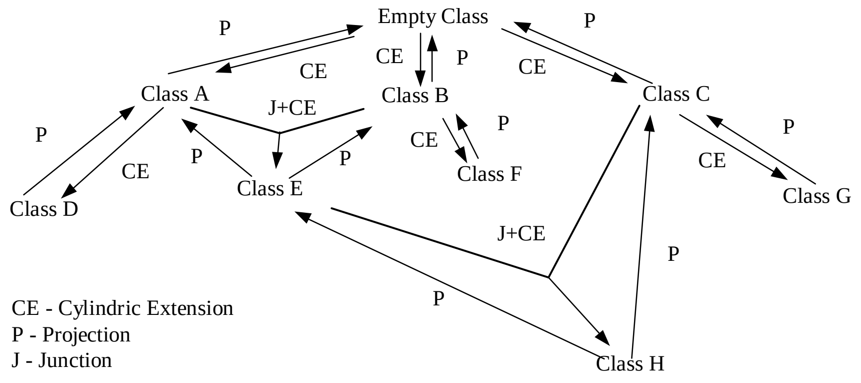

Figure 3 - Example of Class Hierarchy

Definition 18 - **Sub-object:** Let c be an object of class C and d an object of class D a superclass of C. If for any n the instance of d is a sub-tuple of the instance of c, then d is a sub-object of c.

In other words, d is the free projection of c in $N \times D$, i.e. $d = c \downarrow N \times D$.

Definition 19 - **Active and Passive Objects:** An object c of a class C is called an active object if $m > 0$. If $m = 0$, then c is a passive object.

Definition 20 - **Input Interface:** Let c be an active object of a class C and I a superclass of C, defined by:

$$I = \bigtimes_{i} V_i, \forall i \in \{1,...,n\} \text{ such that} \begin{cases} \exists f_j,\ 1 \leq j \leq m \text{ where } i \in P_j \\ \forall f_l,\ 1 \leq l \leq m \quad i \notin Q_l \end{cases}$$

The input interface i of the object c is the passive object generated by the free projection of c in $N \times I$, i.e. $i = c \downarrow N \times I$.

Definition 21 - **Function Specific Input Interfaces:** Let c be an active object of class C, i its input interface, and I^j a superclass of I and C such that :

$$I^j = \bigtimes_i V_i, \forall i \in \{1,...,n\} \text{ such that, for } f_j, i \in P_j \text{ and } \forall l \in \{1,...,m\}, i \notin Q_l$$

The specific input interface for function j of c, i^j, is the free projection of c in $N \times I^j$.

Note that $i^j = c \downarrow N \times I^j = i \downarrow N \times I^j$. If the elements of class C have m functions, then there exist m function specific input interfaces. Each i^j is a sub-object of i and c.

Definition 22 **- Output Interface:** Let c be an active object of class C and O a superclass of C characterized by:

$$O = \bigtimes_i V_i, \forall i \in \{1,...,n\} \text{ such that} \begin{cases} \exists f_j, 1 \leq j \leq m \text{ where } i \in Q_j \\ \forall f_l,\ 1 \leq l \leq m \quad i \notin P_l \end{cases}$$

The output interface o of object c is the passive object generated by the free projection of c in $N \times O$, i.e. $i = c \downarrow N \times O$.

Definition 23 **-Function Specific Output Interfaces:** Let c be an active object of a class C, o its output interface, and O^j is a superclass of O and C such that :

$$O^j = \bigtimes_i V_i, \forall i \in \{1,...,n\} \text{ such that for } f_j\ i \in Q_j \text{ and } \forall l \in \{1,...,m\}, i \notin P_l$$

The output interface specific for function j of c, o^j, is the free projection of c in $N \times O^j$.

Clearly, $o^j = c \downarrow N \times O^j = o \downarrow N \times O^j$ and if the elements of class C have m functions, then there exist m function specific input interfaces. Each o^j is a sub-object of o and c.

Definition 24 **- Existence of Objects:** An object c is said to exist at n if the function which maps instances of c in C is defined for $n \in N$.

2.4 Interaction Among Objects

After formally defining the concepts involving objects, we now proceed to define the concepts necessary to provide objects interaction.

Definition 25 **- Generation and Destruction of Objects:** An object is generated at n if it does not exist at n and does exist at n+1. An object is destroyed (destructively assimilated) at n if it does exist at n but does not exist at n+1.

Definition 26 - **Enabling Scope of Functions:** Consider an active object c from a class C = { $(v_1, v_2, ..., v_n, f_1, f_2, ..., f_m)$}, a function f_j of C, and i^j an input interface specific for function f_j . Let β be the arity of instances of i^j , gi an input index function for f_j , gi : {1, ... , β } $\rightarrow$ {1, ... , n} mapping each component from instances of the input interface specific for f_j to a component in instances of c, and B = {0,1}. An enabling scope for f_j is a set of tuples H = {(h_t, b_t)}, t = 1, ... , β, where h_t is an object of class $V_{gi(t)}$ and $b_t \in$ B is a boolean value indicating if object h_t should ($b_t = 1$) or should not ($b_t = 0$) be destroyed when c triggers.

Definition 27 - **Generative Scope of Functions:** Assume an active object c of class C = { $(v_1, v_2, ..., v_n, f_1, f_2, ..., f_m)$}, a function f_j of C, an output interface o^j specific or function f_j , α the arity of instances of o^j , and an output index function for f_j, go : {1, ... , α } $\rightarrow$ {1, ... , n} mapping each component from instances of output interface specific for f_j to a component in instances of c. A generative scope for f_j is a set of objects S = {s_u }, u = 1, ... , α , where s_u is an object of class $V_{go(u)}$.

Definition 28 - **Enabling of an Active Object:** An active object of class C is enabled at n if all objects belonging to an enabling scope of one of its functions f_j do exist at n. Function f_j is said to be enabled at n.

Definition 29 - **Triggering of an Active Object:** Consider the following:

a)-an object c from a class C,

b)- the instance of c at n, $c(n) = (v_1(n), ..., v_n(n), f_1(n), ..., f_m(n))$,

c)-a function f_j of c at n, enabled by H = {(h_t, b_t)},

d)-a generative scope S = {s_u } for f_j such that if $s \in$ S, then or s does not exist at n, or $s \in$ H,

e)-p, the number of values such that $k \in P_j$, k = 1, ... , n, (P_j from def. 14)

f)-a domain index function gd : (1, ... , p } $\rightarrow$ {1 , ... , n} for f_j ,

g)-the projection of f(.) on V_k , $f(.)\downarrow V_k$,

h)-β, α, gi e go as before.

Triggering an active object at n means:

1)- determination of c's instance at instant n+1, given the instance of c at n and the instances of h_t at n:

$$v_i(n+1) = \begin{cases} v_i(n) & \text{if } i \notin P_j \text{ and } i \notin Q_j \\ h_{gi^{-1}(i)}(n) & \text{if } i \in P_j \text{ and } i \notin Q_j \\ f_j(w_1,...,w_b)\downarrow V_i & \text{if } i \in Q_j \end{cases}$$

$$\text{where } w_r = \begin{cases} h_{gi^{-1}(gd(r))}(n), & \text{if } gd(r) \in P_j \\ v_{gd(r)}(n), & \text{if } gd(r) \notin P_j \end{cases}$$

2)- destruction ($b_t = 1$) of object h_t, $(h_t, b_t) \in H$,at n,

3)-generation, at n , of the objects of S those do not exist at n,

4)-determination of the instances of the objects of S, at n+1, according to the instances of c at n+1:

$$s_u(n+1) = v_{go(u)}(n+1)$$

2.5 Object Systems

Knowing what objects are and which mechanisms allow objects interaction we can define an object system.

Definition 30 - **Object System:** Assume the following:

a)-c_i are objects of class C_i , i = 1, ... , δ ,

b)-$\mathcal{C} = \bigcup_i c_i$,

c)-$\Theta_i = \{ 0, \ldots , m_i \}$ where m_i is the number of functions for object c_i ,

d)-B = {0,1},

e)-γ_i , $0 \le i \le \delta$, $\delta > 0$, are selection functions $\gamma_i : N \rightarrow 2^{\mathcal{C} \times B} \times 2^{\mathcal{C}} \times \Theta_i$ which, for each object c_i at n, select an enabling scope H_i , a generative scope S_i and a function index such that $\forall (c,b) \in H_i$, if b = 1, then $(\forall k \neq i)((c,1) \notin H_k)$, $\forall c \in S_i$, $(\forall k \neq i)(c \notin S_k)$ and $(\forall k)((c,1) \notin H_k)$. H_i is an enabling scope and S_i is a generative scope for function $\gamma_i(n) \downarrow \Theta_i$. If c_i is a passive object or, at n $\nexists H_i \neq \emptyset$ or $\nexists S_i \neq \emptyset$, then $\gamma_i(n) = (\emptyset, \emptyset, 0)$. The third index is null to indicate that no function is to be executed. The meaning of these conditions are grasped as follows. An object of an enabling scope programmed to be destroyed must be destroyed by only one active object. If an object is part of a generative scope of an active object, then it cannot be in any other generative scope. If the object in question is passive, or if active it does not have an enabling scope for any of its functions, then its selection function must do nothing.

An object system Ω is a set of pairs $\{\omega_i\}$, $\omega_i = (c_i, \gamma_i)$, such that :

1)-for n=0, there exists at least one ω_i with an object c_i defined,

2)-for n>0 all active objects c_i with $\gamma_i(n) \neq (\emptyset, \emptyset, 0)$, i.e. objects whose selection functions are in the form $\gamma_i(n) = (H_i, S_i, j)$ may trigger according to its enabling scope H_i and the generative scope S_i , using its j-th internal function, 3)-for $n>0$, all objects c_i which exist at n and do not have its instance at (n+1) determined by item 2, may have $c_i(n+1) = c_i(n)$.

The definition of an object system may be viewed from two different perspectives. In the first, it provides a recursive way of building an object system. In the second view, comprising a given collection of objects, it works as a specification. To call a given set of objects an object system, the values of its instances (and their existence at different times) should be associated with each other according to the laws of triggering and regeneration of objects given by items 2 and 3 of definition 30. These laws determine, at each time instant, the values of the instances (or its inexistence) based on their previous values. The objects that interact at each instant are determined according to the selection functions, which define the enabling scopes, the generative scopes, and the functions to be triggered for each active object. Therefore, the selection function plays a fundamental role in the object system dynamics.

Objects are functions and there are, in principle, no restrictions on their form. For a set of objects the functions can be any. An object system has, as opposed to a set of objects, a clear recursive nature. The recursiveness is not simple because objects may be *partial* functions, i.e. they do not have to be defined for any value of its domain. This leads to the fundamental question about the computability[8,9] of object systems, certainly a desirable property. An object system being computable means that we can determine, for each $n \in N$, the values of the instances for the existing objects at n. Conditions for computability of object systems are given as follows. Assume Ω an object system with a finite number of elements ω_i , with all its selection functions γ_i computable, and all internal functions of all objects c_i of ω_i computable. Then Ω is computable.

These are sufficient conditions only. An object system does not need to necessarily have a finite number of objects to be computable. If for adjacent instants n and n+1 the number of existing objects is finite, then the system is computable.

An object system fulfilling the previous conditions can be viewed as the limit of a (possibly infinite) sequence of objects systems Ω_1 , Ω_2 , ... , each Ω_i with a finite number of objects defined on a finite and incremental domain N_i. That is: $N_0 = \{0\}$, and $N_i = N_{i-1} \cup \{i\}$.

Consequently, each object system Ω_i is computable and the whole sequence is computable as well. The infinite object is the i-th element of this sequence when $i \rightarrow \infty$. Hence, the object system is computable, although infinite.

2.6 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[10,11] 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 4.

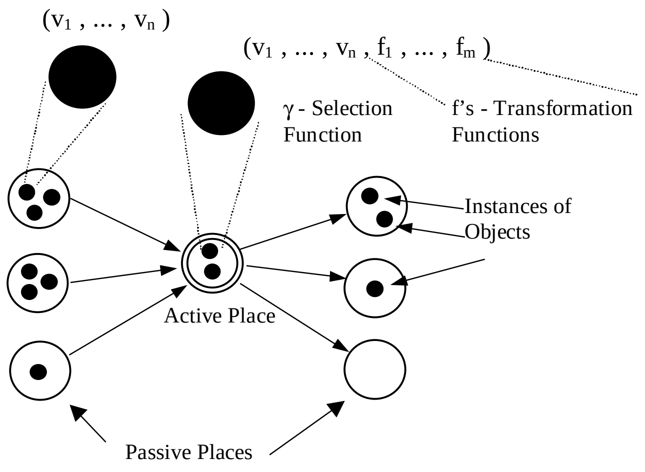

Figure 4 : Example of an Object Network

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. 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.

Definition 31 - **Object Network:** Assume the following:

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

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

c)- $\Pi = \{ \pi_i \}$ a set of places π_i

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

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

f)-ξ a 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.

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

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

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

j)-$\Xi = \Xi(\pi)$ 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 $\Xi(\pi_k) = C$.

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

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

m)-∂_i the number of fields in i^i and ρ_i the number of fields in o^i.

n)-the function fpi_i the attribute function for input gates of objects which are at a place π_i, defined $fpi_i : \{1, \dots, \partial_i\} \rightarrow A$ and $fpi = \{fpi_i\}$.

o)-$fpo_i = \{1, \dots, \rho_i\} \rightarrow A$ the attribute function for output gates of objects that are at a place π_i and $fpo = \{fpo_i\}$

p)-$\Theta_i = \{0, \dots, m_i\}$, where m_i is the number of function for object c_i

q)-$\gamma = \{\gamma_i\}$, $0 \leq i \leq \delta$, $\delta > 0$, which elements are selection functions $\gamma_i : N \rightarrow 2^{\mathcal{C} \times B} \times 2^{\mathcal{C}} \times \Theta_i$ that for each object c_i, in an instant n, select an enabling scope H_i, a generative scope S_i and the index for the function to be executed by the object, having as a restriction 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 $(\forall k)((c,1) \notin H_k)$. More than that, H_i should be an enabling scope and S_i should be a generative scope for function f_k, $k = \gamma_i(n) \downarrow \Theta_i$. 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 function is going to be executed. Those conditions are analogous to the selection function for an object system, added by the following conditions: Any object belonging to the enabling scope of another object should be connected to the place where the active object is by an input arc. Any object belonging to the generative scope of an active object should be put in a place that is connected to the place where the active object is by means of an output arc.

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 $\gamma_i \in \gamma$, $0 \leq i \leq \delta$,

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 $\gamma_i(n)$ 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.

Alike an object system, an object network can be seen as a specification comprising the trajectories over time for a given set of objects. In the same way, an important class of object networks are those that are computable. Again, we can determine a computable object network iteratively, generating a sequence of object networks $\Re_0$, $\Re_1$, ... , where each $\Re_i$ contains a finite number of objects, defined over incremental domains N_i . Each object network from the sequence is equivalent to the previous, being its objects added by a temporal instance, according to the laws of triggering, the selection function defined for the last instance and the localization function. The procedure is similar to the one used for objects systems with infinite number of objects, included the conditions for localization functions. With this methodology, one can determine, for each instant $n \in N$, the instance of any object that exists in n. The initial objects network from the sequence, $\Re_0$ has a special denomination, being called the kernel of an object network. In its kernel, objects and localization functions are defined only for instance n=0, and the selection function should be computable, being described by an algorithm γ'.

Definition 32 - **Kernel of an Object Network:** We define the kernel of an object network as an objects network $\Re_0 = (\Sigma, \Pi, \Xi, A, \eta , fpi, fpo, \mathcal{C}^0 , \xi^0 , \gamma)$, where

- $\Sigma, \Pi, \Xi, A, \eta$, fpi and fpo are as in the Definition of objects network and
- $\mathcal{C}^0 = \{c_i'\}$ is a set of objects defined only for n=0.
- ξ^0 is a localization function $\xi^0 : N \times \mathcal{C}^0 \rightarrow \Pi$, defined only for n=0.
- γ is a selection function determined by means an algorithm γ'.

Starting with a kernel, new finite object networks are computed in sequence. Each algorithm step generates a new network that is, fundamentally, the previous one increased by the definition of new values for $\mathcal{C}$, ξ and γ . The new $\mathcal{C}$ is the old one plus the objects to be generated at step n, according to γ. The new ξ is also the old one, defined now also for instant n, again according to γ. The algorithm γ' incrementally defines function γ. At each step the algorithm defines new instances for the current objects and new values for localization function and selection function such that for each instant n, we obtain new $\mathcal{C}$, ξ and γ corresponding to a new finite objects network. Finite networks correspond to some element of such a sequence. Infinite networks (both networks with a finite number of objects, defined in infinite domains, or networks with an infinite number of objects), are considered as the limit element i of the sequence, when $i \rightarrow \infty$. An example of such algorithm is described in figure 5. Note that each new network in the sequence includes the previous one increased by the definition of a new time instant.

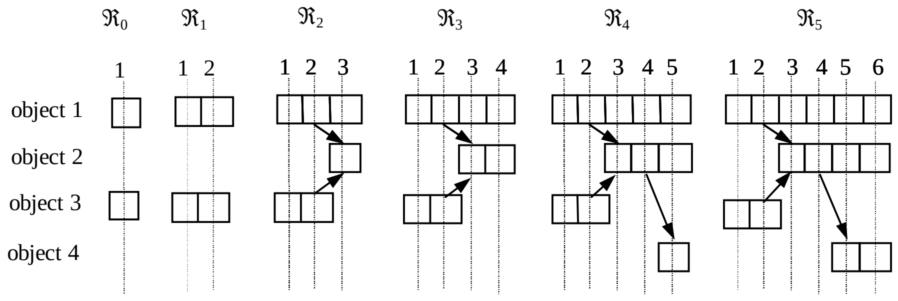

Figure 5 - Example of an Evolution Sequence

Examples of main algorithms described through pseudo-codes are the following:

```
procedure Main:
{Define $\mathcal{C}$, composed by objects $c_i$ given in $\mathcal{C}^0$.
 Define localization function $\xi$, composed by the tuples in $\xi^0$ .
 Define $\gamma = \varnothing$
 For n from 0 to final
    {Apply $\gamma'$ to determine $\gamma(n)$ and refresh $\gamma$.
     For all active objects $c_i$ existing at n:
        {Calculate $\gamma_i(n) = (H_i, S_i, f)$.
         If $H_i = \varnothing$, go to next object
         If $H_i \neq \varnothing$ :
            {execute function f, generating a new instance $c_i(n+1)$
             refresh $c_i$ : $c_i = c_i(n) \cup c_i(n+1)$.
             For all $s_i^k \in S_i$
                {If $s_i^k \notin \mathcal{C}$ generate a new empty object and add to $\mathcal{C}$
                 calculate the value for $s_i^k(n+1)$ from $c_i(n+1)$ and refresh $s_i^k$ .
                 determine $\xi(n+1, s_i^k) \in V(\xi(n, c_i))$ and refresh $\xi$.
                }
            }
        }
    For all objects $c_i$ such that $(c_i, 1)$ does not appear at any other enabling scope
and $c_i$ is not at     neither generative scope
        $c_i(n+1) = c_i(n)$.
    }
}
```

```
Procedure $\gamma$'
{For each active object $c_i$
    {For each function $f_j$ of active object $c_i$
        {Generate a new empty enabling scope for function $f_j$
         For each field k of input interface specific to $f_j$
            {check if in the place pointed by the arc fpo(k) does exist:
                no object, one object or  more than one object
             If there is no object, delete the enabling scope(s) and go to next function
             If there is only one object, put it into the enabling scope(s).
             If there is more than one object, do as many copies of the enabling scope
               as the number of  objects and put one object into each enabling scope.
            }
         For each enabling scope calculate a performance index
        }
    }
  For each active object $c_i$
    {Do a list ordered by the performance index containing the function and the
      enabling scope.
    }
  For each active object $c_i$
    {Choose the first element in the list as the function and enabling scope for the
        object
     Check if is there any conflict with other objects choices.
     If is there a conflict, use a decision criterion. The looser object chooses than the
       next in its list. If the own object $c_i$ belongs to the enabling scope of another
       object, cancel its enabling scope and reorganize the list.
    }
  For each active object $c_i$ with an enabling scope different from empty
    {Create an empty generative scope for $c_i$
     For each field k of output interface specific to function $f_j$ chosen
        {If is there an object in $\mathcal{C}$ not defined at n-1 and n for the desired class,
            put it into the generative scope, else create a new object for the desired
            class and include it.
        }
    }
   Returns, for each object, the enabling scope, the generative scope and the function
    chosen
}
```

2.7 Extensions

To enhance its representation power, additional definitions are developed to allow different classes of concepts. Those concepts are used mainly within computational semiotics theory[4,5,6,7]. They are presented next.

Definition 33 - **Temporal Restriction for Objects:** Let N be a set of time instants, S a class and $o:N \to S$ an object of type S. Let $N' \subseteq N$. The temporal restriction of object o to N', denoted by $o \Downarrow N'$ corresponds to the object $o':N \to S$ such that if $(n,s) \in o$ and $n \in N'$, $(n,s) \in o'$. Otherwise, $(n,s) \notin o'$.

Example: $N = \{ n_1 , n_2 , n_3 \}$, $S = \Re^3$, $o = \{ (n_1 , (0,0,0)) , (n_2 , (0,1,0)) , (n_3 , (1,2,2)) \}$.

$N' = \{ n_2 , n_3 \} \to o' = \{ (n_2 , (0,1,0)) , (n_3 , (1,2,2)) \}$.

Definition 34 -**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 $x : N \to 2^X$.

Examples: Let $N = \{1,2,3\}$ and $X = \{1,2,3,4\}$. An example of a set variable x of type X is :

$x = \{ (1, \{1,2\}) , (2, \{2,3,4\}) , (3, \{1,3\}) \}$

Assume now $X^2 = X \times X$. Thus, a set variable x' of type X^2 is, for example,

$x' = \{ (1, \{ (1,2),(2,3),(2,4),(3,3) \}) , (2, \{(2,3),(4,1),(1,1)\}) , (3, \{(1,3),(2,1) \}) \}$

Note, in the last example, that the value of x for each $n \in N$ corresponds to a relation. In this case, if we set $R_1 = \{ (1,2),(2,3),(2,4),(3,3) \}$, $R_2 = \{(2,3),(4,1),(1,1)\}$ and
$R_3 = \{(1,3),(2,1) \}$, we get, for short, $x' = \{ (1,R_1) , (2,R_2), (3, R_3) \}$.

Definition 35 - **Generic Object:** Let C be a non-empty class. Let c be a set variable of type C. The variable c is called a **generic object** of class C.

Definition 36 - **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 37 - **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 $x : N \to 2^{\tilde{X}}$.

If X is a passive class, $X = X_1 \times ... \times X_m$, $\tilde{X}$ will be, in general, an m-ary fuzzy relation. In some cases, it is interesting to use not a generic fuzzy relation, but a fuzzy relation formed by the cartesian product of different fuzzy sets. In this case, $\tilde{X}$ may be represented by a tuple of m fuzzy sets. If X is an active class, $\tilde{X}$ would consider as fuzzy only the fields that are not functions. The m-ary fuzzy relation, in this case, will be represented by the (fuzzy) cartesian product of all elements that are not functions.

Note that a fuzzy object can represent any (standard) object $o = \{ (n,x) \mid \forall n \in N, x \in X\}$ if we take, for each $n \in N$, a fuzzy set that is a singleton in $x \in X$.

Since a passive object corresponds to a fuzzy relation, it is possible to define operations involving fuzzy objects. The same occurs with active objects, because the operations are related with non-function fields only.

Definition 38 **- Union of Fuzzy Objects:** Let x' and x'' be two fuzzy objects of type X, defined in N such that $\forall n \in N$, if x'(n) is defined, x''(n) is also defined. The union x of x' and x'' is a fuzzy object such that $\forall n \in N$, $x(n) = x'(n)\ \mathsf{S}\ x''(n)$, where S is a matrix operator which applies a triangular co-norm, element to element, in the m-ary relational matrices. Operator S applies only to non-function fields of the corresponding tuples.

Definition 39 **- Intersection of Fuzzy Objects:** Let x' and x'' be two fuzzy objects of type X, defined in N, such that $\forall n \in N$, if x'(n) is defined, x''(n) is also defined. The intersection x of x' and x'' is a fuzzy object such that $\forall n \in N$, $x(n) = x'(n)\ \mathsf{T}\ x''(n)$, where T is a matrix operator which applies a triangular norm, element to element, in the m-ary relational matrices. Again, operator T applies only to the non-function fields of the tuples.

For the sake of computational implementation, it is important to stress that both generic objects and fuzzy objects can be transformed into objects. Suppose that the values of their instances (which are sets or fuzzy sets) are represented by discriminating functions, functions that are approximated by multilayer neural networks. We can generate a standard object whose attributes are the parameters of such neural network. In this case, we represent a generic object (or a fuzzy object), by a standard object. To use a generic knowledge in this form, however, the arguments used to manipulate it must be modified to deal with generic/fuzzy objects in such a representation.

Definition 40 - **Meta-Object:** Let N be an enumerable set, with a generic element n; V be an enumerable set where each $v \in V$ is a variable of type N defined over the occurrence space T, $v : T \rightarrow N$; R be a set of restrictions for the values of variables V (possibly empty) and X be a class. A **meta-object** x of type X is a function $x : V \rightarrow X$.

Examples : Let T = { 1,2, ... } , N = {1,2,3,4,5,6}, V = { v_1 , v_2 , v_3 }, such that v_1 , v_2 , v_3 : $T \rightarrow N$, R = ∅, and $X = X_1 \times X_2$, X_1 = {1,2,3,4}, X_2 = {a,b,c}. An example of a meta-object x' of type X_1 is x' = { (v_1 , 1) , (v_2 , 3) }. Other example of a meta-object x'' of type X is x'' = { (v_1 , (1,a)) , (v_2 , (3,a)) }.

Definition 41 - **Instance of a Meta-Object:** Let x be a meta-object of type X. An **instance** of x is an object x' where the variables of x are substituted for the values provided by specific instances of such variables in the occurrence space.

Examples: Consider the meta-objects x' and x'' as in the example above. An instance of x', doing v_1 = 1 and v_2 = 2 is x''' = { (1 , 1) , (2 , 3) }. Other example, for v_1 = 2 and v_2 = 5, x''' = { (2 , 1) , (5 , 3) }. An instance of x'', for v_1 = 1 and v_2 = 4 is x''' = {(1 , (1,a)), (4 , (3,a))}.

Definition 42 - **Occurrence of a Meta-Object in an Object:** Consider an object o from class X, and a meta-object o' from class X'. An occurrence o'' of meta-object o' in o is an object o'' such that o'' is at the same time a sub-object from an instance of o' and a temporal restriction of a sub-object of o.

Examples: Assume an object x of type X, x = { (1,(1,a)) , (2,(3,b)) , (3,(3,a)), (4,(1,c)) , (5,(2,b)), (6,(3,a)). As in the example above, for v_1 = 1 and v_2 = 2, we have an instance x''' = { (1,1), (2,3) } which is a temporal restriction of sub object $x \downarrow X_1$ of x to N = {1,2}. This is an occurrence of x' in x. Other occurrences for this case do exist, for v = (v_1 , v_2) = (1,3), v = (1,6), v = (4,6). Not so obvious is the case for v = (4,2) and v = (4,3). Meta-object x'' does also occur in x, but there exist only two occurrences, for v = (1,3) and v = (1,6) respectively.

When the restriction R for a meta-object is not empty, domain variables may have restrictions to its values. One way of representing restrictions is by means of algebraic equations and/or inequations using domain variables. In this case the occurrence of meta-objects in objects does consider such restrictions for the determination of possible instances for the meta-object.

Example: In the example above, assume that $v_2 = v_1 + 1$. In this case, there exists only one occurrence of x' in x for v = (1,2), because the restriction is violated for the other cases. Note that here, the meta-object x'' does not occur in x.

Other example could be $v_2 > v_1$. Viewing the inequality as a restriction in the example of Definition 6, we avoid non-intuitive cases v = (4,2) and v = (4,3) as occurrences.

Following the definition, an occurrence does not need to be, necessarily, a meta-object instance, but any sub-object of the former. With this, from the same meta-object there may be different occurrences in different objects, with each object from a different class, but sharing a field of the instance of the meta-object.

Definition 43 - **Occurrence of a Meta-Object in a Generic Object:** Let x be a generic object from class X, and x' a meta-object from class X'. An occurrence x'' of the meta-object x' in x is an object x'' such that x'' is an occurrence for any case of x.

Definition 44 - **Occurrence of a Meta-Object in a Fuzzy Object:** Let o be a fuzzy object from class X, and o' a meta-object from class X'. An occurrence o'' of the meta-object o' in o is a fuzzy object o'' such that o'' corresponds to the intersection of a sub-object from an instance of o', described as a fuzzy object by means of singletons, and a temporal restriction of a fuzzy sub-object of o.

Example: Consider the following fuzzy sets
a_1 = {1/0.2, 2/0.8, 3/0.6 }, a_2 = {1/0.1, 2/0.2, 3/0.9 }, a_3 = {1/0, 2/0.15, 3/0.3 },
b_1 = { 5/0.3, 6/0.4, 7/0.1 }, b_2 = { 5/0.4, 6/0.4, 7/0.8 }, b_3 = { 5/0.1, 6/0.9, 7/0.8 },
c_1 = { 15/0.2, 18/0.9 }, c_2 = { 15/0.3, 18/0.8 }, c_3 = { 15/0.7, 18/0.1 }, the fuzzy object x = { (1,(a_1,b_1,c_1)), (2,(a_2 ,b_2 ,c_2)), (3,(a_3 ,b_3 ,c_3)), and the meta-object x' = { (v_1 , (2,5,15)) , (v_2 , (3,7,18)) }. For $v_1 = 1$ and $v_2 = 3$, there is an instance of meta-object x' which is x'' = { (1 , (2,5,15)) , (3 , (3,7,18)) }. Thus, for a'_1 = {1/0, 2/1, 3/0 }, b'_1 = {5/1, 6/0, 7/0 } and c'_1 = {15/1, 18/0 }, a'_2 = {1/0, 2/0, 3/1 }, b'_2 = {5/0, 6/0, 7/1 } and c'_2 = {15/0, 18/1 } we have the representation of x'' by the fuzzy object x''' = { (1, (a'_1 , b'_1 , c'_1)) , (3, (a'_2 , b'_2 , c'_2)) }. An occurrence of x' in x, in this case, can be found to be x'''' = (x $\Downarrow$ {1,3}) T x''', i.e., x'''' = { (1, (a''_1 , b''_1 , c''_1)) , (3, (a''_2 , b''_2 , c''_2)) }, where, taking the minimum as triangular norm, we have: $a''_1 = a_1$ T a'_1 = {1/0, 2/0.8, 3/0 }, $b''_1 = b_1$ T b'_1 ={5/0.3, 6/0, 7/0 }, $c''_1 = c_1$ T c'_1 = {15/0.2, 18/0 }, $a''_2 = a_3$ T a'_2 = {1/0, 2/0, 3/0.3 }, $b''_2 = b_3$ T b'_2 = {5/0, 6/0, 7/0.8 }, $c''_2 = c_3$ T c'_2 = {15/0, 18/0.1 }.

Definition 45 - **Generic Meta-Object:** Let N be an enumerable set, with generic element n, V be an enumerable set, where each $v \in V$ is a variable of type N, R be a set of restrictions on the variables of V (possibly empty) and X be a class. A **generic meta-object** x of type X is a function $x : V \rightarrow 2^X$.

Definition 46 - **Case of a Generic Meta-Object:** Let x be a generic meta-object from class X. A meta-object x' of type X is a case of generic meta-object x if $\forall v \in V, x'(v) \in x(v)$.

Definition 47 - **Occurrence of a Generic Meta-Object in an Object:** Assume x as a generic meta-object of type X and x' an object of type X'. An occurrence x'' of x in x' is an object x'' such that x'' is an occurrence of any case of x in x'.

Definition 48 - **Occurrence of a Generic Meta-Object in a Generic Object:** Let x be a generic meta-object of type X and x' a generic object of type X'. An occurrence x'' of x in x' is a generic object x'' such that x'' is the union of all occurrences of any case of x in cases of x'.

Definition 49 - **Occurrence of a Generic Meta-Object in a Fuzzy Object:** Let x be a generic meta-object of type X and x' a fuzzy object of type X'. An occurrence x'' of x in x' is a fuzzy object x'' such that x'' is the intersection of a sub-object of x, viewed as fuzzy object, and a temporal restriction of a sub-object of x'.

Example: Consider the fuzzy sets a_1 = {1/0.2, 2/0.8, 3/0.6 },
a_2 = {1/0.1, 2/0.2, 3/0.9 }, a_3 = {1/0.1, 2/0.15, 3/0.3 }, b_1 = { 5/0.3, 6/0.4, 7/0.1 },
b_2 = { 5/0.4, 6/0.4, 7/0.8 }, b_3 = { 5/0.1, 6/0.9, 7/0.8 }, c_1 = { 15/0.2, 18/0.9 },
c_2 = { 15/0.3, 18/0.8 }, c_3 = { 15/0.7, 18/0.1 }, the fuzzy object x = { (1,(a_1,b_1,c_1)),
(2,(a_2,b_2,c_2)), (3,(a_3,b_3,c_3)), and the generic meta-object x' = { (v_1,([2,3],[5,6],15)),
(v_2, ([1,2],[6,7],18)) }. For $v_1 = 1$ and $v_2 = 3$, we have an instance of the generic meta-object x', x'' = { (1 , ([2,3],[5,6],15)) , (3 , ([1,2],[6,7],18)) }. Thus, for
a'_1 = {1/0, 2/1, 3/1 }, b'_1 = {5/1, 6/1, 7/0 } and c'_1 = {15/1, 18/0 },
a'_2 = {1/1, 2/1, 3/0 }, b'_2 = {5/0, 6/1, 7/1 } and c'_2 = {15/0, 18/1 } we get a representation of x'' by the fuzzy object x''' = { (1, (a'_1 , b'_1 , c'_1)) ,
(3, (a'_2 , b'_2 , c'_2)) }. An occurrence of x' in x, in this case, can be found through
x'''' = (x $\Downarrow$ {1,3}) $\top$ x''', i.e., x'''' = { (1, (a''_1 , b''_1 , c''_1)) ,
(3, (a''_2 , b''_2 , c''_2)) }, where, taking again the minimum as a triangular norm, we have: $a''_1 = a_1 \top a'_1$ = {1/0, 2/0.8, 3/0.6 }, $b''_1 = b_1 \top b'_1$ ={5/0.3, 6/0.4, 7/0 },
$c''_1 = c_1 \top c'_1$ = {15/0.2, 18/0 }, $a''_2 = a_3 \top a'_2$ = {1/0.1, 2/0.15, 3/0 },
$b''_2 = b_3 \top b'_2$ = {5/0, 6/0.9, 7/0.8 }, $c''_2 = c_3 \top c'_2$ = {15/0, 18/0.1 }.

Definition 50 **- Fuzzy Meta-Object:** Let N be an enumerable set, with a generic element n, V be an enumerable set, where each $v \in V$ is a variable of type N, R be a set of restrictions over the variables in V (possibly empty), X a class, $\tilde{X}$, a fuzzy set defined over X and $2^{\tilde{X}}$, the set of all fuzzy sets defined on X. A **fuzzy meta-object** x of type X is a function $x : V \to 2^{\tilde{X}}$.

Definition 51 **- Occurrence of a Fuzzy Meta-Object in an Object:** Let x be a fuzzy meta-object of type X and x' an object of type X'. An occurrence x'' of x in x' is a fuzzy object x'' such that x'' is the intersection of a sub-object of x and a temporal restriction of a sub-object of x' described as a fuzzy object.

Definition 52 **- Occurrence of a Fuzzy Meta-Object in a Generic Object:** Let x be a fuzzy meta-object of type X and x' a generic object of type X'. An occurrence x'' of x in x' is a fuzzy object x'' such that x'' is the intersection of a sub-object of x and a temporal restriction of a sub-object of x' described as a fuzzy object.

Definition 53 **- Occurrence of a Fuzzy Meta-Object in a Fuzzy Object:** Let x be a fuzzy meta-object of type X and x' a fuzzy object of type X'. An occurrence x'' of x in x' is a fuzzy object x'' such that x'' is the intersection of a sub-object of x and a temporal restriction of a sub-object of x'.

Note that the definitions of occurrence of fuzzy meta-objects are the same, wherever being in an object, generic object or fuzzy object. In all cases, the representation of object or generic object is first converted to a fuzzy object, and next the definition for an occurrence of a fuzzy meta-object in a fuzzy object is used.

In our examples of occurrences, we showed only meta-objects with two variables, i.e., $V = \{v_1 , v_2 \}$. But this is not a requirement. The use of meta-objects with two variables is particularly useful to represent events, i.e. sudden changes between two states. But other types of occurrences do exist, as e.g. an unitary occurrence that represents a single state. This kind of occurrence is useful, e.g., to obtain information about some attribute of the object during its existence. Meta-objects with more than 2 variables are used to represent tendencies or behaviors as, e.g., an increasing or decreasing behavior, or a periodic behavior. The formal model herein stated allows the representation of occurrences with all those subtle differences.

To use meta-objects (and its generic and fuzzy extensions) in the framework of object networks, we have to convert them to objects. But first, all objects that are to be related to the meta-objects have to be modified to handle some kind of memory. This is showed in figure 6.

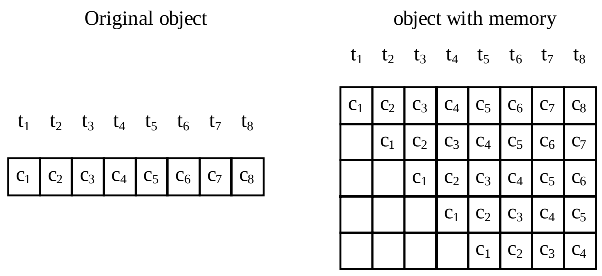

Figure 6 - Assembling Memory in an Object

Once the related objects are equipped with memory, we can modify the meta-object to become an object. This transformation is shown in figure 7.

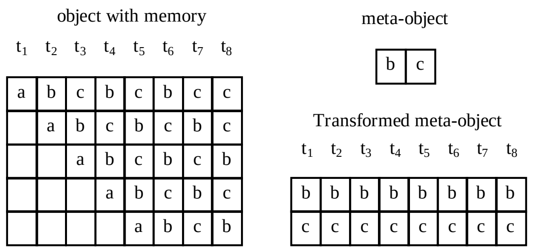

Figure 7 - Transformation of a meta-object into an object

Now, to check the occurrence of a meta-object in an object (and its variations), we use an active object that is fed both with the modified meta-object and the object with memory. Its active function is responsible for doing the respective computations.

In real systems, object memory can not be infinite. An alternative strategy for this problem is to use hierarchical memory using occurrence knowledge to generate a higher level memory. An example is shown in figure 8.

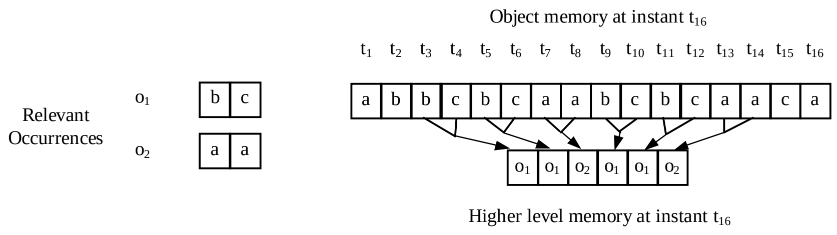
Object memory at instant t_{16}
Relevant Occurrences
Higher level memory at instant t_{16}

Figure 8 - Composition of a Higher Level Memory

As it can be seen in the example, a higher level memory has a smaller size. This may lead to information loss because higher-level memory stores only the occurrences considered to be relevant. The procedure to generate higher level memories can be successively applied, creating many levels, in a complex hierarchy.

3 Computational Intelligence and Object Nets

In this session we provide examples to show how we can implement and computational intelligence[12,13] techniques with object networks.

From computational semiotics[4,5,6,7], we know that there are three main functions used to build intelligent systems: deductive, inductive and abductive functions, also called arguments. Arguments are described by means active objects in an object network.

3.1 Fuzzy Systems

A general fuzzy production system is shown in figure 9.

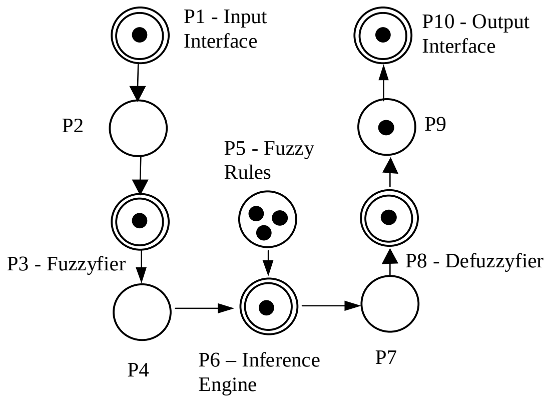

Figure 9 – Fuzzy System

The main argument in a fuzzy production system is the deductive argument in P6, called the inference engine. The other arguments in P1, P3, P8 and P10 are also deductive arguments, but their only task is to convert knowledge into a suitable format.

3.2 Neural Networks

Figure 10 shows the representation of a self-organizing neural network by an object network. Note that this neural network has two main arguments. The deductive argument works during the feedforward phase when it generates an output from an input. The learning function is an inductive argument that performs the self-organization of the neural network. It acts considering the input, and modifying the neural structure (i.e., the weights and offsets of the neural net) to perform learning.

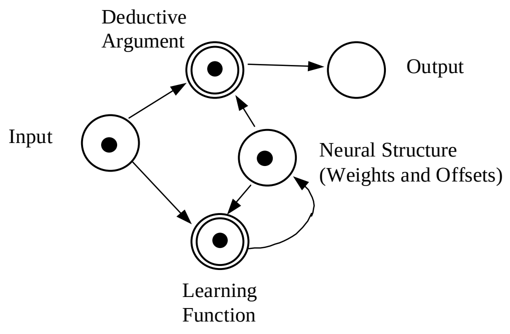

Figure 10 – Self-Organizing Neural Network

Other types of neural networks (like supervised neural networks) would have a similar representation including, in this case, a place for the desired output feeding the learning function argument.

3.3 Evolutionary Systems

An example of an evolutionary system is given in figure 11. In this example, there is an original population used as input for 4 inductive arguments (performing crossover, mutation, inductive mutation, etc), generating new sub-populations. Those 4 new sub-populations are used, in conjunction with the original population to fed an abductive argument that will choose the new population (and destroy the old one). The best individual of this new population is extracted by an abductive argument to generate a solution, and the new population is redirected again to the beginning through a feedback deductive argument that simply moves the new population from the new population place to the original population place.

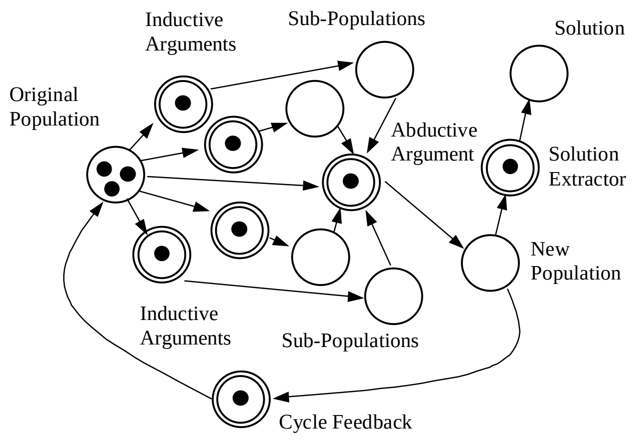

Figure 11 – Evolutionary System

3.4 Hybrid Systems

Soft computing techniques can be enlarged using not only its three main constituents (i.e., fuzzy systems, neural networks and evolutive systems), but (with object networks) hybrid systems can be constructed as well from the tools provided by the basic types of arguments associated with the three operations with knowledge: knowledge extraction, knowledge generation and knowledge selection.

4 Object Networks and Software Engineering

Object-oriented modeling approach has been extensively to specify, visualize, construct and document software, databases, business and many other systems. However, it is important to point out the difference between object-oriented programming and object-oriented modeling. Object-oriented programming basically involves adding inheritance and dynamic binding to programming. Object-oriented modeling comprises a software engineering methodology used to properly specificy complex software systems[14]. There are several object-oriented modeling methodologies proposed in the literature. Among them, we should mention the OMT[15] (Object Modeling Technique), one of the earliests to gain popularity and the UML[16] (Unified Modeling Language), derived from OMT and other technologies, which is being viewed as a new standard amongst software developers using object technology.

Most of the modeling techniques use a set of diagrams to capture the meaning of the system being modeled. For example, in figure 12, we summarize the diagrams used by UML. To each kind of diagram it is attached a corresponding semantics, used to specificy a part of the system.

Use case diagram		
Class diagram		
Behavior diagrams	Statechart diagram	
	Activity diagram	
	Interaction diagrams	Sequence diagram
		Collaboration diagram
Implementation diagrams	Component diagram	
	Deployment diagram	

Figure 12 – Diagrams used by UML

The choice of diagrams to be used in a system modeling has a profound influence on how a problem is approached and how a solution is developed. As a general tool, UML provides different types of diagrams to allow different types of systems to be specified.

Intelligent systems are a special kind of software systems, whose particularities make them very difficult to be modeled by generic methodologies like UML or OMT. These methodologies still lack diagrams to adequately manage some characteristics of inteligent systems. A strong limitation of these approaches concerns their use of a limited scope of the object-oriented paradigm, which is, the idea of a set of objects that exchange messages. Although being the usual meaning of what is an object system, it does not convey the whole expressiveness of the paradigm. The formalization of what is an object and an object system introduced in this paper attempts to surpass this limitation by considering messages also as objects. In this sense, our formalization subsumes the usual conceptualization, and embraces a more abstract view of what an object is. It allows a representation for adaptation, self-organization and parallelism, important issues when modeling intelligent behavior, which are not easyly tackled by generical modeling tools. When put together with higher order models of intelligence, like semiotics and computational intelligence, the object networks appear as a promising tool for modeling intelligent systems.

The design and implementation of an autonomous vehicle controller, example developed under this paradigm can be found in the work of Gudwin[4,6]. In this example, the task is to navigate a vehicle in an unknown environment, by interactively building an internal model for the environment, and using the model to make control decisions about the next actions to be performed.

5 Conclusion

The notion of object network has been introduced with the purpose to provide a modeling and implementation framework for computational intelligence. Beginning with a general and formal approach, put in the form of a foundation for a theory of objects, the idea of an object system was developed, followed by the key concept of object network. Object network has the formal and representational power needed to deal with intelligent systems. Several potential applications have been envisioned including control of autonomous vehicles using computational semiotics methodology[4,5,6].

Acknowledgements

The second author acknowledges CNPq, the Brazilian National Research Council for grant #300729/86-3.

References

1. A.Snyder, "The Essence of Objects:Concepts and Terms" - *IEEE Software*, pp. 31-42, Jan. (1993).
2. Y. Wand, "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, pp. 537-559, 1989).
3. M.I.Wolczko, - *Semantics of Object-Oriented Languages* – (Ph.D. Thesis - Technical Report Series UMCS-88-6-1 - Department of Computer Science, University of Manchester, Manchester M13 9PL, England, 1988).
4. R.R.Gudwin, *Contribuições ao Estudo Matemático de Sistemas Inteligentes* – Ph.D. Thesis - DCA-FEEC-UNICAMP, Maio (1996) (in portuguese)
5. R.R.Gudwin and F.A.C.Gomide, *Computational Semiotics : An Approach for the Study of Intelligent Systems - Part I : Foundations* – (Technical Report RT-DCA 09 – DCA-FEEC-UNICAMP, 1997).
6. R.R.Gudwin and F.A.C.Gomide, *Computational Semiotics : An Approach for the Study of Intelligent Systems - Part II : Theory and Application* – (Technical Report RT-DCA 09 – DCA-FEEC-UNICAMP, 1997).
7. R.R.Gudwin and F.A.C.Gomide, *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).
8. M. Davis, *Computability & Unsolvability* (McGraw-Hill Book Company, New York, 1958).
9. R.L.Epstein and W.A.Carnielli, *Computability : Computable Functions, Logic and the Foundations of Mathematics* (Wadsworth & Brooks/ Cole Advanced Books & Software - Pacific Grove, California, USA,1989).
10. K.Jensen, - "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).
11. T.Murata, - "Petri Nets : Properties, Analysis and Applications" - *Proceedings of the IEEE*, vol. 77, n. 4, April (1989).
12. L.Zadeh, – "Soft Computing and Fuzzy Logic", *IEEE Software*, vol. 11, n. 6, pp. 48-56, (1994).
13. J.Zurada, R.J.Marks II, and C.J.Robinson, - *Computational Intelligence - Imitating Life* – (IEEE Press, USA, 1994).
14. S. Sigfried – *Understanding Object-Oriented Software Engineering* – (IEEE Press, USA, 1995).
15. J.Rumbaugh, M.Blaha, W.Premerlani and F. Eddy - *Object-Oriented Modeling and Design* – (Prentice Hall, USA, 1991).
16. M. Fowler, K. Scott - *Uml Distilled : Applying the Standard Object Modeling Language* - (Addison-Wesley Object Technology Series, USA, 1997).

Exercises

1. Assume we are interested in modeling an industrial oven and consider that, for our purposes, the following list of attributes fully describes an oven: Width (w), Height (h), Depth (d), Weight (p), Color (c), Temperature (t), Number of heaters (nh), Clean State (cl), Age (a) and Conservation State (cs).

Notice that some of the attributes are static, as Width, Height, Depth, Weight, Color and Number of heaters. Width, Height and Depth are given in cm, Weight in kg, Color by a triplet (R,G,B), and Number of heaters is an integer number. Temperature is a dynamic attribute, ranging from 0 to 200 °C. Age is also a dynamic attribute, ranging from 0 to 50 years. Clean State and Conservation State are normalized dynamic attributes, ranging from 0 to 1. A clean state of 0 means that the oven is totally dirty. A clean state of 1 means a totally clean oven. The same holds with the Conservation State: 0 means a spoiled (useless) oven, whereas 1 means a perfect working oven.

Now, suppose that the dynamic attributes are as given by the functions depicted in figure 13:

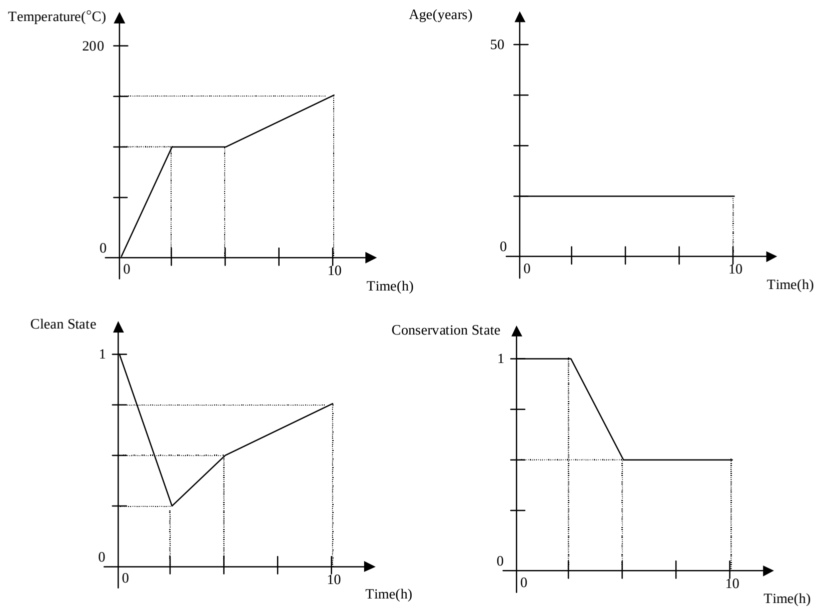

Figure 13 – Dynamic Attributes of Industrial Oven

Based on the formal definition of object, define the classes involved and the mathematical object "oven" that models the industrial oven in the period of time indicated in the figure 13.

2. Consider the object model for an industrial oven developed in exercise 1. We want to model the verb "to warm" and to use it associated with the oven. The purpose to answer the questions: Did the oven warm ? When ?

Based on the formal definition of meta-object, build a (fuzzy/generic) meta-object to model the verb "to warm". Next, find the occurrences of such meta-object in the object "oven".

3. Assume the meta-object developed in exercise 2 and the object developed in exercise 1. Transform the meta-object "to warm" into an object, and the object "oven" into an object with memory. Next, build an object network that is able to compute if (and when) something has warmed, and feed it with the object "oven" to obtain the answer.

4. Consider the graphs of object networks illustrated in figures 9, 10 and 11, illustrating examples of a fuzzy system, a neural network and an evolutionary system. Provide a formal definition of the kernel for each object network, and develop them through a sequence of object networks, until reaching object networks with 10 time steps.

5. Recall the example of an object network depicted in figure 10, which represents a neural network with unsupervised learning. Develop the graph for an object network for a neural network with supervised learning.

6. Let N be the set of integers and R the set of reals. Let it be a class C_1 representing instances of rhematic knowledge pieces given by $C_1 = \{(v_1,v_2)\}$, where $v_1 \in N$ corresponds to a flag indicating the rhematic knowledge type, as follows: 0 - sensorial piece of knowledge, 1 – designative piece of knowledge, 2 – prescriptive piece of knowledge and $v_2 \in R_n$ corresponds to the set of attributes characterizing the rhematic knowledge pieces (do not care, by now, about it). Let it be a class C_2, representing instances of objects of type sensor, given by $C_2 = \{(v_3,v_4,f_1)\}$, where $v_3 \in N$ corresponds to a timer indicating the time instant, $v_4 \in C_1$ corresponds to the output interface and f_1 is a function $f_1{:}N \rightarrow N \times C_1$, that for each time instant updates the internal timer and generates an object of type C_1 corresponding to a sensorial piece of knowledge (flag=0). Let it be a class C_3 representing instances of objects of type actuator, given by $C_3 = \{(v_5, v_6, f_2)\}$, where $v_5 \in N$ is a timer, $v_6 \in C_1$ is the input interface and f_2 is a function $f_2{:}N \times C_1 \rightarrow N$, that for each time instant updates the internal timer and destructively assimilate an object of type C_1, corresponding to a prescriptive piece of knowledge (flag=2). Let it be a class C_4, representing instances of objects of type reasoners, given by $C_4 = \{(v_7, v_8, f_3, f_4)\}$, where $v_7 \in C_1$ is the input interface, $v_8 \in C_1$ is the output interface, f_3 is a function $f_3 : C_1 \rightarrow C_1$, that destructively assimilates sensorial pieces of knowledge (flag=0), generating designative pieces of knowledge (flag=1), and f_4 is a function $f_4 : C_1 \rightarrow C_1$, that destructively assimilates designative pieces of

knowledge (flag = 1), generating prescriptive pieces of knowledge (flag=2). Let it be the kernel of an object network given by:

$\Re_0 = (\Sigma, \Pi, \Xi, A, \eta, fpi, fpo, \mathcal{C}^0, \xi^0, \gamma)$
$\Sigma = \{ C_1, C_2, C_3, C_4 \}$,
$\Pi = \{\pi_1, \pi_2, \pi_3, \pi_4\}$,
$\Xi = ((\pi_1, C_2), (\pi_2, C_3), (\pi_3, C_1), (\pi_4, C_4) \}$,
$A = \{ a_1, a_2, a_3, a_4\}$,
$\eta = \{ (a_1, (\pi_1, \pi_3)), (a_2, (\pi_3, \pi_4)), (a_3, (\pi_4, \pi_3)), (a_4, (\pi_3, \pi_2)) \}$,
$fpo_{\pi 1}(1) = a_1$, $fpi_{\pi 2}(1) = a_4$, $fpi_{\pi 4}(1) = a_2$, $fpi_{\pi 4}(2) = a_2$, $fpo_{\pi 4}(1) = a_3$, $fpo_{\pi 4}(2) = a_{3,}$
$\mathcal{C}^0 = \{ c_1, c_2, c_3 \}$, where c_1 is of type C_2, c_2 is of type C_3 and c_3 is of type $C_{4,}$
$Nu = (o, r)$ is a default value, $Nu \in C_{1,}$
$c_1 = \{ (0, (0, Nu, f_1)) \}$,
$c_2 = \{ (0, (0, Nu, f_2)) \}$,
$c_3 = \{ (0, (Nu, Nu, f_3, f_4)) \}$,
$\xi^0 = \{ (0, c_1, \pi_1), (0, c_2, \pi_2), (0, c_3, \pi_4) \}$.

Using this data, do the following:

a) Draw the graph for the object network.
b) Evolve the network $\Re_0$ for 6 steps (until $\Re_6$), using an arbitrary selection function γ (arbitrated step by step), assuming that the knowledge type requirements (if sensorial, designative or prescriptive) are sustained.
c) Speculate on an asynchronous (distributed) algorithm that should be used for the determination of all instances of γ.

7. Think about the implications of using synchronous (centralized) and asynchronous (distributed) selection functions, assuming that an object network is allowed to have two active objects competing to destructively assimilate the same object. Speculate on how to generate an asynchronous selection function dealing with this problem.