OBJECT NETWORKS : A COMPUTATIONAL FRAMEWORK TO COMPUTE WITH WORDS

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: In this work, we introduce a framework as a modeling and implementation tool to compute with words. The basis of the framework is a computational model called object networks. We present the mathematical concept of objects, generic objects and fuzzy objects. The interaction of these elements composes object systems. In particular, emphasis is given on a special class of object systems, the object networks. They are proposed here as a dynamic software architecture suitable for handling the issues involved when computing with words, but independent of the specific numerical algorithms used. The object networks architecture is easily translated into computer programs, and support explanatory databases build upon atomic pieces of information and from the query entered, among other important features. It is also shown how object networks manage semantics of linguistic, semiotic terms, to provide higher level linguistic computations. Illustrative examples are also included.

1 Introduction

Words are the elementary granules of information used by humans to communicate with each other, using sentences from a natural or artificial language. The information conveyed by those sentences can be of two types. The first type is related to the appearance and position where given words take place in sentences. This type of information considers grammatical aspects only, disregarding the concepts embedded in each word by itself. The second type of information is just the information that a word can carry by itself, i.e., its meaning as isolated granules of information. This type of information is addressed as words semantics. The grammatical aspects of words were studied extensively during the earlier periods of artificial intelligence. The semantic aspects were enhanced only recently, within the study of fuzzy sets, linguistic variables and fuzzy logic[1].

But how can humans use words to communicate each other? How do they represent and perform computations with them ? This is a basic question addressed by semiotics[2]. Let us look at figure 1.

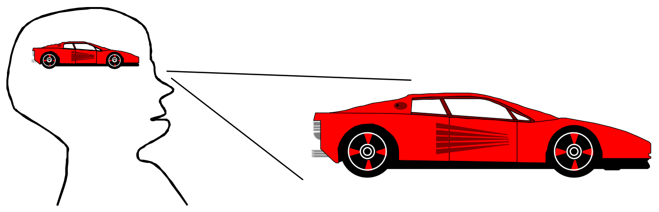

Figure 1 – Concept Formation

After being in contact with a phenomenon of the real world, humans create an internal representation for the observed phenomenon. Further, humans associate this representation to a word that summarizes and labels the concept. When presented to the word, they first recognize the word by itself, as illustrated in figure 2.

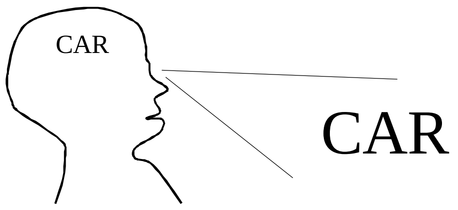

Figure 2 – Word Recognition

In a second step, the recognition of the word leads to the understanding of the concept (figure 3).

When communicating, the use of words is an optimization of information transfer, because it does not need to detail the concept being conveyed, but uses the previous contact of each subject with the phenomenon, sharing experiences to transfer the information. Some problems could arise if the semantic of words is not exactly the same for two different humans, but to be effective they must share some similarity (figure 4).

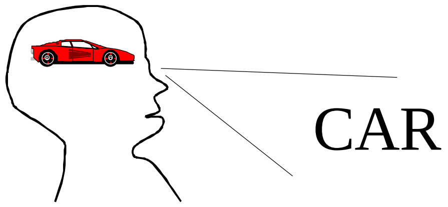

Figure 3 – Symbol Interpretation

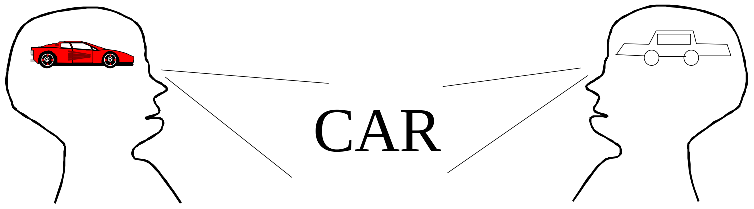

Figure 4 – Different Symbol Interpretation

From the computational point of view, one of the objectives of computing with words is to allow computers communicate with humans, using this semiotic paradigm, i.e., grounding the concepts being held by words. This can be achieved by allowing the computer to share experiences with humans. In this sense, computers should process sensory information to develop internal representations for phenomena from environment.

The study of sign processing in general, within the scope of computers, is addressed by the field of Computational Semiotics[3,4,5,6,7] . There, the different types of phenomena existing in the world are categorized in a taxonomy of types of knowledge, and a formal representation is proposed for them. In this work, we use the Computational Semiotics paradigms to focus on the use of words instead general signs. We show how the main tool within Computational Semiotics, the object network, is used to model the different types of concepts that can be acquired from environment.

In next section, we introduce the main concepts to be used: objects, generic objects, fuzzy objects, meta-objects, object system and object network. In section 3, we address the paradigm of “computing with words”, through the view of computational semiotics and object networks and in section 4 we illustrate on how object networks can be used to implement other paradigms within the context of soft computing. Section 5 ends with the conclusions.

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 formal model that implements this specification. With this formal representation, we study the interaction among objects to compose object systems and a particular kind of object system, the object network.

2.1 The Conceptual Object

Our concept of object[8] 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[9] . 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 5.

Each active object is assumed to have two types of interfaces: input and output interfaces, as in figure 5. 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.

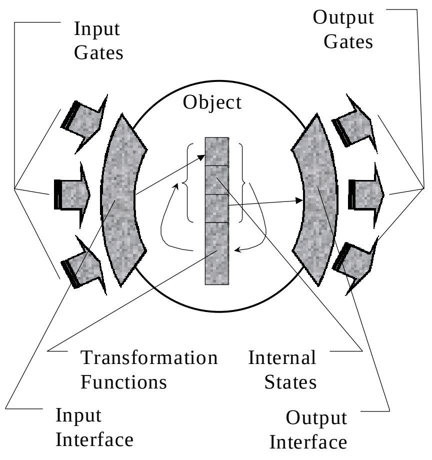

Figure 5 – The Conceptual Object

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 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.

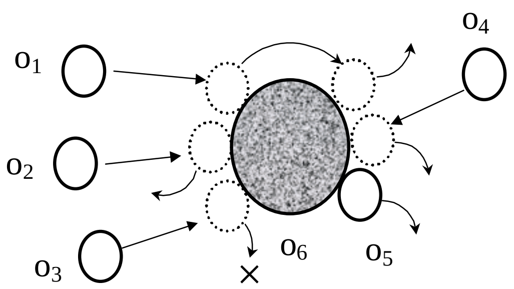

Figure 6 – Object Interactions

The triggering mechanism may allow different kinds of behavior, as illustrated in figure 6. 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.

2.2 The Formal Object

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 and detailed examples illustrating the main definitions stated below, the reader is referred to the work of Gudwin[3,4,5,6,7] . 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 , ... , q_n be generic elements of the sets Q_1 , Q_2 , ... , Q_n respectively. A tuple is a structure joining q_1 , q_2 , ... , q_n into a single element denoted by $q = (q_1 , q_2 , ... , q_n)$.

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 $q = (q_1 , q_2 , ... , q_n)$ be a tuple. The **arity** of q, Ar(q), is the number of tuple's components: Ar(q) = n.

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.

Definition 4 - **Induction Formula :** Let $q = (q_1 , q_2 , ... , q_n)$ 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.

Definition 5 - **Induction of a tuple:** Let $q = (q_1 , q_2 , ... , q_n)$ be a tuple in $Q = Q_1 \times ... \times Q_n$ 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.

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

Definition 8 – **Projection:** Let $R = \{r_i \}, r_i = (r_{i1} , ... , r_{in})$ be an n-ary relation in $R_1 \times ... \times R_n$ and k be an induction formula with unary indices $k = [k_1 , k_2 , ... , k_m]$, $k_i \in \{1, ... , n\}, k_i \neq k_j$, if $i \neq j, i = 1, ... , m , j = 1, ... , m, m \leq n$. 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, $R_{(k)} = \cup r_{i(k).}$

Definition 9 - **Free Projection:** Let $R = \{r_i \}, r_i = (r_{i1} , ... , r_{in})$ be an n-ary relation defined in $U = R_1 \times ... \times R_n$ 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 $R = \{ (r_{i1} , r_{i2} , ... , r_{in}) \}$ be an n-ary relation in $R_1 \times ... \times R_n$. The cylindrical extension P of R in $P_1 \times ... \times P_m$, denoted by $P = R \uparrow P_1 \times ... \times P_m$, where $\forall k \in \{1, ..., n\}\ \exists P_j = R_k$, $1 \leq j \leq m$, is the greatest (in the sense of the greatest number of elements) relation $P \subseteq P_1 \times ... \times P_m$ such that $P \downarrow R_1 \times ... \times R_n = R$.

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 $R_1 \times ... \times R_n$ and $S_1 \times ... \times S_m$, respectively, and $P = P_1 \times ... \times P_o$ an universe where $\forall i \in \{1, ..., n\}\ \exists P_k = R_i$, and $\forall j \in \{1, ..., m\}\ \exists P_h = S_j$, $o \leq n + m$. 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 $R*S|_P \downarrow R_1 \times ... \times R_n = R$ and $R * S|_P \downarrow S_1 \times ... \times S_m = S$.

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, $x \subset N \times X$.

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, at 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 $X = X_1 \times ... \times X_n$, 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.

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

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

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

$$f_j : \mathop{\times}_{p \in P_j} V_p \rightarrow \mathop{\times}_{q \in Q_j} V_q .$$

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

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 objects. 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 a cylindrical extension. 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 7).

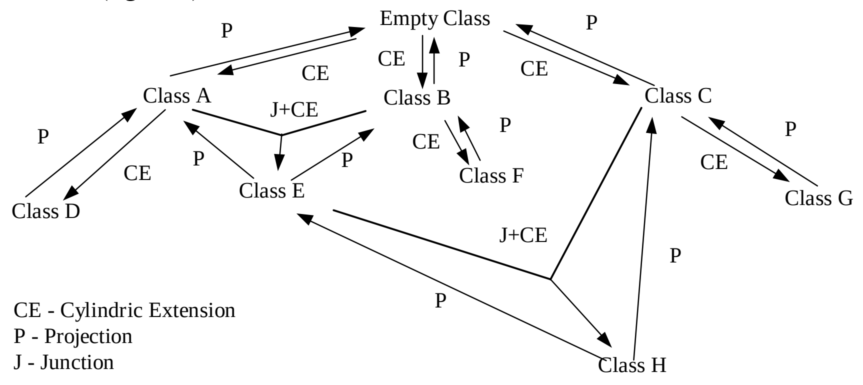

Figure 7 - Example of Class Hierarchy

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:

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

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:

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

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 $n \in N$.

Definition 25 - **Temporal Restriction for Objects:** Let N be a set of time instants, S a class and o:N→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→S such that if $(n,s) \in o$ and $n \in N'$, $(n,s) \in o'$. Otherwise, $(n,s) \notin o'$.

Definition 26 -**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 27 - **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 28 - **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 29 - **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 30 - **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) 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 31 - **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) 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 32 - **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, $v : T \to N$; 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 $x : V \to X$.

Definition 33 - **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.

Definition 34 - **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.

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.

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 35 - **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 36 - **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.

Definition 37 - **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 38 - **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 39 - **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 40 - **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 41 - **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'.

Definition 42 - **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 $x : V \rightarrow 2^{\tilde{X}}$.

Definition 43 - **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 44 - **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 45 - **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'.

2.3 Interaction Among Objects

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

Definition 46 - **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 47 - **Enabling Scope of Functions:** Consider an active object c from a class $C = \{ (v_1, v_2, \ldots, v_n, f_1, f_2, \ldots, f_m) \}$, 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, $gi : \{1, \ldots, \beta\} \to \{1, \ldots, n\}$ 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 $H = \{(h_t, b_t)\}$, $t = 1, \ldots, \beta$, 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 48 - **Generative Scope of Functions:** Assume an active object c of class $C = \{ (v_1, v_2, \ldots, v_n, f_1, f_2, \ldots, f_m) \}$, 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, $go : \{1, \ldots, \alpha\} \to \{1, \ldots, n\}$ 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 $S = \{s_u\}$, $u = 1, \ldots, \alpha$, where s_u is an object of class $V_{go(u)}$.

Definition 49 - **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 50 - **Triggering of an Active Object:** Consider the following:

a)-an object c from a class C,

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

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 $k \in P_j$, $k = 1, \ldots, n$, (P_j from def. 14)

f)-a domain index function $gd : (1, \ldots, p\} \to \{1, \ldots, n\}$ 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:

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

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

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:

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

2.4 Object Systems

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

Definition 51 - **Object System:** Assume the following:

a)-c_i are objects of class C_i, $i = 1, \dots, \delta$,

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

c)-$\Theta_i = \{0, \dots, m_i\}$ 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 $\gamma_i : N \to 2^{\mathcal{C} \times B} \times 2^{\mathcal{C}} \times \Theta_i$ 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 to build an object system. In the second view, when 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 51. 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 $i \to \infty$. Hence, the object system is computable, although infinite.

2.5 *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 8.

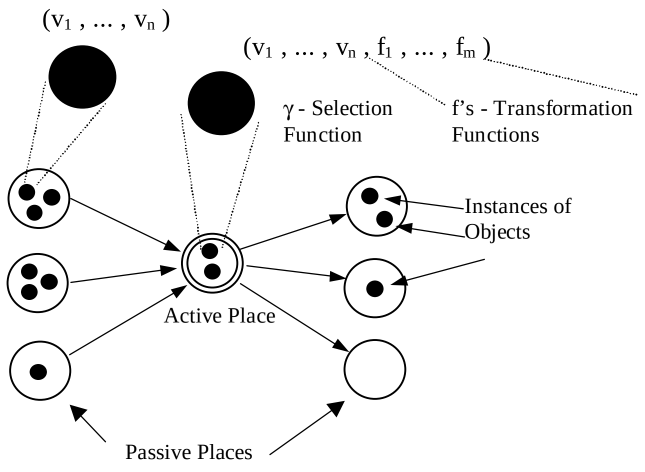

Figure 8 : Example of an Object Network

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 52 - **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 $\eta : A \to \Pi \times \Pi$

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

g)-$F(\pi)$ a mapping $\Pi \to 2^{\Pi}$, 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 $\Pi \to 2^{\Pi}$, 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 $\Pi \to 2^{\Pi}$, such that $X(\pi) = F(\pi) \cup V(\pi)$.

j)-$\Xi = \Xi(\pi)$ a mapping of classes $\Pi \to \Sigma$, 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 $fpi_i : \{1, ... , \partial_i\} \rightarrow A$ and $fpi = \{fpi_i\}$.

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

p)-$\Theta_i = \{ 0, ... , m_i \}$, 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.

Similarly to 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, $\mathcal{C}^0$, ξ^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 9. Note that each new network in the sequence includes the previous one increased by the definition of a new time instant.

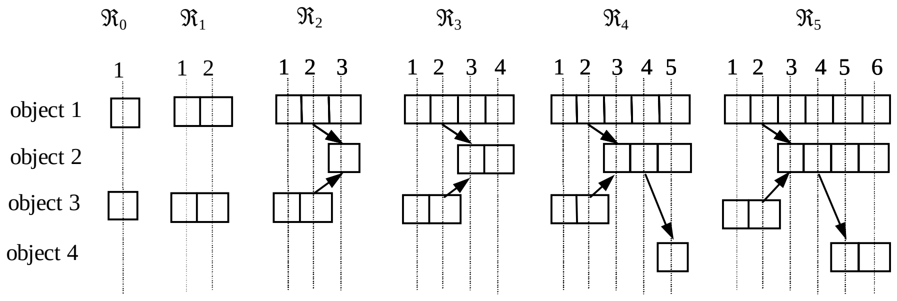

Figure 9 - Example of an Evolution Sequence

For a detailed example of the main procedures to assemble an object network, see the work of Gudwin[3,4,7].

2.6 *Using Meta-Objects within Object Networks*

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 10.

Original object

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8

object with memory

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
	c_1	c_2	c_3	c_4	c_5	c_6	c_7
		c_1	c_2	c_3	c_4	c_5	c_6
			c_1	c_2	c_3	c_4	c_5
				c_1	c_2	c_3	c_4

Figure 10 - Assembling Memory in an Object

object with memory

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
a	b	c	b	c	b	c	c
	a	b	c	b	c	b	c
		a	b	c	b	c	b
			a	b	c	b	c
				a	b	c	b

meta-object

b	c

Transformed meta-object

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
b	b	b	b	b	b	b	b
c	c	c	c	c	c	c	c

Figure 11 - Transformation of a meta-object into 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 11.

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.

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. a 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.

3 Computing with Words

Words may represent a huge variety of concepts from real world. These concepts are of different natures and originate different types of knowledge. In a natural language, words are classified by the type of meaning they can be assigned to. We may have words representing adjectives, substantives, verbs, prepositions, adverbs, etc. In computational semiotics, these meanings are referred to as rhematic iconic pieces of knowledge. Among them, the most important ones, concerning computability, are adjectives, substantives and verbs. Adjectives are related to some quality associated to an entity, being described within computational semiotics as sensorial rhematic knowledge. Substantives are related to individuals or entities of the real world, being described within computational semiotics by the object type of knowledge. And finally, verbs are related to a dynamics involving the entities of real world, being modeled within computational semiotics by a type of knowledge called occurrence knowledge.

Current approaches of computing with words only deal with the semantics of adjectives or, in other words, qualities. Qualities are modeled by values assigned to linguistic variables, represented by fuzzy sets. For example, consider the following fuzzy proposition:

P1 = Carol lives near Mary

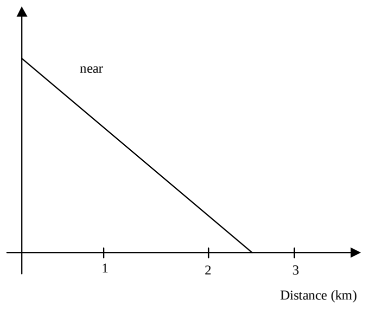

Figure 12 – Model for Fuzzy Proposition

If we use the fuzzy set of figure 12 to model such a proposition, we are in fact not modeling the whole proposition, but only the quality “near”. All other words are only grammatically modeled, in the sense that they have to appear in the proposition to let it make sense. But we are not modeling neither “Carol”, nor “live”, or “Mary”. The same is true with a rule like:

P2: If oven temperature is high then cooking time should be small

In this case, we are using a fuzzy relation R to represent the proposition, e.g., R= high $\times$ small (figure 13).

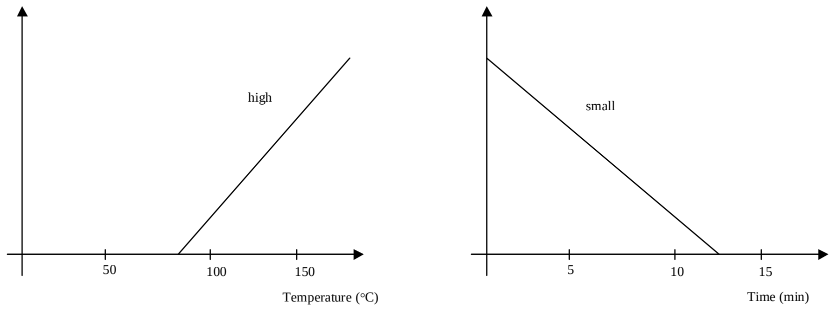

Figure 13 – Model for Fuzzy Proposition

Note that the semantics of “oven”, “temperature”, “time” and “cooking” are not modeled.

Object Networks, differently from the current approaches, provide a framework where not only quality is processed, but also entity and entity dynamics. In the next subsection, we show an example of an object network to compute with words according to the currently known approach. Subsequently we suggest on how other types of meaning may be addressed under the scope of computational semiotics.

3.1 Computing Qualities

In this example, we have two linguistic variables, oven_temperature and cooking_time. The oven temperature has its universe within 0 and 200°C range, and the cooking time within 0 and 20 minutes. These linguistic variables are used to build a very simple fuzzy rule base, as follows:

RB: If oven_temperature is low then cooking_time is large
If oven_temperature is moderate then cooking_time is medium
If oven_temperature is high then cooking_time is small

This rule base expresses (linguistically) the relation about the temperature of the oven and the time that should be used in cooking some item. Observe that both are qualities associated with other linguistic concepts. The oven_temperature is a quality (an attribute) associated with the oven, and the cooking_time is a quality associated to the dynamic process of cooking some food. Note that oven_temperature by itself doesn't give too much information of what is an oven at all. Similarly, cooking_time is not too informative about the process of cooking.

The problem here is how to compute the cooking time, using as input information about the oven temperature and the given linguistic rule base. We may consider two different outputs for this problem. The first is a specific time, e.g. to program the timer of an automatic oven. The other one is a linguistic output, to be used e.g., as an advice to the user.

The first step in solving this problem is to frame it into the object network formalism. For this purpose, let it be the following classes:

$C_1 = \{(s_1)\}$ – Class of Symbols, where $s_1 \in$ S and S is the set of words.

$C_2 = \{(s_2, i_1)\}$ – Class of Labeled Icons, where $s_2 \in$ S and $i_1 \in$ I, where I is the set of all fuzzy sets of a universe.

$C_3 = \{(s_3, (s_4, i_2), i_3, f_1)\}$ – Class of Symbol-Icon Converters, where $s_k \in S$, $i_k \in I$ and $f_1 : S \times (S \times I) \rightarrow I$ is a partial function $f_1(s_3, (s_4, i_2)) = \begin{cases} i_2, \text{if } s_3 = s_4 \\ \text{not defined, otherwise} \end{cases}$, which puts its result in i_3.

$C_4 = \{(i_4)\}$ – Class of Fuzzy Sets (Unlabeled Icons), where $i_4 \in$ I

$C_5 = \{(r_1)\}$ – Class of Fuzzy Rule Bases, where $r_1 \in R$ is a fuzzy rule base, and R is the set of all fuzzy rule bases.

$C_6 = \{(i_5, r_2, i_6, f_2)\}$ – Class of Inference Engines, where $i_k \in I$, $r_2 \in R$, and f_2 is a function that performs the fuzzy composition of i_5 and r_2, putting the result in i_6.

$C_7 = \{(i_7, v_1, f_3)\}$ – Class of Defuzzifiers, where $i_7 \in I$, $v_1 \in V$, V is a real value, and f_3 is a function $f_3 : I \to V$, that performs the defuzzification of the fuzzy set in i_7.

$C_8 = \{(v_2)\}$ – Class of Real Values, where $v_2 \in V$.

$C_9 = \{(i_8, (s_5,i_9), s_6, f_4)\}$ – Class of Icon-Symbol Converters, where $i_k \in I$, $s_k \in S$ and $f_4 : I \times (S \times I) \to S$ is a partial function $f_4(i_8,(s_5,i_9)) = \begin{cases} s_5, \text{if } i_8 = i_9 \\ \text{not defined, otherwise} \end{cases}$, which puts the result in s_6.

Next, consider the following object network, represented by its kernel $\mathfrak{R}_0$:

$\mathfrak{R}_0 = (\Sigma,\Pi,\Xi,A,\eta,fpi,fpo,\mathcal{C}^0,\xi^0,\gamma)$
$\Sigma = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9\}$
$\Pi = \{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6, \pi_7, \pi_8, \pi_9, \pi_{10}, \pi_{11}\}$,
$\Xi = \{ (\pi_1, C_1), (\pi_2, C_2), (\pi_3, C_3), (\pi_4, C_4), (\pi_5, C_5), (\pi_6, C_6), (\pi_7, C_4), (\pi_8, C_7), (\pi_9, C_8), (\pi_{10}, C_9), (\pi_{11}, C_1)\}$
$A = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11}\}$,

$\eta = \{ (a_1, (\pi_1, \pi_3)), (a_2, (\pi_2, \pi_3)), (a_3, (\pi_3, \pi_4)), (a_4, (\pi_4, \pi_6)),$
$(a_5, (\pi_5, \pi_6)), (a_6, (\pi_6, \pi_7)), (a_7, (\pi_7, \pi_8)), (a_8, (\pi_8, \pi_9)),$
$(a_9, (\pi_7, \pi_{10})), (a_{10}, (\pi_2, \pi_{10})), (a_{11}, (\pi_{10}, \pi_{11}))\}$
$fpi_{\pi3}=\{(1, a_1), (2, a_2)\}$, $fpo_{\pi3}=\{(1, a_3)\}$
$fpi_{\pi6} = \{(1, a_4), (2, a_5)\}$, $fpo_{\pi6} = \{(1, a_6)\}$
$fpi_{\pi8} = \{(1, a_7)\}$, $fpo_{\pi8} = \{(1, a_8)\}$
$fpi_{\pi10} = \{(1, a_9), (2, a_{10})\}$, $fpo_{\pi10} = \{(1, a_{11})\}$

In this example, we assume the following initial set of objects:

$\mathcal{C}^0 = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}\}$,

Where

$c_1 = \{(0, (\text{“high”}))\}$, is an object of type C_1, put in π_1, and corresponding to the word used as an input, in this case the word “high”.

$c_2 = \{(0, (\text{“low”}, mf_1))\}$, is an object of type C_2, put in π_2, and corresponding to one of the labeled icons present at the iconic knowledge base.

c_3 = { (0, (“moderate”,mf_2)) }, is an object of type C_2, put in π_2, and corresponding to one of the labeled icons present at the iconic knowledge base.
c_4 = { (0, (“high”,mf_3)) }, is an object of type C_2, put in π_2, and corresponding to one of the labeled icons present at the iconic knowledge base.
c_5 = { (0, (“small”,mf_4)) }, is an object of type C_2, put in π_2, and corresponding to one of the labeled icons present at the iconic knowledge base.
c_6 = { (0, (“medium”,mf_5)) }, is an object of type C_2, put in π_2, and corresponding to one of the labeled icons present at the iconic knowledge base.
c_7 = { (0, (“large”,mf_6)) }, is an object of type C_2, put in π_2, and corresponding to one of the labeled icons present at the iconic knowledge base.
$c_8 = \{ (0, (Def_s ,(Def_s,Def_i), Def_i ,f_1)) \}$, is an object of type C_3, put in π_3, and corresponding to a symbol-icon converter. The values $Def_i \in I$ are default values for icons, and $Def_s \in S$ are default values for symbols. They are needed because the tuples values are related to input and output interfaces, not already used in the previous iterations.
c_9 = { (0, (rb)) }, is an object of type C_5, put in π_5, and rb is the fuzzy rule base to be used.
$c_{10} = \{ (0, (Def_i ,Def_{RB} , Def_i , f_2)) \}$, is an object of type C_6, put in π_6, and corresponding to the inference engine.
$c_{11} = \{ (0, (Def_i ,Def_v , f_3)) \}$, is an object of type C_7, put in π_8, and corresponding to the defuzzifier. $Def_v \in V$ is a default real value.
$c_{12} = \{ (0, (Def_i ,(Def_s,Def_i), Def_s ,f_4)) \}$, is an object of type C_9, put in π_{10}, and corresponding to the icon-symbol converter.

Then, the corresponding localization function becomes:

$$\xi^0 = \{ (0, c_1 , \pi_1), (0, c_2 , \pi_2) , (0, c_3 , \pi_2) , (0, c_4 , \pi_2), (0, c_5 , \pi_2) , (0, c_6 , \pi_2) , (0, c_7 , \pi_2), (0, c_8 , \pi_3) , (0, c_9 , \pi_5) , (0, c_{10} , \pi_6), (0, c_{11} , \pi_8) , (0, c_{12} , \pi_{10}) \}$$

The selection function, allowing the transition for the first step is:

$\gamma (0) = \{\gamma_1 , \gamma_2 , \gamma_3 , \gamma_4 , \gamma_5 , \gamma_6 , \gamma_7 , \gamma_8 , \gamma_9 , \gamma_{10} , \gamma_{11} , \gamma_{12} \}$
$\gamma_8(0) = (\{(c_1,1), (c_4 , 0)\} , \{c_{13} \}, 1)$
$\gamma_1(0) = ... = \gamma_7(0) = \gamma_9(0) = ... = \gamma_{12}(0) = (\emptyset, \emptyset, 0)$

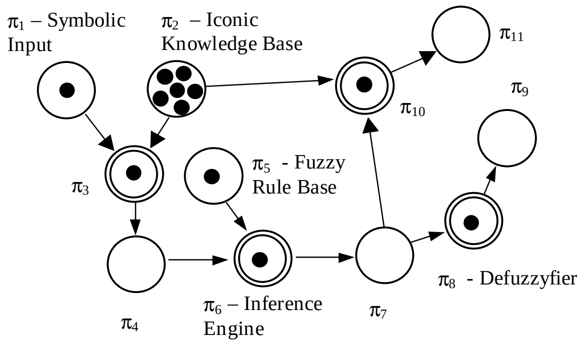

Figure 14 - Object Network at step 0

Observe that using this selection function, object c_1 is destroyed from step 0 to step 1, but object c_4 is only non-destructively assimilated. They are used by object c_8 to generate a new object c_{13} (an icon), at π_4 . With this, at step 1, the object network stays like in figure 15.

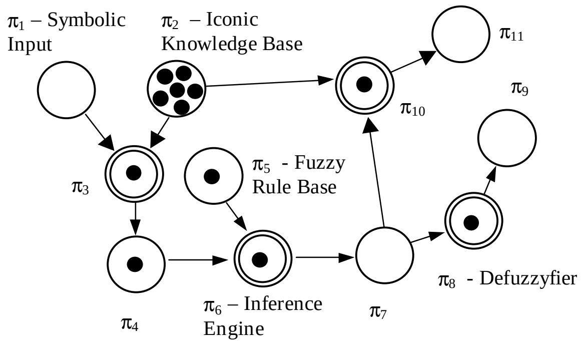

Figure 15 - Object Network at step 1

At step 1, the existing objects are:

$\mathcal{C}^1$ = { c_2 , c_3 , c_4 , c_5 , c_6 , c_7 , c_8 , c_9, c_{10} , c_{11} , c_{12} , c_{13} },

Objects c_2 , c_3 , c_4 , c_5 , c_6 , c_7 did not suffer any modification. The instance of c_8 for step 1 becomes:

c_8 (1) = (“high” ,(“high”,mf_3), mf_3 ,f_1)

Objects c_9 , c_{10} , c_{11} , c_{12} have not been modified.

Object c_{13} is a new object of type C_4 , and its instance for step 1 is:

$c_{13}(1)$ = (mf_3)

The localization function, at step 1 becomes:

$$\xi^1 = \{ (1, c_2 , \pi_2), (1, c_3 , \pi_2), (1, c_4 , \pi_2), (1, c_5 , \pi_2), (1, c_6 , \pi_2), (1, c_7 , \pi_2), (1, c_8 , \pi_3), (1, c_9 , \pi_5), (1, c_{10} , \pi_6), (1, c_{11} , \pi_8), (1, c_{12} , \pi_{10}), (1, c_{13} , \pi_4) \}$$

The selection function, enabling the transition to step 2 (figure 16) is:

$\gamma (1) = \{ \gamma_2 , \gamma_3 , \gamma_4 , \gamma_5 , \gamma_6 , \gamma_7 , \gamma_8 , \gamma_9 , \gamma_{10} , \gamma_{11} , \gamma_{12} , \gamma_{13}\}$
$\gamma_{10}(1) = (\{(c_{13},0), (c_9 , 0)\} , \{c_{13} \}, 1)$
$\gamma_2(1) = ... = \gamma_9(1) = \gamma_{11}(1) = ... = \gamma_{13}(1) = (\emptyset, \emptyset, 0)$

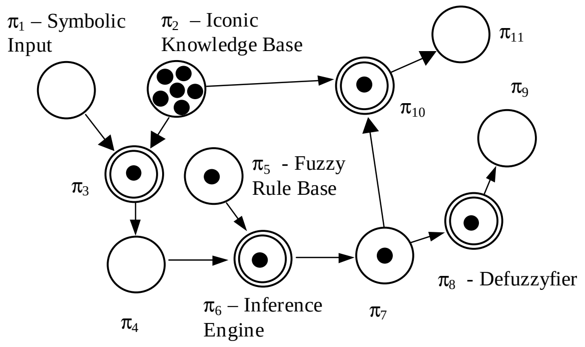

Figure 16 - Object Network at step 2

This selection function takes object c_{13}, uses its information together with the information within c_9 to compute a new fuzzy set (an icon), and store this new information again in c_{13}, that is moved to π_7 .

At step 2, the existing objects remain the same:

$\mathcal{C}^2 = \{ c_2 , c_3 , c_4 , c_5 , c_6 , c_7 , c_8 , c_9, c_{10} , c_{11} , c_{12} , c_{13} \}$,

Objects c_2 , c_3 , c_4 , c_5 , c_6 , c_7, c_8 , c_9 did not suffer any modification. The instance of c_{10} for step 2 becomes:

c_{10} (2) = (mf_3, rb, mf_x ,f_2) , where mf_x is the result of fuzzy composition involving mf_3 and rb, performed by function f_2 . Objects c_{11} and c_{12} also didn't have any modification. Object c_{13} receives a new value, after being conducted from π_4 to π_7: $c_{13}(2) = (mf_x)$

The localization function at step 2 becomes

$\xi^1 = \{ (2, c_2 , \pi_2), (2, c_3 , \pi_2), (2, c_4 , \pi_2), (2, c_5 , \pi_2), (2, c_6 , \pi_2), (2, c_7 , \pi_2),$
$(2, c_8 , \pi_3), (2, c_9 , \pi_5), (2, c_{10} , \pi_6), (2, c_{11} , \pi_8), (2, c_{12} , \pi_{10}), (2, c_{13} , \pi_7) \}$

The selection function, which allows the transition to step 3 (figure 17) is:

$\gamma (2) = \{ \gamma_2 , \gamma_3 , \gamma_4 , \gamma_5 , \gamma_6 , \gamma_7 , \gamma_8 , \gamma_9 , \gamma_{10} , \gamma_{11} , \gamma_{12} , \gamma_{13}\}$
$\gamma_{11}(2) = (\{(c_{13},1)\} , \{c_{14} \}, 1)$
$\gamma_{12}(2) = (\{(c_{13},0), (c_5 , 0)\} , \{c_{15} \}, 1)$
$\gamma_2(2) = ... = \gamma_{10}(1) = \gamma_{13}(1) = (\emptyset, \emptyset, 0)$

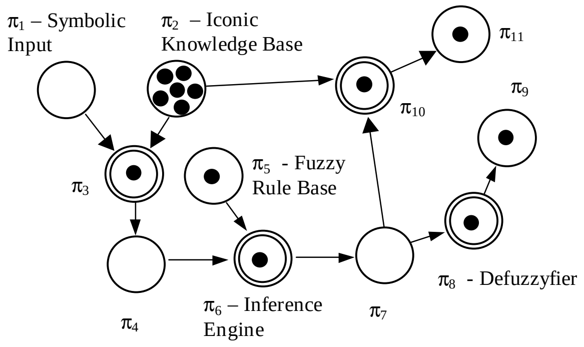

Figure 17 - Object Network at step 3

With these selection functions, object c_{13} is both used to generate a new object c_{14}, (put at π_9 - the defuzzification measure of mf_x) and a new object c_{15}, put at π_{11} (the most plausible linguistic term associated with the membership function mf_x).

With this selection function, at step 3, the existing objects become:

$\mathcal{C}^3 = \{ c_2 , c_3 , c_4 , c_5 , c_6 , c_7 , c_8 , c_9 , c_{10} , c_{11} , c_{12} , c_{14} , c_{15} \}$,

Objects c_2 , c_3 , c_4 , c_5 , c_6 , c_7 , c_8 , c_9 , c_{10} did not suffer any alteration. The instance of c_{11} and c_{12} for step 3 becomes:

$c_{11} (3) = (mf_x , x , f_3)$

$c_{12} (3) = (mf_x ,(\text{“small”},mf_4), \text{“small”} ,f_4)$

Objects c_{14} and c_{15} get the following instances:

$c_{14} (3) = (x)$

$c_{15} (3) = (\text{“small”})$

Both c_{14} and c_{15} are solutions for the example problem stated.

3.2 Other Types of Meaning

Meaning modeling for other types of words different from qualities is still an open question. Computational semiotics provides a tool for assigning such a meaning for some types of substantives (entities) and some types of verbs (entities dynamics), but mechanisms for performing computations using such a representation are under development. In this section, we illustrate on how we may use objects and meta-objects to do such a task.

Consider, e.g., that we want to assign a meaning to the word "oven". To do this, we have to find a collection of attributes that adequately represent the concept of an oven. Suppose that we acquaint that the following list is a good set of such attributes: Width (w), Height (h), Depth (d), Weight (p), Color (c), Temperature (t), number of heaters (nh), Cleaning State (cl), Age (a), Conservation State (cs). Then, to fully describe an oven in a particular instant of time, we may use a tuple like:

$$(w,h,d,p,c,t,nh,cl,a,cs)$$

But, to fully describe an oven, we have to consider not only a description for the current time instant, but also to its whole existence in the real world, during different periods of time. For different instants, some of the attributes may have a different value than the current one. So, to fully model the word "oven", we have to use a mathematical object. Let O be a class, given by O = { (w,h,d,p,c,t,nh,cl,a,cs) }. Then, an object o of type O is said to be a model for an oven from the real world. Notice that each of oven's attributes, may be modeled by a number, a set of numbers or a fuzzy set. Depending on each case, we may have a standard object, a generic object or a fuzzy object.

Consider now that we want to assign a meaning to the verb "to warm". The main meaning for such a word is that, for some entity of the world, there should be an attribute, measuring a temperature, and there must be a change for this temperature, from one value, somewhat classified as small, to other value considered large. This meaning can be represented using a meta-object.

For example, let us define a class C = {(t)}, (notice that it is a superclass of O), and define a fuzzy meta-object w of class C given by:

$$w = \{(v_1, \text{small}), (v_2, \text{large})\},$$

where small and large are two fuzzy sets defined on the universe of temperatures. This meta-object can be represented by table 1 below:

	v_1	v_2
t	small	large

Table 1 – Generic Meta-object warmed

Note that once the meta-object is defined, it can be used with an object to make a phrase make sense. For example, “to warm” may be used with any other substantive, which is modeled by an object that has “temperature” among its attributes. In this case, variables v_1 and v_2 are substituted by instants of time, in the object description, to establish a correspondence with the dynamic behavior “to warm”, assuring that it really happened in the object history through time.

It is important to stress that the examples presented in this sub-section are only a sketch on how effectively using the meaning of substantives and verbs in computing. It is also worth to mention that the computations envisioned here will certainly use the kind of techniques addressed so far, using the current computing with words paradigm. What we actually claim is that object networks are a very suitable tool to do computations concerning words. They not only are able to perform the computations which have been already proposed in the literature, but introduce a formalism that suggests extensions to be made in order to enhance the representation power of the paradigm.

4 Computational Intelligence and Object Nets

The same framework used in last section to perform computation with words, can be used to model other types of paradigms within computational intelligence[12] and soft computing[13] .

In this section, we briefly suggest how we can use object networks to model neural networks and evolutionary systems.

4.1 Neural Networks

Figure 18 shows the representation of a self-organizing neural network by an object network. Note that this neural network has two main arguments[6] . 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 (e.g., the weights and offsets of the neural net) to perform learning.

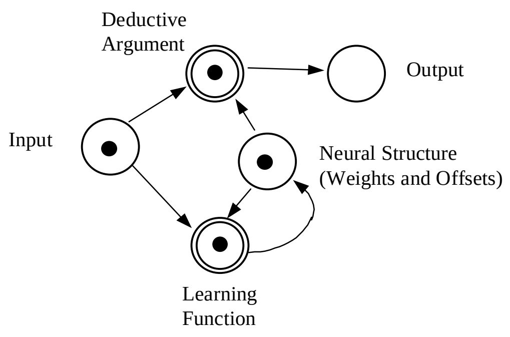

Figure 18 – Self-Organizing Neural Network

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.

4.2 Evolutionary Systems

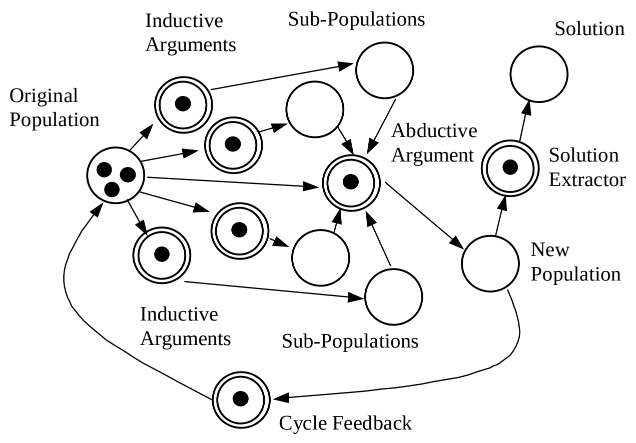

Figure 19 – Evolutionary System

An example of an evolutionary system is given in figure 19. In this example, there is an original population used as input for four inductive arguments (performing crossover, mutation, inductive mutation, etc.), generating new sub-populations. The four new sub-populations are used in conjunction with the original population to feed 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.

5 Conclusion

The notion of object network has been introduced with the purpose to provide a modeling and implementation framework for computing with words. Beginning with the mathematical concept of objects, generic objects and fuzzy objects, we addressed their interaction to each other, composing object systems. A special class of object systems, the object networks was emphasized, proposed here as a dynamic software architecture suitable for handling the issues involved when computing with words, independently of the specific numerical algorithms used. An example of an object network performing computations with words was detailed. We also suggested how object networks may manage semantics of linguistic, semiotic terms, to provide higher level linguistic computations.

The main conclusions are that object networks, addressed within the context of computational semiotics, are a very suitable tool to perform computation with words. Object networks provide a representation for the current approaches in this field, and suggest extensions to increase the representation power of such a paradigm.

Acknowledgements

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

References

1. L.Zadeh, “Fuzzy Logic = Computing with Words” – *IEEE Transactions on Fuzzy Systems*, pp. 103-111, vol. 4, no. 2, May (1996).
2. C.S. Peirce, *Collected Papers of Charles Sanders Peirce* - vol I - Principles of Philosophy; vol II - Elements of Logic; vol III - Exact Logic; vol IV - The Simplest Mathematics; vol V - Pragmatism and Pragmaticism; vol. VI - Scientific Metaphysics - edited by Charles Hartshorne and Paul Weiss - Belknap Press of Harvard University Press – (Cambridge, Massachusetts, 2nd printing, 1960).

3. 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).
4. 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).
5. 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).
6. R.R.Gudwin and F.A.C.Gomide, *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).
7. R.R.Gudwin and F.A.C. Gomide, "Object Network : A Formal Model to Develop Intelligent Systems" in *Computational Intelligence and Software Engineering*, J.Peters and W. Pedrycz (Eds.) – World Scientific, 1998.
8. A.Snyder, "The Essence of Objects:Concepts and Terms" - *IEEE Software*, pp. 31-42, Jan. (1993).
9. 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).
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. J.Zurada, R.J.Marks II, and C.J.Robinson, - *Computational Intelligence - Imitating Life* – (IEEE Press, USA, 1994).
13. L.Zadeh, – "Soft Computing and Fuzzy Logic", *IEEE Software*, vol. 11, n. 6, pp. 48-56, (1994).