Computational Semiotics : A New Aproach for the Study of Intelligent System
Part II : Theory and Application

Ricardo Gudwin
DCA-FEEC-UNICAMP
gudwin@dca.fee.unicamp.br

Fernando Gomide
DCA-FEEC-UNICAMP
gomide@dca.fee.unicamp.br

ABSTRACT: The aim of this paper is to introduce the theoretical foundations of an approach for intelligent systems development. Derived from semiotics, a classic discipline in human sciences, the theory developed provides a mathematical framework for the concept of knowledge and for knowledge processing. As a result, a new perspective to study and to develop intelligent systems emerges. In part I, we focused on the essentials of such a theory. It included the background needed from semiotics, describing the relationship between semiotics, intelligent systems and cognition. A taxonomy of the elementary types of knowledge was proposed, followed by another classification considered by the point of view of its application in a cognitive system. In addition, a mathematical definition of objects and object networks, were proposed. Here, in part II, we proceed introducing the issues concerning knowledge modeling and system development. The elementary types of knowledge are modelled using the concepts of objects and object networks. Based on the idea of applied knowledge, a general architecture for an intelligent system is suggested. An example concerning control of an autonomous vehicle is addressed to illustrate the use of computational semiotics in developing intelligent systems, and to show how intelligent systems can be implemented. Finally, the conclusions and further work are addressed.

1. Introduction

The formal body of theory introduced in Part I is considered with the purpose of consolidating the definition of computational semiotics. We use the concepts introduced informally in Part I, and the formal model of object and object network to model each type of knowledge described earlier.

In Part I, we repeatedly used the term “structure”, or “mathematical structure”, without a further explanation of what would be the meaning of these words. The basic mathematical structure used here to represent the diverse types of knowledge is the mathematical object and object networks. Rhematic and dicent knowledge are modeled as passive objects whereas argumentative knowledge is modeled by active objects. To restrict the activity of active objects, we use object networks to assure that argumentative knowledge is applied only when necessary.

2. Representation of Rhematic Knowledge

2.1 Specific Sensorial Rhematic Knowledge

This is the most elementary knowledge type. It comprises a model of input interface, restrictions of input interface to specific input times, free projections of input interface in its sub-dimensions, or projections followed by restrictions. Its model is given by the mathematical object. For a set N of discrete instant times, and a sensorial space $S = S_1 \times S_2 \times ... \times S_k$, the model for a specific sensorial rhematic knowledge is given as a passive object o, $o : N \rightarrow S$.

DEFINITION 1 - Temporal Restriction for Objects

Let N be a set of time instants, S a class and $o:N\rightarrow S$ 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 $o':N\rightarrow S$ such that if $(n,s) \in o$ and $n \in N'$, $(n,s) \in o'$. Otherwise, $(n,s) \notin o'$.

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

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

Examples of the many types of specific sensorial rhematic knowledge are as follows:

Consider a system with inputs x_1, x_2 and $x_3 \in X = \{0,1,2\}$, and the set of time instants $T = \{ t_1 , t_2 , t_3 \}$. Suppose that at t_1 , $x_1 = 0$, $x_2 = 0$ and $x_3 = 0$. At t_2 , $x_1 = 0$, $x_2 = 1$ and $x_3 = 0$. And, at t_3 , $x_1 = 1$, $x_2 = 2$ and $x_3 = 2$. The sensorial rhematic knowledge corresponding to input interface in this case is represented by the object $o = \{ (t_1 , (0,0,0)) , (t_2 , (0,1,0)) , (t_3 , (1,2,1)) \}$. Assume the induction formulae $\alpha = [1, (2,3), (2,2)]$. Object $o_{(\alpha)} = \{ (t_1 , (0,0)) , (t_2 , (0,1)) , (t_3 , (1,2)) \}$ is also an example of specific sensorial rhematic knowledge. Object o', defined by o restricted to $T' = \{ t_1 , t_3 \}$, $o' = \{ (t_1 , (0,0,0)) , (t_3 , (1,2,1)) \}$ is also an example of a specific sensorial rhematic knowledge, as well as object o'', given by $o_{(\alpha)}$ restricted to T', $o'' = \{ (t_1 , (0,0)) , (t_3 , (1,2)) \}$. The object $o''' = o \Downarrow T''$, $T'' = \{t_2 \}$, $o''' = \{ (t_2 , (0,1,0)) \}$ is also an example.

In many situations, objects similar to o''' are used to represent knowledge coming from sensors. In fact, this is very common. Note that, in this case, they do not carry its temporal history and can be easily stored.

2.2 Generic Sensorial Rhematic Knowledge

A generic sensorial rhematic knowledge is an abstraction of a specific sensorial rhematic knowledge. It is capable of representing, at the same time, many specific sensorial rhematic knowledge. To convey this characteristic, the mathematical object is not sufficient by itself. To handle it, we need to introduce first the following definitions:

DEFINITION 2 -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 3 - 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 4 - 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 5 - 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, a m-ary fuzzy relation. In some cases, it will be 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 occur with active objects, because the operations are related with non-function fields only.

DEFINITION 6 - 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 7 - 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.

The basic structure of generic sensorial rhematic knowledge is the passive generic object or, more generally, a passive fuzzy object. Note that if we define a set of specific sensorial rhematic knowledge, it is possible to build a generic object such that these specific knowledges are cases of such generic object. If these cases are weighted by values between 0 and 1, we have a fuzzy object.

There are different ways and methods to specify a relation, i.e., there are different ways and methods to specify a generic object. For example, we can define a distance measure in a metric space and, given a point in this space, build a relation composed by all the points within a certain distance from that point. This criteria is known, in cognition sciences, as the prototype method (Cohen & Murphy, 1984). In this case, a specific sensorial rhematic knowledge is used as a prototype, and given a distance measure, one can define a generic sensorial rhematic knowledge. Other example uses discriminating function which, given a point, determines if such point belongs or not to the relation. This is the method used, e.g. in neural networks such as the multilayer Perceptrons (Lippman, 1987), using a hard-limiter activation function. A Perceptron emulates a discriminant function, which can be as precise as determined by the number of neurons and layers in the network. The advantage of discriminating functions over the prototype method is that in the second, only hyperspheric regions can be defined whereas using discriminating functions one can define regions of generic topology. More specifically, in the case of Perceptrons with 3 layers, it was proved that they can discriminate generic spaces, with precision depending on the number of neurons of the intermediate layer. In general, a Perceptron with 3 layers can be viewed as a generic approximator for continuous discriminant functions in compact domains. If the Perceptron's non-linear activation function is continuous (e.g. a sigmoidal function) the discriminant function obtained by the Perceptron characterizes a fuzzy relation. Thus, instead of a generic object, we obtain the specification of a fuzzy object.

It is interesting to note, in this case, a parallelism between these observations and the relatively recent discoveries on the neural morphology of human brain (Verschure, 1993; Edelman & Mountcastle, 1978; Edelman, 1987). According to these discoveries, the basic structures found in human cortex are the neuron columns, also known as neural groups. A neural group corresponds to a group of strongly interconnected neurons that are activated at the same time in the presence of a given stimuli. The neurons within the group, which may have multiple layers, react at the same time to a determinate stimuli, contrary to those neighbor neurons that do not belong to the group. A group may be represented, in a structural way, by a multi-layer Perceptron. If we associate this structure to the definition of a generic sensorial rhematic knowledge, one can suggest that those structures may be working, in human brain, as a way of storing generic sensorial rhematic knowledge.

For the sake of computational implementation, however, 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 which 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.

2.3 Specific Object Rhematic Knowledge

Differently from sensorial rhematic knowledge, that can be obtained directly from the system's sensors, the object rhematic knowledge is an abstraction over sensorial information. An abstraction that is capable of concentrating a greater amount of information when compared to sensorial rhematic knowledge. Basically, the object rhematic knowledge is a model for a bundle of sensorial rhematic knowledge, that repeats over and over at the sensors, due to some environment (unknown) property. This model may be obtained, e.g., as an abduction over sensorial knowledge. In this case, the existence of an object is proposed to explain the sensorial stimulations that the system receives. It is obvious that such abduction have to be consistent, i.e. the object rhematic must be akin to the sensorial knowledges collected by the system. How do we implement such abduction ? Basically, sensor space should be recoded by means of transformations that are invariant in the abducted objects. With that, given a sensorial rhematic knowledge, we can determinate to which object rhematic knowledge it is related to. This transformation may map points in sensor space to a new space, called object space. This space must have a metric such that different stimulations in sensor space , related to a same object, are near in the object space.

Despite its different origin, specific object rhematic knowledges are structurally similar to specific sensorial rhematic knowledges. So, we represent them in the same way: by passive mathematical objects. The only difference is in the object classes. Instead of being defined according to the sensor space, the object classes can, in principle, be anyone (passive). This happens because the structure of such classes depends on the transformation which maps sensor space onto object space.

2.4 Generic Object Rhematic Knowledge

Similarly, the way a specific object rhematic knowledge is related with a specific sensorial rhematic knowledge, generic object rhematic knowledge is akin to generic sensorial rhematic knowledge. It can be represented by a passive generic object or by a passive fuzzy object. The class of generic object (or fuzzy object) does not need to have, in principle, any relation with the sensor space. Again, this happens because classes are generated by transformation of sensorial knowledge, and there are no restrictions on the image of such transformation. Also, generic (fuzzy) objects can be converted to standard mathematical objects for computational purposes.

2.5 Occurrence Rhematic Knowledge

Occurrence rhematic knowledge is used to formally describe subsets of object history. An object temporal trajectory can be seen as a sequence of instances of a class. This sequence can contain sub-sequences that appear once or repeat itself in the sequence. Occurrence rhematic knowledge corresponds to the identification and representation of such sub-sequences (which are called occurrences). Instances may have many fields, and only part of them may be relevant in the occurrence characterization. With this, the formal model for occurrences may, in addition to pointing the sub-sequences, mask only the relevant fields to be characterized. We also consider that sub-sequences are not necessarily built up of contiguous instances. Any sub-sequence of the original sequence is allowed. In some cases, we may also want to correlate occurrences that happen in two or more different objects. The formal model should allow such a correlation. Therefore, one can represent occurrences that affect more than one object. To derive a mathematical model comprising different occurrences, an extension of the object formal model, the meta-object is introduced next.

DEFINITION 8 - 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 9 - 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 10 - 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 do 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, whith each object from a different class, but sharing a field of the instance of the meta-object.

DEFINITION 11 - 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 12 - 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}) $\top$ 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: $a''_1 = a_1 \top a'_1$ = {1/0, 2/0.8, 3/0 }, $b''_1 = b_1 \top b'_1$ ={5/0.3, 6/0, 7/0 }, $c''_1 = c_1 \top c'_1$ = {15/0.2, 18/0 }, $a''_2 = a_3 \top a'_2$ = {1/0, 2/0, 3/0.3 }, $b''_2 = b_3 \top b'_2$ = {5/0, 6/0, 7/0.8 }, $c''_2 = c_3 \top c'_2$ = {15/0, 18/0.1 }.

DEFINITION 13 - 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 $\to 2^X$.

DEFINITION 14 - 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 15 - 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 16 - 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 17 - 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 18 - 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 19 - 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 20 - 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 21 - 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 ocurrence 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 the definition for an occurrence of a fuzzy meta-object in a fuzzy object is used.

The occurrence rhematic knowledge is mathematically represented by meta-objects. More specifically, this representation uses meta-objects, generic meta-objects or fuzzy meta-objects. A specific occurrence rhematic knowledge is represented by meta-objects, generic meta-objects or fuzzy meta-object, where we explicitly set for each v_i a specific value in N. This is equivalent to say that the restrictions in R for those meta-objects should be algebraic equations of attribution. Thus, an specific occurrence rhematic knowledge is represented by an instance of a meta-object, generic meta-object or fuzzy meta-object. For generic occurrence rhematic knowledges, this condition is not necessary. The choice among a meta-objects, a generic meta-objects or a fuzzy meta-objects is according to the knowledge we want to represent.

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. the unitary occurrence, that basically 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 1.

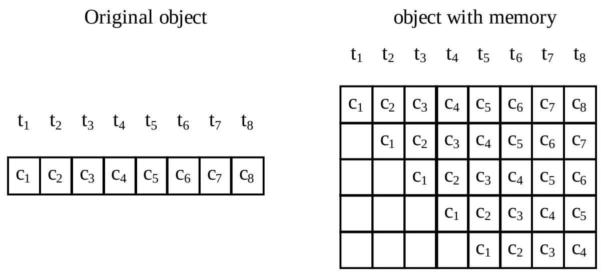

Original object
object with memory
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8
c_1 c_2 c_3 c_4 c_5 c_6 c_7 c_8
c_1 c_2 c_3 c_4 c_5 c_6 c_7
c_1 c_2 c_3 c_4 c_5 c_6
c_1 c_2 c_3 c_4 c_5
c_1 c_2 c_3 c_4

Figure 1 - Assembling Memory in an Object

Once the related objects are equiped with memory, we can modify the meta-object to become an object. This transformation is shown in figure 2.

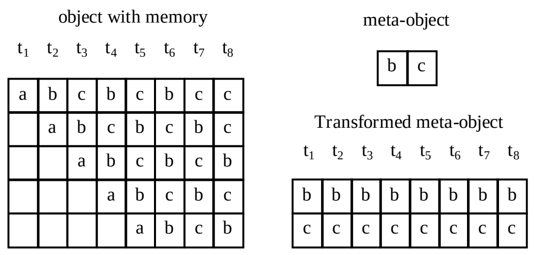


Figure 2 - Transformation of a meta-object into an object

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. Therefore, there is a limitation in the use of occurrence rhematic knowledge in such cases. 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 3.

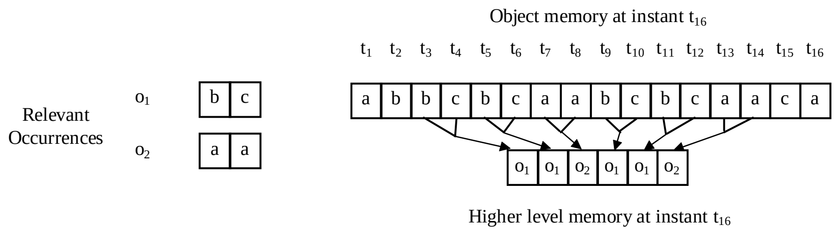

Object memory at instant t_{16}
Relevant Occurrences
Higher level memory at instant t_{16}

Figure 3 - Composition of a Higher Level Memory

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

3. Dicent Knowledge Representation

Essentially, dicent knowledge, is logical knowledge, and as such, it is represented by expressions involving propositions and logic connectives. Its main characteristic is the association of a truth value with a given expression. The truth value of a proposition may be found from either the truth values of other propositions, or by means of rhematic knowledges grounding the proposition. Propositions can be classified in two main classes: primitive or composed. Primitive propositions can be further classified as iconic or symbolic. Iconic propositions are expressions composed by rhematic knowledge grounding the proposition. In this sense, rhematic knowledge is related with dicent knowledge, because it provides a way for truth value determination. The relationship between rhematic knowledge and truth values of dicent knowledge can be established in two ways. In the first, rhematic knowledge is used to determine the truth value of an iconic proposition. In the second, the truth value of an iconic proposition is used to determine the rhematic knowledge associated, or to complement pieces of rhematic knowledge currently unknown. In iconic propositions, dicent knowledge and rhematic knowledge are coupled, in the sense that one may justify the other. Symbolic propositions are somewhat different from iconic proposition. In symbolic propositions, there is no link with rhematic knowledge to ground the proposition’s truth value. Thus, truth values of symbolic propositions can only be determined from truth values of other propositions. In a closed system (i.e. a system interacting with an environment), it is necessary to have conditional propositions coupling iconic and symbolic propositions to allow their truth values determination.

In classic artificial intelligence, iconic propositions are not considered. Therefore, dicent knowledge and rhematic knowledge are not related and the problem known as the "symbol grounding problem" arises. This occurs because the grounding given by rhematic knowledge is missing in such systems. The way this issue is usually handled is to assume the truth values of a particular subset of propositions (called facts) known, deriving the remaining truth values through conditional propositions (called rules) and deductive inference. Note that the coupling between rhematic and dicent knowledge does exist, anyway. What happens in classic AI is that there is no methodology for the treatment of this coupling. It occurs implicitly, with the statement that the truth values of some propositions are known. This is equivalent to say that truth values are given by external agents e.g., a human being or another system able to handle the coupling between rhematic and dicent knowledge. In the model herein proposed, a methodology to treat the coupling between dicent and rhematic knowledge emerges, as stated by iconic propositions. Therefore, there is no need, in principle, to assume (as in classic AI) that the truth values of some propositions are known a priori.

DEFINITION 22 Expression

An expression E is a sequence of symbols $e_1 , e_2 , ... , e_n$.

DEFINITION 23 Iconic Proposition

Let S be a set of structures corresponding to sensorial rhematic knowledges, $S = \{ s_1 , ... , s_k \}$, B, a set of structures corresponding to object rhematic knowledges, $B = \{ b_1 , ... , b_l \}$, O, a set of structures corresponding to occurrence rhematic knowledges, $O = \{ o_1 , ... , o_m \}$ and N, the set of natural numbers. An iconic proposition p is an expression formed in the following manner:

$$p = a (c_1 , ... , c_n) / f , r$$

where $n \in \{1, 2, 3, ... \}$, $a \in O$ is called the **verb**, $c_i \in S$ or $c_i \in B$, $i = 1, ... , n$ are called **relata,** f is a function $f : N \to N \times N$ which maps each field of the verb's class to a pair (x_1, x_2), where $x_1 \in \{1, ... , n\}$ is the index corresponding to some relatum, and x_2 is an index corresponding to a field of the verb's class given by x_1 , and r is the set of restrictions (possibly empty), that impose possible restrictions on temporal values to be used in substitutions of the verb's variables.

An iconic proposition can be read as "a occurs in $c_1 , ... , c_n$ according to f and r". This is equivalent to say that a may occur in the objects $c_1 , ... , c_n$, if we map a's fields in c_i's fields according to f, provided that the restrictions in r are fulfilled.

Example: Assume objects o_1 and o_2 representing two object rhematic knowledges, the meta-object m_1 representing an occurrence rhematic knowledge, a function f given by f(1) = (1,2) and f(2) = (2,1), an empty set of restrictions $r = \emptyset$. In addition, suppose that o_1's class is $A = A_1 \times A_2 \times A_3$, o_2's class is $B = B_1 \times B_2 \times B_3$, m_1's class is $C = C_1 \times C_2$, $A_2 = C_1$ and $B_1 = C_2$, $o_1 \downarrow A_2 = o'_1$, $o_2 \downarrow B_1 = o'_2$, $m_1 (v_1) = (o'_1 (t_6) , o'_2 (t_6))$, $m_1 (v_2) = (o'_1 (t_8) , o'_2 (t_8))$, $m_1 (v_3) = (o'_1 (t_9) , o'_2 (t_9))$. Then, a diagram representing the iconic proposition $m_1 (o_1 , o_2) / f, r$ is showed in figure 4.

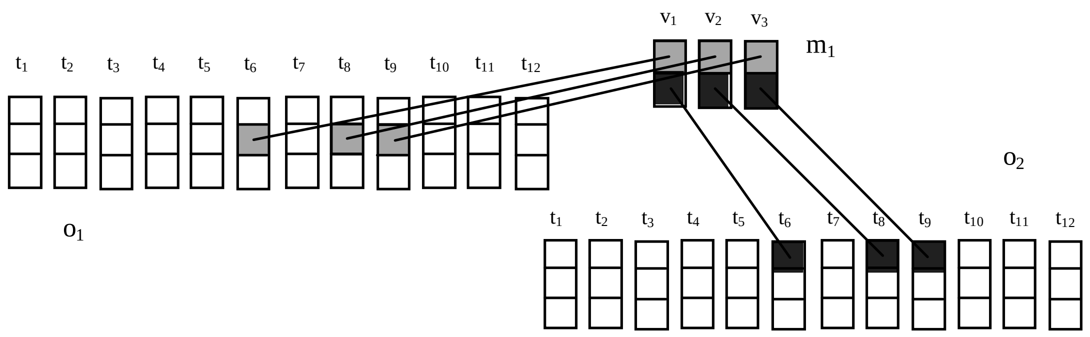
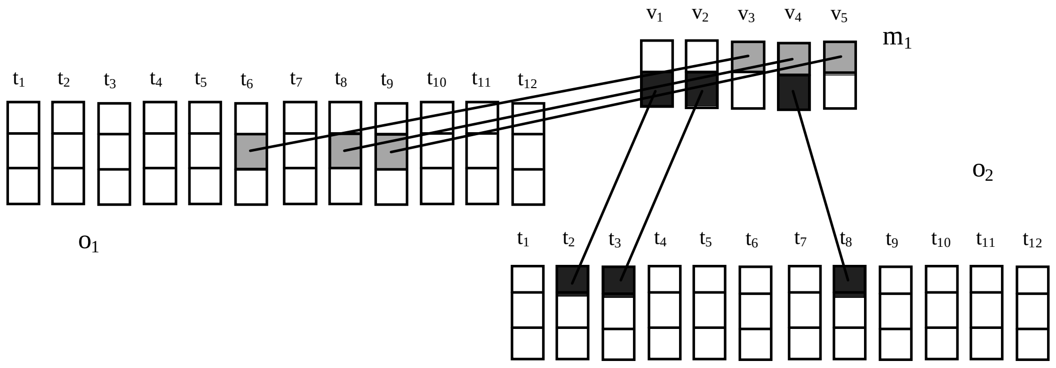


Figure 4 - Example of an Iconic Proposition

In this case, we say that the meta-object m_1 occurs in o_1 and o_2 or, in an equivalent statement, that the iconic proposition $m_1 (o_1 , o_2) / f, r$ has the truth value "true".

When the verb is represented by a generic meta-object, we have the situation showed in figure 5.



v_1 v_2 v_3 v_4 v_5
m_1
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12}
o_2
o_1
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12}

Figure 5 - Iconic Proposition using Generic Meta-Objects

In the example of figure 5, the first components of $m_1 (v_1)$ and $m_1 (v_2)$, and the second components of $m_1 (v_3)$ and $m_1 (v_5)$ are generic, that is, they may have any value (in A_2 in the first case and in B_1 in the second). Thus, only the fields marked in figure have to be equal for the truth value of such proposition to be "true".

DEFINITION 24 - Symbolic Proposition

A symbolic proposition is an expression with only one symbol, not attached to any rhematic knowledge.

DEFINITION 25 - Primitive Proposition

A primitive proposition is either an iconic or a symbolic proposition.

DEFINITION 26 - Truth Value

A truth value is a value in [0,1] which corresponds to the degree of truthness of a given proposition. We can establish the following association: truth value = 0 $\rightarrow$ "false", truth value = 1 $\rightarrow$ "true". Truth values in (0,1) are neither totally false nor totally true. The truth value 0.5 means total indetermination.

DEFINITION 27 – Procedure to determine the truth value of an iconic proposition according to its rhematic knowledge

Assume an iconic proposition $p = a (c_1 , ... , c_n) / f , r$. The truth value V(p) of p, can be found from the rhematic knowledge associated to it by:

$$V(p) = v_1 \top ... \top v_n$$

where:

1. if there exist m occurrences o_{im} of a in c_i (according to f and r) which are fuzzy objects,

$$v_i = \sup_m \left(\sup\ o_{im} \right),$$

2. if there exists at least one occurrence o_i of a in c_i (according to f and r) which is an object or a generic object, $v_i = 1$.

3. If does not exist any occurrence of a in c_i , $v_i = 0$.

DEFINITION 28- Proposition

Let P be a set of primitive propositions, L_1 a set of unary logic operators and L_2 a set of binary logic operators. An expression R is a proposition if and only if:
1. R is a primitive proposition, or
2. R can be decomposed in $R_1\ l_2\ R_2$, where $l_2 \in L_2$ and R_1 and R_2 are propositions, or
3. R can be decomposed in $l_1\ R_1$, where $l_1 \in L_1$ and R_1 is a proposition.

Normally we use $L_1 = \{\sim$ (negation) $\}$ and $L_2 = \{ \wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (implication) $\}$ The operators are used to calculate the truth value of a proposition, according to the truth values of its primitive propositions. For instance, usually we have: $V(\sim a) = 1 - V(a)$, $V(a \wedge b) = V(a)\ \mathsf{T}\ V(b)$, $V(a \vee b) = V(a)\ \mathsf{S}\ V(b)$, $V(a \rightarrow b) = V(\sim a \vee b)$

Summing up, dicent knowledge can be represented by expressions which, in turn can or cannot refer to rhematic knowledge. Thus, there is always a truth value associated with rhematic knowledge. The truth value, in iconic propositions, can be found from the associated rhematic knowledge. In general, truth values can be determined from the truth values of other propositions (this is done using argumentative knowledge, to be addressed in next section). Therefore, we can encapsulate dicent knowledge in a tuple (E,V), where E is an expression representing the dicent knowledge and V is the corresponding truth value. If necessary, temporal aspects can be included (as in temporal logic), using objects which map time in tuples of type (E,V).

Next, we show how rhematic and dicent knowledge are used to generate new knowledge, or to update current knowledge.

4. Argumentative Knowledge Representation

Argumentative knowledge is the most complexed and sophisticated among all types of knowledge discussed so far. Its complexity is due to its role in transformation, i.e. its characteristic in being itself not only knowledge, but more than that, knowledge able to perform transformations to generate new knowledge. The arguments (or argumentative knowledge) embeds what is usually known as inference, reasoning, learning, adaptation, etc.

An unified study of arguments is still lacking in AI and other areas. Deductive arguments are deeply rooted in the context of logic. Synthetic arguments were studied in the context of machine learning (Bolc, 1987; Michalski et. al. 1983; Michalski et.al. 1986). The field of machine learning emphasizes mainly inductive arguments (Michalski, 1983), and eventually combinations of inductive with deductive argument to provide learning by analogy (Carbonell, 1983). Michalski (1987) introduced the following classification of different types of learning: direct learning , learning by instruction, deductive learning, learning by analogy, learning trough examples and learning trough observation and discovery.

In the past, machine learning was heavily supported by knowledge representation based in predicates and extensions of predicate logic. Here we note a large gap, as far as rhematic knowledge is concerned. Normally, the emphasis is on dicent knowledge using symbolic propositions only. In general terms, abduction is not explored, with a few exceptions (Ram & Leake, 1991; Bylander et.al. 1991), mainly due to computational complexity problems (Bylander et.al. 1991).

Recently, contributions to the study of arguments came from the field known as "computational intelligence" (Zurada et.al, 1994). Three main areas can be distinguished: fuzzy logic and fuzzy systems, neural networks and evolutive computing. It is interesting to note the similarity among these three areas of computational intelligence and the three basic types of arguments discussed in this section. Although not totally precise, we may associate fuzzy logic and fuzzy systems with deductive arguments, neural networks with inductive arguments and evolutive computing with abductive arguments. In particular, evolutive computing brought a new impetus to the study of abductive arguments, making its implementation computationally feasible.

Depending on the knowledge available and on the knowledge types involved, different types of arguments may be identified, modifying or generating new knowledge,. A complete identification of all types of arguments is beyond the scope of the present work, however. Here, we formal definitions for the three basic types of arguments: deductive, inductive and abductive arguments, respectively.

4.1 Genericity[1]

Within the scope of rhematic and dicent knowledge, different knowledges of the same type can be classified according to its genericity, i.e. how much is one of them more generic than the other. For rhematic knowledges, more generic knowledge will be those that collect a greater number of specific knowledges. For dicent knowledges, a more generic knowledge is the one which allows a greater number of interpretations with truth value “true”. For example, the conjunction of two propositions is more specific, the disjunction of two propositions is more generic, than each of them individually, etc.

In set theory, the most generic set is the universal set. The different subsets of the universe set may be ordered by genericity according to the inclusion relation. If a set A is included in a set B then A is more specific (less generic) than B. Note, however, that the inclusion relation does provide a partial order only. Different subsets of universal set having no common elements cannot be ordered by the inclusion relation. In this case, it makes no sense to speak about genericity. Recalling that a relation is a set, the concept of genericity can be extended to relations.

The concept of genericity is important because it supports information compactation. A generic knowledge may represent number of specific instances knowledge. The more generic the knowledge is, a greater number of specific knowledge it represents. So, in some cases, it is much more interesting to have only one generic knowledge than many specific instances of knowledge.

In our case, the idea of genericity is important because the classification of arguments is based on the genericity relation between the knowledge in the premise and the knowledge in the conclusion of arguments. Arguments in which the knowledge contained in the conclusion is also part of the premise are called analytic or deductive. Otherwise, they are called synthetic. Synthetic arguments include in conclusions knowledge which is not part of the premises. If inclusion is done in a constructive way, e.g., using a distance measure (we include knowledge that is close to some knowledge in premise), the argument is said to be inductive. If inclusion is destructive, i.e. we generate many pieces of knowledge (e.g. randomly) and eliminate those not compatible with the knowledge in the premise, the argument is called abductive. Note that, in this case, inclusion occurs because it is not in disagreement with the knowledge in premise, i.e., it is a plausible inclusion. Sometimes, it is very difficult to determine what is a plausible inclusion. In a general way, plausibility can be determined by a plausibility function and a threshold. If the value of the plausibility function, for a given knowledge, is greater than the threshold, then the knowledge is included. Otherwise, it is not included.

4.2 Formal Model of Argument

An argument can be formally defined as an active object. Objects belonging to the input interface of sn active object will set the premise set. The objects belonging to the output interface of active objects will set the conclusions set. The object activation function is a function mapping premises into conclusions. More formally, we have:

DEFINITION 29 - Argument

Assume an active class A, with tuples of type $(a_1 , \ldots , a_n , f)$ and consider an object $a : T \rightarrow A$, a set P with all elements of a’s input interface, a set C with all elements of a’s output interface. The object a is an argument. The set P is called the premise set of the argument and the set C the conclusion set.

Arguments can either be pure or hybrid. A pure arguments only has input and output fields in its tuples (they do not have internal variables). Hybrid arguments does have internal variables as a result of its integration with rhematic or dicent knowledge. Thus, hybrid arguments are not just arguments, but structures which simultaneously represent an argument knowledge and some sort of dicent or rhematic knowledge. The use of pure arguments or hybrid arguments is due to its particular convenience for some case.

An active object may have more than one function. In this case, it represents not only one, but as many arguments as the number of functions. However, only one function can be triggered at a time instant. This means that, in a given instant, only one argument of such an object will be operative.

[1] The term “genericity” does not exist in the english language. It is introduced here to characterize a measure of generality.

DEFINITION 30 - Deductive Argument

Let A be an active class with tuples of type $(a_1 , ... , a_n , f)$ and assume an argument $a : T \rightarrow A$, the set P of premises of a and the set C of conclusions of a. If, for each instance of a, it is the case that knowledge in C is included in P, then the argument a is deductive (or alternatively, analytic).

DEFINITION 31 - Synthetic Argument

Consider an active class A, with tuples of type $(a_1 , ... , a_n , f)$, an argument $a : T \rightarrow A$, the set P of premises of a and the set C of conclusions of a. If f is such that for any instance of a, there is a an element in C which is not contained in P, then the argument is synthetic.

DEFINITION 32 - Inductive Argument

Assume an active class A, with tuples of type $(a_1 , ... , a_n , f)$, a synthetic argument $a : T \rightarrow A$, the set P of premises of a and the set C of conclusions of a. If f is such that some elements of C are not contained in P, but are generated using elements in P such that they are related to those elements through a distance measure, then the argument is inductive.

DEFINITION 33 - Abductive Argument

Let A be an active class with tuples of type $(a_1 , ... , a_n , f)$. Consider also a synthetic argument $a : T \rightarrow A$, being P the set of premises of a and C the set of conclusions of a, and pl a plausibility function, measuring how much a piece of knowledge is not in contradiction with knowledge in P. If f is such that the knowledge in C but not in P is selected according to pl, then the argument is abductive[2].

Note that the terms "plausibility" and "plausibility function" are not necessarily connected with the "plausibility measure" as introduced by Dempster-Shafer (Shafer, 1976). The meaning of plausibility function here is closer to that in (Collins & Michalski, 1989), and indicates how much plausible (or reasonable) is the knowledge generated by an argument.

The definitions of inductive and abductive arguments are general enough to include many different types of knowledge. In general, we note a complementarity between these two arguments. The inductive argument is constructive in the sense that it generates new knowledge from knowledge contained in its premises. These new knowledge need not be verified further because they are close to known knowledge which in principle guarantee its plausibility. The abductive argument is destructive. It does not matter how new knowledge is generated, because it must be verified before to be accpeted. Unplausible knowledges is not accepted. Note that, since it is not necessary to specify how knowledge is derived, it can be inductively generated. In this case an abductive argument has inductive components. The key point in abductive arguments is that knowledge in premises is used to test the plausibility of the new knowledge to be included in conclusion. If they are also used to generate conclusion we may say that the argument is at the same time inductive and abductive. Normally, arguments of such type are important when the distance from the newly generated knowledge to the knowledge in the premises is not too small (the similarity between those knowledges is small). In these cases, despite its inductive construction, plausibility can be jeopardized. Thus, testing its plausibility is a must, resulting in inductive-abductive arguments. In some cases, it is more important to find the most plausible knowledge. In this case, a plausibility function is used to e.g., eliminate all except the most plausible knowledge. This is an extreme kind of abduction, used for instance, in genetic algorithms and by (Bylander et.al. 1991). Additional examples of different types of arguments can be found in (Gudwin, 1996).

5. Intelligent Systems Organization

According to Albus (1991), an intelligent system is composed by four interconnected modules, respectively : sensorial perception (SP), world model (WM), value judgment (VJ) and behavior generation (BG), as depicted in figure 6:

[2] In (Bylander et.al., 1991), an alternative definition of abduction can be found. The scope of Bylander's definition is, however, more restrictive.

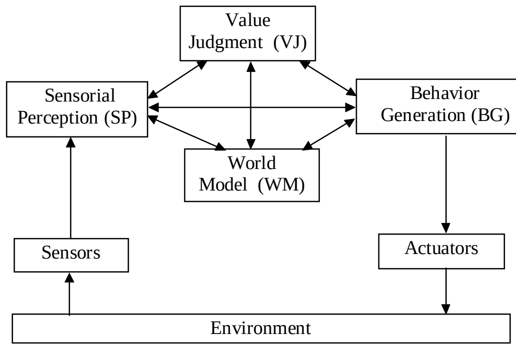


Figure 6 - Intelligent Systems Modules

The SP module compares input from sensors with predictions from WM. In SP the similarities and differences between the observations and the predictions are integrated in time and space, in an attempt to detect events and to match patterns, objects and relationships from the environment. Data from a field of sensors, during extended periods of time, suffer a fusion process, generating a consistent and unique perception of the state of the environment. Algorithms in SP should compute the distance, shape and orientation, plus surface characteristics and physical and dynamic attributes of objects and regions of space.

The WM module represents, at each time, the best estimation of environment state as seen by the intelligent system. In WM there exists a data base about the environment, plus an information storage and retrieving system operating over this data base. By means of a simulation process, WM is capable to generate expectations and predictions. Thus, WM is able to accept requests about the past, the present and the expected future of the environment state. It provides these services to the BG module to generate plans and to select behaviors. It also serves the SP module to generate correlations, comparisons with the world model and the recognition of states, objects and events, based on the world model. In the same way, it serves VJ module to compute values as costs, benefits, risk, uncertainty, importance and attractiveness, etc. The WM module is kept actualized by means of information from the SP module.

The VJ module determines what is good or bad, reward and punishment, trivial or important, probable or improbable. VJ performs a general evaluation of the current environment state and the predictions provided by the hypothetical plans from BG. It computes factors such as cost, risk and benefits for both the observed situation and the planned activities. It assigns to state variables measures of probability, correctness, credibility and uncertainty. It also assigns measures of attractiveness or repulsiveness to objects, events and regions of space. Therefore, the VJ module provides a criteria for decision making, selecting some actions against others. Without a value system, no intelligent system can reach its targets.

The BG module is capable of generating plans, goals and targets and executing tasks. Tasks are recursively decomposed in sub-tasks, and these are scheduled to let the system reach its goals. Targets are selected and plans generated by means of a cyclic interaction among BG, WM and VJ modules. The BG module generates plans, WM module predicts the results of these plans and VJ module evaluates the results. BG selects the best plans and executes them through actuators. It also supervises the execution of past plans, modifying them whenever necessary.

Note that there exists a very close relationship between the ideas of Albus and the concepts of designative, appraisive and prescriptive knowledges discussed in part I (Gudwin & Gomide, 1997). The SP and WM modules are akin to designative knowledge, i.e. sensorial, object or occurrence rhematic knowledges, as well as dicent knowledge when used to represent current and past states, and to predict future behavior. The VJ module is alike appraisive knowledge, usually sensorial rhematic knowledge, or dicent knowledge, when used to do evaluations about some designative knowledge. The BG module is related with designative and prescriptive knowledge. The designative knowledge can canstruct plans and set targets to be reached. The prescriptive knowledge is used to effectively generate a behavior, sending data to actuators. Therefore, from the perspective of knowledge as viewed in this paper, an architecture of an intelligent system can be as depicted in figure 7.

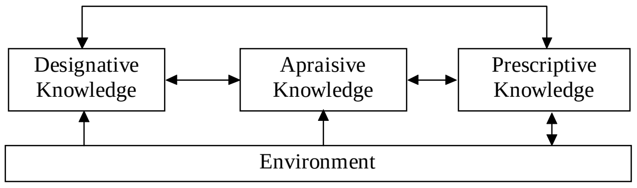
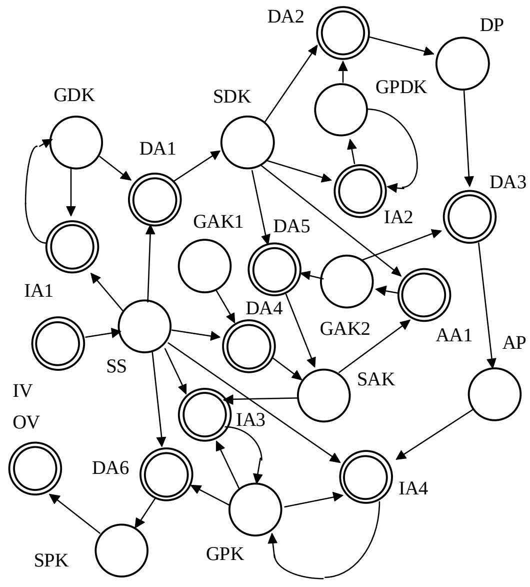


Figure 7 - Architecture for an Intelligent System

As discussed previously, in figure 7 knowledge ise represented by objects (recall that objects are also used to represent generic objects, fuzzy objects, meta-objects, generic meta-objects and fuzzy meta-objects) placed in the proper positions. To provide interaction among its components, argumentative knowledge (in the form of active objects) can be used. The result is an object network that behaves like an intelligent system in the very same sense of Albus.

For illustrative purposes, a general example of an intelligent system is shown in figure 8.



Legend			
IV	Input Vector	GPDK	Generic Predictive Designative Knowledge
OV	Output Vector		
SS	Sensorial Space	DP	Designative Prediction
DAn	Deductive Argument n	GAKn	Generic Appraisive Knowledge n
IAn	Inductive Argument n	SAK	Specific Appraisive Knowledge
AAn	Abductive Argument n	AP	Appraisive Prediction
GDK	Generic Designative Knowledge	GPK	Generic Prescriptive Knowledge
SDK	Specific Designative Knowledge	SPK	Specific Prescriptive Knowledge

Figure 8 - Example of an Intelligent System

In figure 8, the source object at IV (Input Vector) feeds the place SS (Sensorial Space) with information from the input vector. An inductive argument (IA1) uses this information to modify the objects in GDK (Generic Designative Knowledge). The objects in GDK are generic objects whose purpose is to store designative patterns that occur in the input vector. Using a deductive argument (DA1), the objects at SS are compared with objects in GDK to generate objects at SDK (Specific Designative Knowledges), that is:. the specific patterns detected at a given time. The objects at GPDK (Generic Predictive Designative Knowledge) correspond to generic objects which correlate sequences of objects occurring in SDK to provide predictions. These objects are used to determine a designative prediction (DP). In other words, a prediction of the objects that should be in SDK using the deductive argument DA2. Objects in GPDK are generated and modified according to the sequence of objects in SDK, using the inductive argument IA2. The partial object network composed by IA1, GDK, DA1, SDK, IA2, GPDK, DA2 and DP corresponds to the designative module in figure 7.

Analogously, from the object in SS and from a generic appraisive knowledge (in GAK1), the deductive argument DA4 generates a specific appraisive knowledge in SAK. We assume knowledge in GAK1 as inborn of the intelligent system (because it is tied to the objectives of the intelligent system). However, a series of generic appraisive knowledge in GAK2 can be learned, from the correlation of instances in SDK and SAK. An abductive argument (AA1) observes the instances in SDK and SAK and modifies GAK2. The designative knowledges in SDK generate specific appraisive knowledge in SAK using the deductive argument DA5. From the designative predictions in DP, the same generic appraisive knowledge are used to generate appraisive predictions in AP, using deductive argument DA3. The partial object network composed by DA4, GAK1, SAK, AA1, GAK2, DA3 and AP corresponds to the appraisive module of figure 7. The arguments in DA4 and DA5 correspond to the arcs connecting the designative and appraisive modules of the figure 7.

The sensorial input in SS is also used, together, with a generic prescriptive knowledge in GPK, to generate a specific prescriptive knowledge in SPK using the deductive argument in DA6. The prescriptive knowledge in SPK is then consumed by the output vector (OV) to finish the cycle. The generic prescriptive knowledge in GPK is always updated using the sensorial input in SS, the specific appraisive knowledge in SAK, using the inductive argument IA3. Similarly GPK is updated by SS, the appraisive predictions in AP, and the inductive argument in IA4. The partial object network composed by SPK, DA6, GPK, IA3 and IA4 corresponds to the prescriptive module of figure 7. The arguments in IA3 and IA4 are the arcs connecting the appraisive and prescriptive modules.

As it can be seen, as long as different types of knowledge are used (specially when involving learning), the intelligent system representation becomes more complicated. In this example, only sensorial rhematic knowledge (specific and generic), and argumentative knowledges were used. Networks to implement more sophisticated types of knowledge and other kinds of interrelationships may become more complicated.

6. Application Example : Autonomous Vehicle Navigation

In this section, we present an application example to illustrate the use of computational semiotics to develop an intelligent control system for an autonomous vehicle.

Autonomous vehicle navigation is a well known problem in the literature, but many challenges still remain. For recent papers see (Wang et.al. 1991; Verschure et.a. 1992; Fan & Lui 1994; Spence & Hutchinson, 1995; Krozel & Andrisani II 1995; Beom & Cho, 1995; Rao, 1995; Yen & Pfluger, 1995).

Briefly, the problem is as follows . An autonomous vehicle must navigate in an unknown environment, with obstacles and targets. The question is on how to control the vehicle to avoid obstacles and to reach the targets.

The different approaches to tackle this problem may be classified in three categories. In the first (and oldest) the problem is solved using a model of the environment, describing the obstacles and the targets. The vehicle geometry is not considered in this case, and the least circle involving the vehicle is added to the obstacles border. With this, the problem is simplified and the issue is to find a feasible trajectory of a point, using a searching procedure. An example of this approach is found in (Krozel & Andrisani II, 1995). The vehicle geometry may be considered, whenever the vehicle itself can navigate in parts of the environment where the least circle cannot (Fan & Lui 1994). More sophisticated examples include environments with moving obstacles (Spence & Hutchinson, 1995).

The second approach does not use a model of the environment, but a reactive behavior to sensor stimuli (Brooks, 1991). Brooks approach is based on previously programmed behaviors, which are actively selected according to sensor stimuli. Chen & Trivedi (1995) introduced a general scheme for sensor based trajectory planning and the navigation problem. Alternative sensor based approaches do not plan trajectories, but combines two basic behaviors: obstacle avoidance and target seeking. In this case, from sensor data, a control action is found via fusion of both behaviors. This is different than Brook's approach, where there was not a combination of multiple behaviors, but a selection of one of them. Examples include (Yen & Pfluger, 1995; Beom & Cho, 1995). The basic behaviors can be either pre-programmed, or be learned from the interaction between the vehicle and the environment. Examples addressing learning can be found in (Verschure et.al. 1991; Oliveira et.al. 1994; Fabro 1996).

The third approach for navigation problems is an hybrid scheme combining methods of the first and the second. In this case, sensors are used to construct an incremental world model, from which a trajectory is planned. An example is found in (Rao, 1995).

There is an intense debate concerning the different approaches, mainly about the question if a world model is really necessary to solve the problem satisfactorilly. Brooks (1991) argues that a world model in not mandatory, and that with some number of adequate behaviors, the problem can be effectively solved only using the reactive approach. He claims that an emergent behavior arises due to the integration of the different behaviors with environment stimulation. This emergent behavior would be, according Brooks, capable to satisfactorily solve the navigation problem. Considering approaches like from Verschure (1992), where we not only have diverse behaviors, but those behaviors are even learnt due to the interaction with the environment, and more than one behavior can be simultaneously active for the determination of a control action (implementing a fusion of behaviors), there seems to be quite evidence that Brook's argument is plausible.

Behind this controversy there are basically two decision making models. The first, a logic model, provides mechanisms to predict alternatives considering not only immediate decisions, but a sequence of decisions to achieve the goal. Goal achievement is evaluated against a world model. The second, an intuitive model, follows the same basic principles, but only immediate decisions are evaluated through heuristics.

It is worth to note that both decision making models can be ultimately seen as search procedures. The logic model performs search in its whole knowledge space to create a decision tree and to find the best sequence of decisions. In other words, logic models perceives the problem globally. Generally speaking, this means that some sort of memory (global information) is used and decisions are globally derived. The intuitive model performs a more restrictive search, because only one step is done. In contrast with logic models, the intuitive models perceive the problem locally (based on local information), which means that memory is not needed and decision are only locally consistent. In simple situations the intuitive model can be successfully used. But, despite the heuristics quality, there will always exist cases where only local information is not sufficient. In these cases, it may be essential to consider global information, a characteristic of the logic model. Hybrid approaches take advantage of logic and intuitive models to interactively build a world model using heuristics not only to take the next decision but to evaluate a whole sequence of decisions. If the sequence of decisions is considered to be suitable, the first decision of this sequence is then used as a control action. An example of an analogous methodology, but applied in the context of discrete event systems control, can be found in (Chung & Lafortune, 1992). An intelligent control system following this philosophy is given in (Gudwin & Gomide, 1994a).

The approach developed here can be regarded as an hybrid approach. Through its sensor system, the vehicle interactively creates a world model. The world model (designative knowledge) is used to determine a heuristics (appraisive knowledge), which in turn is employed to derive the control action (prescriptive knowledge).

6.1 Problem Description

We assume an autonomous vehicle with a sensor and motor system, within an environment with objects of several different characteristics. The vehicle is powered by a rechargeable battery monitored by the sensor system. Depending on its kind, an object within the environment may charge or discharge the battery, when in contact with the vehicle. During its movement, the vehicle must achieve some goals. The current goal may change at any time, depending on which of them is more important at that moment. The basic goals considered here are: environment characterization, preservation of vehicle's physical integrity and preservation of battery charge within operational levels.

We also assume a two level control system. The task of the higher level, is to determine targets. These targets depend on the goals to be achieved. For instance, it may decide on the environment characterization goal to permit the vehicle to update its environment model beyond that currently known. Goal may also be related with battery consumption constraint because along its trajectory, the vehicle must maintain its battery charge within operational levels. When the energy level is too low, the vehicle must contact environment objects which furnish energy. For this purpose, it is necessary to discover this sort of objects, and their location as well. The idea is to drive the vehicle to the closest object of this sort. At the higher hierarchical decision level, the main problem is priority scheduling to decide, at each moment, what is more important: to explore the environment or to charge the batteries. The lower hierarchical level task is to execute direct control to drive vehicle from its current position to the the target. During its movements, it is necessary to keep the vehicle physical integrity, by avoiding collisions with rigid objects and those that drain energy from batteries. The intelligent controler described here will be restricted to the lower level control task only.

6.2 Vehicle Description

We consider a vehicle with three types of sensors: remote information sensor, contact sensor and battery charge sensor, respectivelly. Actuators position the remote information sensor and the steering angle of the front wheels and sets vehicle velocity. Additional attributes of the vehicle state are indirectly determined from sensors and actuators information.

6.2.1 Remote Information Sensor

This sensor is a simplification for a vision system. Basically, it is an array of (8 × 8) sensors that cover a rectangular area, up to a maximum distance. It can at the same time, provide color and location information about the environment objects it senses. It does not determine the objects characteristics. The sensor rectangular area can be focused to any position, limited to a maximum distance and around the vehicle (see figure 9).

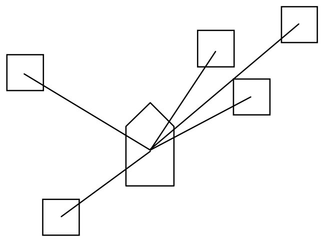


Figure 9 - Example of Focusing Remote Sensors

6.2.2 Contact Sensors

Four contact sensors are located in the 4 extremities of the vehicle. Their purpose is to sense object intrinsic characteristics. Basically, objects can be classified as pleasant or unpleasant, depending on the stimuli received from the contact sensors. They provide information about favorable or disfavorable situations, depending on the goals.

6.2.3 Battery Charge Sensor

This sensor measures the battery charge level, in percental terms. If batteries are totally discharged the sensor will measure 0. Whereas, if totally charged, it will measure 100. The batteries have a slow and linear discharge rate over time. However, when the vehicle touches some specific kinds of objects, a faster discharge or a faster charge may occur. These characteristics can be perceived tracing the temporal behavior of this sensor.

6.2.4 Remote Sensors Position Actuators

The position of remote informations sensor is set by position actuators. There are two: the first sets the angle with respect to the vehicle longitudinal axis (φ), and the second, the length (ρ) to be covered (focused) (see figure 10).

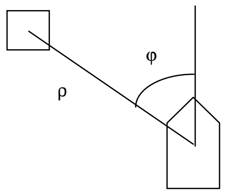


Figure 10 - Remote Sensors Position Actuators

6.2.5 Vehicle Movement Actuators

Two movement actuators drive the vehicle. The first provides the traction to drive the vehicle with a nominal velocity v. Depending on v being positive, negative or zero, the vehicle moves forward, backward or stops, respectively. The value of v does not refer to the real vehicle velocity, but the velocity that it would be if friction were null. For non-null friction values, the real velocity is less than the nominal. Friction is determined according to the environment object properties in which the vehicle is navigating. The second actuator sets the steering angle θ of the wheels taking the vehicle longitudinal axis as a reference (figure 12). The steering angle is such that -45° ≤ θ ≤ +45°.

6.3 Environment's Description

The environment is assumed to be a rectangle delimitated by walls, in which resides many objects. The objects are characterized by their physical properties: color, hardness, taste and energy transfer factor. Each object may have a different color, assumed to be detectable by the remote information sensor. Hardness measures the capacity of the vehicle to cross the objects. Different grades of hardness are associated with different grades of vehicle mobility. Hardness values may vary from 0 to 1. Objects with hardness equal to 1 are solid objects which means that vehicle can not transpose it. The taste of an object models the grade of desire or repulse (pleasure or displeasure) when the vehicle contact the object. This "taste" is detected by contact sensors, with values varying from -1 to 1: -1 means displeasure wheras 1 means pleasure. A null value means indifference. Energy transfer factor models the capacity to furnish to, or to consume energy from the vehicle when in contact. Positive values of energy transfer factor correspond to increase battery energy and negative values to decrease energy. This factor is within -1 to 1 range. Hardness, taste and energy transfer factor have a direct relationship with vehicle color. Therefore, we assume that, if we determine the color of an object, then we get at the same time its hardness, taste and energy transfer factor.

6.4 Vehicle Dynamic Model

The vehicle dynamic model is as follows. Starting with the vehicle coordinates, as shown in figure 11, the remaining parameters are battery load f, friction μ, nominal velocity v, distance between axis D and the wheel angle θ (figure 12).

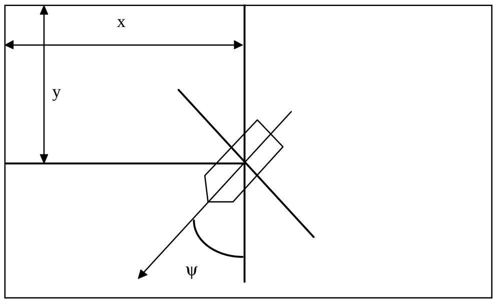


Figure 11 - Vehicle's Coordinate System

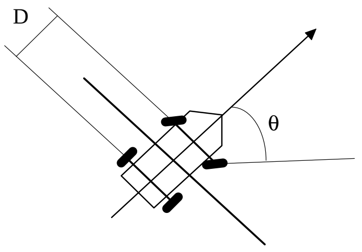


Figure 12 - Distance between Axis and Wheel Angle

A nonlinear model is adopted as detailed below:

If θ = 0, then

$$x(k+1) = x(k) + (1-\mu).v.\cos\left(\psi + \pi/2\right)$$

$$y(k+1) = y(k) + (1-\mu).v.\mathrm{sen}\left(\psi + \pi/2\right)$$

$$\psi(k+1) = \psi(k)$$

$$f(k+1) = f(k) - 10^{-5} + 10^{-2} . \Delta f$$

If θ ≠ 0, then

$$x(k+1) = x(k) + \frac{D.\mathrm{sen}(\Delta\psi)}{\mathrm{sen}(\theta)}.\cos\left(\psi + \pi/2\right) - \frac{D.\left(1-\cos(\Delta\psi)\right)}{\mathrm{sen}(\theta)}.\mathrm{sen}\left(\psi + \pi/2\right)$$

$$y(k+1) = y(k) + \frac{D.\mathrm{sen}(\Delta\psi)}{\mathrm{sen}(\theta)}.\mathrm{sen}\left(\psi + \pi/2\right) + \frac{D.\left(1-\cos(\Delta\psi)\right)}{\mathrm{sen}(\theta)}.\cos\left(\psi + \pi/2\right)$$

$$\psi(k+1) = \psi(k) + \Delta\psi$$

$$f(k+1) = f(k) - 10^{-5} + 10^{-2} . \Delta f$$

where $\Delta\psi = \frac{(1-\mu).v.\mathrm{sen}(\theta)}{D}$ and k is the discrete time.

The parameter D is always constant, but the parameter μ is not. It is determined by the following procedure. If moving forward (positive nominal velocity), we measure the hardness of objects in the left and right front of the vehicle (at the point of the contact sensors), and we take the maximum as the value for μ. This procedure assures that, if one of the points has $\mu = 1$, then the vehicle will not move forward (there is a solid object there). If there is a frontal collision, the vehicle can be driven backwards, to move out of the collision situation. If the vehicle is moving backwards, the procedure to find μ is similar, but considering the hardness of objects in the rear. Thus, in the case of a back collision, the vehicle can move forward. Non-controllable variable is Δf, which corresponds to the energy transfer to the batteries. This variable is calculated in the same way as done for friction. However, in this case we consider the four points of the rectangle involving the vehicle. The energy transfer factor for the four points is calculated and actual Δf is computed as the mean of these 4 values.

7. Intelligent Control System Development

The controller input is the following tuple: $(x,y,\psi,f,v,\theta,\rho,\varphi,c,s)$, where x,y,ψ (pitch angle) are the vehicle position, f is the battery load (in %), v is the nominal velocity in last step, θ is the steering angle, in the last step, ρ is the distance from the vehicle's center to the remote sensors in the last step, φ is the angle of remote sensors in the last step, c is a tuple with the 4 contact sensor values (c_1, c_2, c_3, c_4), with c_1 measuring the taste in right rear extremity, c_2 in left rear extremity, c_3 in the left front extremity and c_4 in the right front extremity and s is an array with 64 visual sensors $((s_{11}, ..., s_{18}),(s_{21}, ..., s_{28}), ..., (s_{81}, ..., s_{88}))$, where s_{11} refers to the top left corner and s_{88} refers to the right bottom corner of a rectangle with side L, the vehicle visual field. The visual sensors are equally spaced within the array

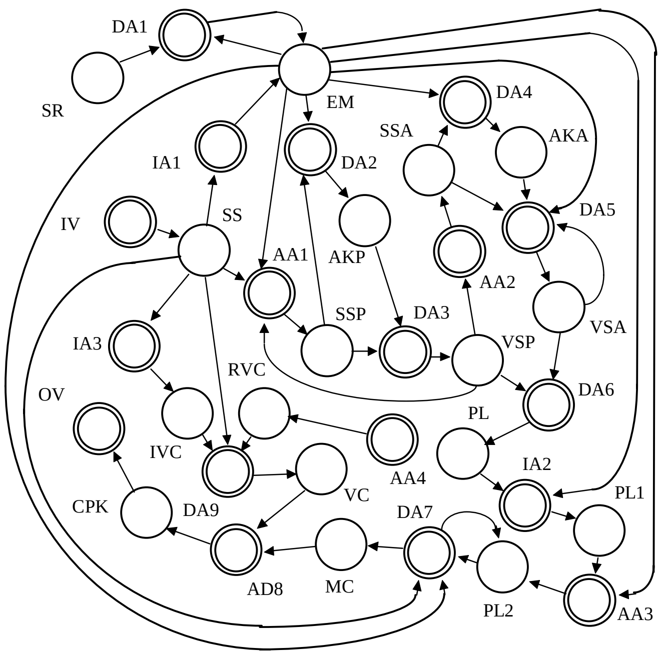
The controller output is the following tuple: (v,θ, ρ,φ), where v ,the nominal velocity and θ, the steering angle of the wheels, relates to the motor control, and ρ, the distance to the remote sensor and φ, the angle of remote sensors composes the visual control.

The control system was developed using the object network whose architecture is depicted in figure 13. The detailed description of each part of the network is postponed unitl next section. Before, we provide a global view of the system to clarify the idea and concepts introduced so far.

The input vector (IV) corresponds to an argument which generates sensor inputs and sends them to the sensor space (SS). By means of an inductive argument 1 (IA1), the system induces objects models that are put in the place of the environment model (EM). Using a set of rules (SR) these models are integrated by deductive argument 1 (DA1), which substitute old models by an updated integrated model. These steps compose a perception and model formation module. The model stays in (EM).

Using data from the sensor space (SS) and the environment model (EM), abductive argument 1 (AA1) generates a set of suggestion points (SSP) corresponding to possible vehicle sub-targets. Deductive argument 2 (DA2) uses the environment model (EM), evaluates if the points in (SSP) are suitable for vehicle movement, and send its evaluation to (AKP), i.e., the appraisive knowledge about points. Deductive argument 3 (DA3), using the appraisive knowledge in (AKP), decides to send or not the points from (SSP) to the place of valid set of points (VSP). Each point in (VSP) generates new points, due to the abductive argument 1 (AA1), in a way similar to the above. Each two valid points, generates an arc in the set of suggestion arcs (SSA), through abductive argument 2 (AA2). These arcs are a part of a plan being elaborated to drive the vehicle. Each arc represents a possible move from an origin to a destination. This move is evaluated by the deductive argument 4 (DA4), using the environment model (EM), being further sent to the place of appraisive knowledge about arcs (AKA). Despite having its origin and destination as valid points, the corresponding arc may be not valid if the movement crosses an undesirable object of the environment. The evaluation in (AKA) is used by the deductive argument 5 (DA5) to decide to include or not an arc from (SSA) in the valid set of arcs

(VSA). From the valid points in (VSP) and the valid arcs in (VSA), deductive argument 6 (DA6) generates a plan, which is sent to (PL). Inductive argument 2 (IA2) tries to optimize the plan by changing two arcs for a third if it takes the vehicle to the same place and have a better evaluation.



Legend			
IV	Input Vector	SSA	Suggestion Set of Arcs
OV	Output Vector	AKA	Appraisive Knowledge about Arcs
SS	Sensorial Space	VSA	Valid Set of Arcs
DAn	Deductive Argument n	PLn	Plan n
IAn	Inductive Argument n	MC	Motor Control
AAn	Abductive Argument n	RVC	Random Visual Control
EM	Environment Model	IVC	Induced Visual Control
SSP	Suggestion Set of Points	VC	Visual Control
AKP	Appraisive Knowledge about Points	CPK	Control Prescriptive Knowledge
VSP	Valid Set of Points	SR	Set of Rules

Figure 13 - Object Network for the Autonomous Vehicle Control

The optimized plan is then sent to (PL1). The abductive argument 3 (AA3) generates for each point composing the plan in (PL1), a set of new points close to the original ones in an attempt to detect if they are more appropriate, according to the current environment model in (EM). This argument will next generate a new plan and send it to (PL2). From this plan, the data from the sensorial space and from the environment model (EM), deductive argument 7 (DA7) generates an object with control information consistent with the plan in (PL2). Those steps integrate the motor control module.

Using data from sensorial space (SS), inductive argument 3 (IA3) generates a suggestion for visual control, that is, a position for the remote information sensor close to last one, compensated to take into account the vehicle movement and modified to focus the sensor on a system object. This suggestion is sent to (IVC). Parallelly, abductive argument 4 (AA4) generates a random suggestion, and send it to the random visual control place (RVC). Based on data in (SS), deductive argument 9 (DA9) decides if the visual control is trying to focus on an environment's object or is seeking for a new object, choosing the control suggestion in (IVC) or (RVC). The selected option is sent then to the place of visual control (VC). The steps above describe the visual control module.

Finally, the visual control (VC) and motor control (MC) are integrated, by the deductive argument 8 (DA8), to generate the control prescriptive knowledge (CPK). This is then sent to the output vector (OV), finishing the intelligent control cycle.

8. Control System Implementation

In this section, we present a description of each step that compound the intelligent control system introduced in last section. To make it easy to understand the steps, they will be presented by means an object oriented language. This representation has the double characteristic of making it easy to understand the system and to illustrate how an objects network may be directly translated and processed in a programming language. The chosen language is C++.

8.1 Input Interface Module

This module corresponds to the objects network section presented in figure 14:

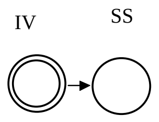


Figure 14 - Input Interface Module

The class associated to SS, tSS is the following:

```
class tSS
{double x;
 double y;
 double pitch;
 double ax,
 double ay;
 double apitch;
 double fuel;
 double vel;
 double dpitch;
 double scangle;
 double sclength
 int s[8][8];
 double c[4];
 double mx;
 double my;
};
```

where we find the following correlation:

x	x	apitch	ψ(t-1)	sclength	ρ
y	y	fuel	f	s	s
pitch	ψ	vel	v	c	c
ax	x(t-1)	dpitch	θ	mx	x target
ay	y(t-1)	scangle	φ	my	y target

Note that more than the variables in the input model, class tSS has also a memory of values x,y and ψ, for last step, and the coordinates of the target to be reached.

The class associated to IV, tIV is the following:

```
class tIV
{public:
     tSS Generate(void)
      {tSS newd;
       newd = CollectDataSensor();
       return(newd);
     }
};
```

Class tIV is a class with an unique function, that returns an object of type tSS, filled with the actual values of sensors. So, at each instant, an object of type tIV that is in IV generates a new object of type tSS, that is put in SS.

8.2 Perception and Environment Modeling Module

This module corresponds to the objects network section presented in figure 15.

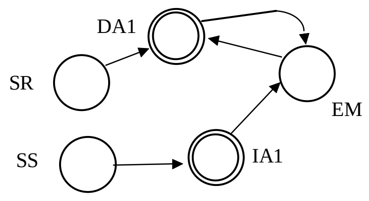


Figure 15 - Perception and Environment Modeling Module

In this module, the environment model is generated. It remains stored in EM, and is widely used by other system's modules.

The class associated to EM, tEM is given by:

```
class tEM
{int x0,y0,x1,y1;
 int color;
 double eval;
};
```

Class tMA models the attributes of typical objects found in environment. In this class, x0,y0, x1 and y1 correspond to the geometric coordinates of a rectangle, that determine the position and size of an environment's object. The field eval models the characteristic taste of the object, being equal to -1 if the object is repulsive, 1 if attractive, or 0 if it doesn't care. Intermediary values correspond to more or less degrees of attraction/repulsiveness. The value of field eval may be known when the contact sensors effectively detect an environment object, or when the system previously associated a color to a taste, being then determined by induction. The field color models the object color, that is seen by the visual sensors.

The class associated to IA1, tIA1 is given by:

```
class tIA1
{public:
 void FindPatterns(tSS ss, place EM);
};
```

Class tIA1 generates objects with only one function, the function FindPatterns, that has as input an object of class tSS, and a place (EM), where it put the generated objects. The function FindPatterns is rather sophisticated. It observes the visual sensors codified in ss and detect the border of all rectangles that are identifiable from the sensor matrix. Its work is illustrated in figure 16.

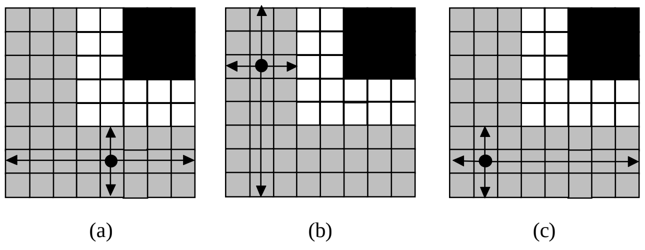


Figure 16 - Example of Pattern Recognition

For each point in the matrix, a sweep is done, first in the horizontal direction and then in the vertical direction, searching for point with the same color of the current point. In cases (a) and (b), this search is simple. Case (c) illustrates a must of the algorithm: the search in the vertical direction may be made for each column in the horizontal direction, taking the less distance found. If it is not done, the point indicated in (c) would return the square of the whole matrix, as the searched point may cause such interpretation, if the search was made only in one column. For the case presented the algorithm finds three rectangles, two gray (identified in (a) and (b)) and one black.

The rectangles found by the function would generate then objects of type tEM, that are put in place EM. Note that the maximum number of rectangles discovered by the function would be 64 (one for each point), i.e. when each color in the matrix of points is different from its neighbor.

After the generation of those elementary objects in EM, the objects in SR and DA1 are responsible for its integration, generating a more consistent model. The place of set of rules (SR) has basically three objects, corresponding to the three rules of integration, that are shown in figure 17.

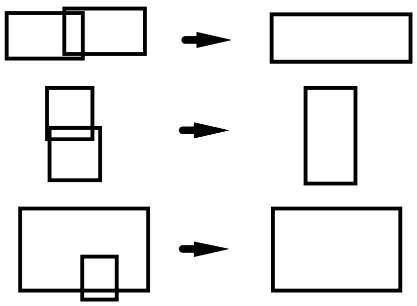


Figure 17 - Rules for Objects Integration

The rules are very simple. The first one says that if two objects are horizontally aligned, they are an unique object. The second rule says that if two objects are vertically aligned, they are also an unique object. And the third rule says that if given object is totally inside another object, it is only a part of the former. In the three rules, objects may not be perfectly aligned, but there is a tolerance, either in alignment or in inclusion. This tolerance is necessary, as the visual sensors are not exact. Each point in the visual sensor is not really a point, but comprises a whole region, and the sensorial result is due to the integration of all real points in this region. With this, there is an intrinsic error in measurement. This is the tolerance used for the integration of objects. Note that the resulting object is always equal to the internal rectangle, in the alignment, to avoid an undesirable effect that occur in the detection of edges. For the inclusion rule, the big rectangle can not be expanded, even if the internal rectangle overflows the limits of the external rectangle, to avoid the same undesirable effect.

The class associated to SR is tSR:

```
class tSR
{int verb;
 int rel1;
 int rel2;
 int function;
};
```

The attributes of objects of type tSR correspond to the conditional propositions, where the terms are codified by its indices. Those indices may be recognizable by the argument in DA1. The semantic corresponds to:

IF verb(rel1,rel2) THEN function(rel1,rel2)

The used rules are the following :
IF halign(rel1,rel2) THEN hadd(rel1,rel2)
IF valign(rel1,rel2) THEN vadd(rel1,rel2)
IF included(rel1,rel2) THEN return(rel1)

The class associated to DA1, tDA1, is given by:

```
class tDA1
{public:
  tEM MergeObject(tEM o1, tEM o2, tSR r);
};
```

An object of type tDA1 takes two objects from EM, o1 and o2, plus a rule from SR, r, consumes the two objects in EM and returns a new object that is put in EM. In the objects network, this operation is done as much times as necessary, until neither three rules are enabled.

With this, we implement the perception and environment modeling module. The object of type tSS that is in SS is used by the object in IA1 to generate objects in EM, corresponding to models of objects from environment that are perceptually captured by the system. As the system is capable of receiving only partial information from system, those models need to be integrated, for that through the knowledge of multiple facets of the real object, we reach a model that is more close to the real object. This task is executed by the rules in SR and by the deductive argument in DA1. All this cycle is realized as soon as new information arrives by means the argument in IA1.

8.3 Points and Arcs Generation Module

This module is a sub-module of control module. It is described separately, due to its complexity. It corresponds to the network section presented in figure 18.

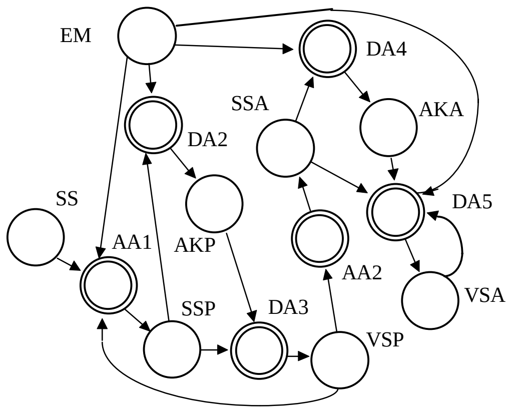


Figure 18 - Points and Arcs Generation Module

The classes associated to SSP and VSP are equivalent, being called tP, or class of points.

```
class tP
{double x;
 double y;
 double p;
 tP *pp;
 int type;
};
```

The attributes of objects of such class correspond basically to the point coordinates (x,y,p), where p is an angle, corresponding to a possible vehicle movement, if it is at point (x,y). The attribute pp corresponds to the father point, i.e. the point used as a source for its generation. The abductive argument in AA! generates suggestion points that are put in SSP. This generation may be in 4 different ways. The way a point is generated is identified by the attribute type. In the first method, we take the target coordinates and use them to generate a point. This method is used to generate a point that should be the final point in every plan, i.e. the own target. Its attribute type is 0 and pp is equal to NULL (it is not considered). In the second method, the vehicle's current position is used to generate a point. This method is used to generate the first point in every plan, i.e. the current position. Its attribute type is 1 and pp is also NULL. In the third method, the argument starts from a valid point, in VSP, to generate a new point. The attribute type is 2 and the address of the source point is put in pp. The fourth method uses points from the environment's objects models to generate points. Its attribute type is 3 and pp is NULL. The class associated to AA1 therefore has 4 functions:

```
class tAA1
{public:
 tP GenerateNewPoint0(tSS src);
 tP GenerateNewPoint1(tSS src);
 tP GenerateNewPoint2(tP src, tEM *em);
 tP GenerateNewPoint3(tEM *em);
};
```

The function GenerateNewPoint0 is trivial. It simply extract from src the target coordinates, put NULL in pp and 0 to type.

The function GenerateNewPoint1 is also trivial. It basically extracts from src the parameters x,y and pitch and put them in x,y and p. The attribute pp is null and type is 1. This concludes the new point definition. This point is then sent to SSP.

The function GenerateNewPoint2 is fair more sophisticated. From a base point, given by src, the argument generates randomly an angle from 0 and 2π. This angle corresponds to an axis that would be used in the generation of the new point, as in figure 19. The function generates then a line, in the direction of the angle, and verifies if, for each object model in EM that is classified as repulsive, if is there any collision. If the line does not cross any such object, the environment's external limits are considered. Once the algorithm finds the closes collision, the new point will be at 50% of the distance from the collision point, according to figure 19.

We only consider, however, the objects that are considered repulsive, i.e. the ones which attribute eval is less than 0. The objects with the attribute eval equal or greater than 0 are not considered, being ignored by the function.

At last the function determines the point coordinates, identifies than its father point (point src), and put its address in pp. It also sets type equal to 2. Generates than a new object of type tP that is sent to SSP.

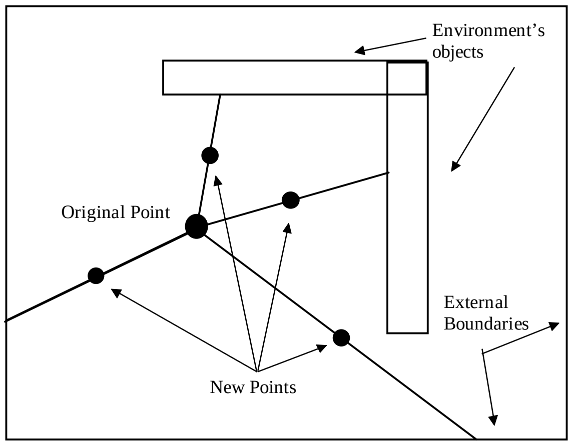


Figure 19 - Generation of New Points

The function GenerateNewPoint3 takes each object from EM and puts a security border over the rectangle vertex, as can be seen in figure 20.

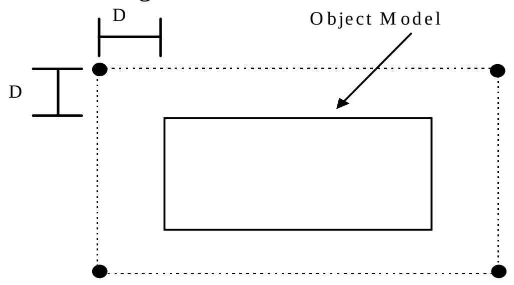


Figure 20 - Generation of New Points according to the Environment Model

This security border usually corresponds to the vehicle's width, for it to travel without danger. This value is, however, a system parameter, and can even be subject to learning.

The function also returns a value NULL for pp and 3 for type.

The place AKP, corresponding to the place with the appraisive knowledge about points do have an associated class tEval:

```
class tEval
{double eval;
};
```

The class associated to DA2 is called tEvalPoint, as its basic function is to evaluate if a given point is suitable or not for the vehicle to be.

```
class tEvalPoint
{public:
   tEval EvalPoint(tP point, tMA *ma);
};
```

The function EvalPoint takes the point "point" and evaluates if it is a suitable point for the vehicle to move to, generating an object of type tEval, corresponding to the result of such evaluation. To provide such evaluation, the function uses the values of "tastes" associated to environment's objects, and the distances from the given point to those objects. The following formulae is used:

$$\mathrm{Eval} = \min_{i}\left(g_i \ . \ e^{(-0.4 \ . \ \mathrm{dto(i)})}\right)$$

In this formulae, g_i corresponds to the taste of object i, from the model, and dto(i) corresponds to the distance from the point to object i. The distance to object i is calculated as shown in figure 21.

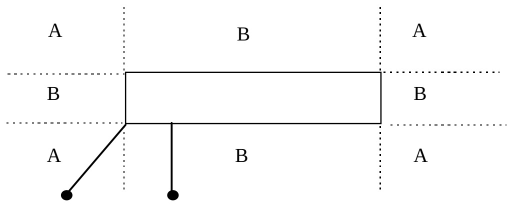


Figure 21 - Distance to Object Computation

If the point is into any of type A regions , then the distance to the object is the distance to the closest rectangle vertex. If the point is into a type B region, then the distance is the distance to the closest rectangle line.

The class associated to DA3, tDA3 is given by:

```
class tDA3
{double threshold;
 public:
 tP TestPoint(tP point, tEval eval);
};
```

Function TestPoint is very simple. It verifies if a point evaluation is higher than a threshold, and if yes, returns the point to the place of valid points (place VSP). Note that either for the acceptance or rejection of a point as a valid point, we used parameters obtained from an empirical way. In a more sophisticated system, these parameters may be subject to learning.

The class associated to SSA (Suggestion Set of Arcs) is tA:

```
class tA
{tP *orig;
 tP *dest;
};
```

The class associated to AA2 is tAA2:

```
class tAA2
{public:
   GenerateNewArc1(tP *p);
   GenerateNewArc2(tP *p1, tP *p2);
};
```

Objects from this class have two functions. Function GenerateNewArc1 has as parameter a point of type 2, that identifies its parent point in itself. The arc is then generated using as orig the address of its parent-point and as dest its own address. The function GenerateNewArc2 may have as a parameter p1, a point of type 1, 2 or 3, and as parameter 2 a point of type 0 or 3. The point generated has as orig the parameter p1 and as dest the parameter p2.

The class associated to AKA is tEval.

The class associated to DA4 is tEvalArc:

```
class tEvalArc
{public:
   tEval EvalArc(tA arc, tEM *em);
};
```

Function EvalArc checks if the orig point is suitable, in the same way as in EvalPoint. After that it checks if the line from orig to dest crosses any object from environment. If there is such a cross, the function considers the object's taste. This procedure is shown if figure 22:

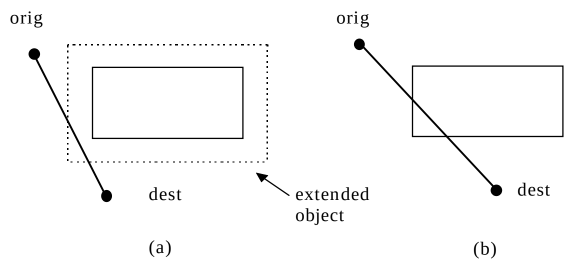


Figure 22 - Arc Evaluation

When the point dest is of type 2, we use an extended model for the object, as in case (a) from figure. When the point dest is of type 3, or is the target, we use case (b) from figure.

The object in AKA (appraisive knowledge of arcs) holds than and evaluation for an arc in SSA. This evaluation measures how suitable is the move from orig to dest for the vehicle.

The class associated to DA5 is tDA5:

```
class tDA5
{double threshold;
 public:
 tA TestArc(tA arc, tEval eval, tA *va, tEM *em);
};
```

Function TestArc is more sophisticated than its dual TestPoint. Beyond checking if the arc evaluation is higher that a threshold, it checks many other situations. From a list of valid arcs (va), obtained from VSA, we test if arc "arc" does not cross any other valid arc (that is already in VSA), if the distance from its points is greater than a minimum value and if the distance from the destination point to the objects considered repulsive (in terms of taste) is greater than a minimum value also. Arcs that satisfy all those conditions are sent then to VSA.

8.4 Plan Generation and Optimization Module

The plan generation and optimization module takes the valid points and arcs, put in VSP and VSA, and generates a plan that is optimized and further transformed into control actions. This module, with the points and arcs generation module integrates the motor control module. The network section corresponding to the plan generation and optimization module is shown in figure 23:

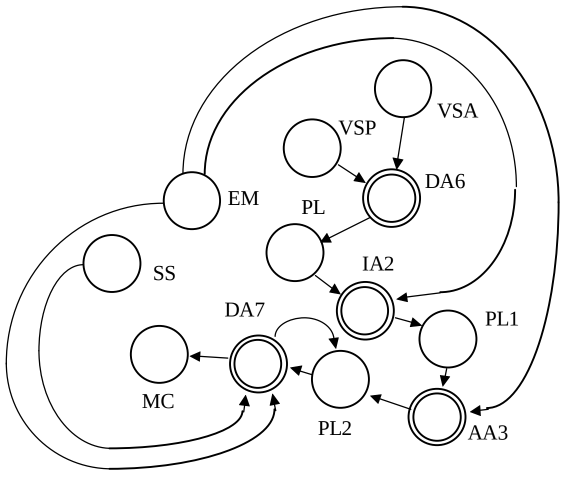


Figure 23 - Plan Generation and Optimization Module

The class associated to PL, PL1 and PL2 is tPL:

```
class tPL
{tP *pl;
 int npl;
};
```

The class associated to DA6 is tDA6:

```
class tAD6
{public:
   tPL GeneratePlan(tP *p, tA *a);
};
```

Function GeneratePlan basically takes the list of arcs in VSA, locate the respective points in VSP and generates a list of points that corresponds to the sequence of points to be followed by the vehicle in its trajectory to the target. To generate that list, it searches the arc that has as its dest the target, and from its orig, searches another arc that has it as dest, and so successively, until reaching

The inductive argument in IA2 takes this plan, and do a first optimization, deleting points that are redundant, as is shown in figure 24:

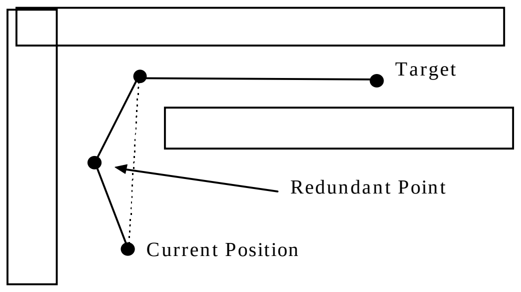


Figure 24 - Redundant Points Deletion

Redundant points are those that, if deleted, still allows a safe move, as is indicated by dotted line in figure.

The class associated to IA2 is tIA2:

```
class tAI2
{public:
  tPL OptimizePlan1(tPL pl, tEM *em);
};
```

The function takes the plan in PL and generates a new plan, without the redundant points, that is put in PL1.

The abductive argument AA3 takes the plan in PL1, and optimizes it, slightly changing its points, as is shown in figure 25:

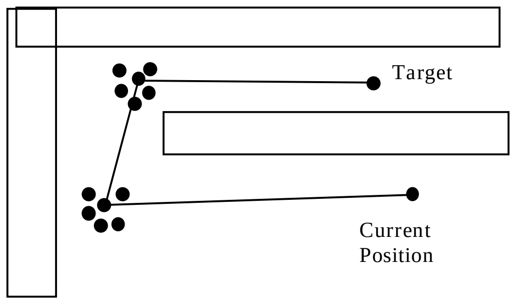


Figure 25 -Plan Points Optimization

The argument randomly generates for each point in plan, different from current position and target, a series of other points, within a limited range. The new plan is tested then, using the environment model, to check if it is a better plan. If any abducted point is best than the original, it is changed.

The class associated to AA3 is tAA3:

```
class tAA3
{public:
   tPL OptimizePlan2(tPL pl, tEM *em);
};
```

The function OptimizePlan2 takes the plan in PL1 and based in model em, generates a new plan that is put in PL2.

The class associated to MC is tMC

```
class tMC
{double vel;
 double dpitch;
};
```

The object in MC corresponds to motor control order to be sent to actuators. It is basically composed by the velocity and wheel's steering angle. The object in MC is generated by the deductive argument in DA7.

The class associated to DA7 is tDA7:

```
class tAD7
{public:
  tCM MotorControl(tPL *pl, tSS ss, tEM *em);
  tPL ActualizePlan(tPL pl);
};
```

The function MotorControl is rather sophisticated. Usually, to reach a given point, the control action to be taken is very simple. It is sufficient to find the polar coordinates of the point to be reached, from the point of view of the vehicle, set a positive velocity and the steering angle equal to the found angle. But, in situations as in figure 25, there is a need of some maneuvering to reach target.

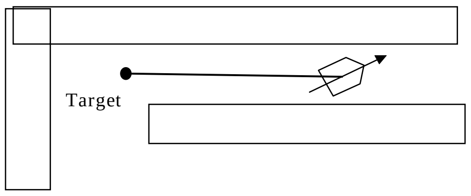


Figure 25 - Situation Needing Maneuvering

To check if a given situation needs maneuvering, the argument calculates the optimum trajectory to reach the target. This optimum trajectory corresponds to set the steering angle to its maximum (in right or left direction, depending o the target position), and let the vehicle move around a circle, until it aligns itself to the target. This position is shown in figure 26:

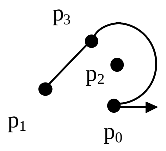


Figure 26 - Vehicle's Minimum Curve

In figure, vehicle starts from a position p_0 moving in the indicated direction, and desires to reach p_1 . The range of the circle that describes the vehicle's trajectory, given the steering angle is given by:

$$R = \frac{D}{2\cos\left(\frac{\pi - \theta}{2}\right)}$$

Supposing a maximum angle of $\pi/4$, the minimum range is:

$$R_{\min} = \frac{D}{2\cos\left(\frac{3\pi}{8}\right)}$$

Point p_2 is calculated using a distance equal to $R_{\min}$ and an angle of $\pi/2$ from the desired direction. Point p_3 is calculated by:

$$p_3^x = p_2^x + R_{\min}\cos\xi$$
$$p_3^y = p_2^y + R_{\min}\,\mathrm{sen}\xi$$

where $\xi = \mathrm{arctg}\left(\frac{p_1^y - p_2^y}{p_1^x - p_2^x}\right) + \frac{R_{\min}}{\sqrt{\left(p_1^y - p_2^y\right)^2 + \left(p_1^x - p_2^x\right)^2}} - \frac{\pi}{2}$

Once p_3 is calculated, the system supposes that the vehicle is over this point, with a pitch value corresponding to the direction from there to p_1 . It then calculates the position of all the 4 contact sensors, if the vehicle was at this point, and determines, using the environment model, which will be the taste sensed if the vehicle was in such position. If the minimum taste of all 4 sensors is acceptable (something about -0.9), this means that the vehicle does not need to maneuver. If not, it sets a maneuvering state. In this state, the vehicle checks which contact sensor indicates the worst taste, and from it, set a sequence of moves from right to left, alternating forward and backward movements, until the vehicle goes out of maneuvering conditions (i.e., until the taste in p_3 becomes acceptable). The output of MotorControl procedure generates an object that is sent to MC.

The function ActualizePlan, starting from the vehicle's current position, checks if the first point in plan was reached, deleting it from the list in such case. The corrected plan is re-sent to PL2.

8.5 *Visual Control Module*

The visual control module is responsible for the movement of the visual sensor, that is used by the perception and environment modeling module. Despite being a separate control, its action is decisive for the perfect work of such module, demanding a coordination among those modules for the perception to be operational.

The network section responsible for visual control is indicated in figure 27:

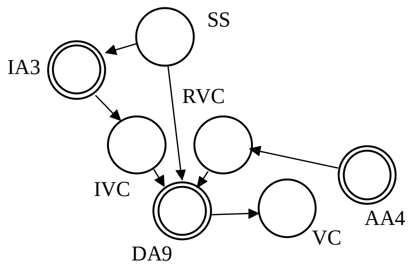


Figure 27 - Visual Control Module

The class associated to IVC, RVC and VC is tVC:

```
class tCV
{double scangle;
 double sclength;
};
```

The class associated to IA3 is tIA3:

```
class tIA3
{public:
   GenerateInducedVisualControl(tSS ss);
};
```

The function GenerateInducedVisualControl is rather sophisticated. Based in the image from the visual sensor, it calculates the center of area of image, and suggests a move equivalent to this center of area. The procedure is shown in figure 28:

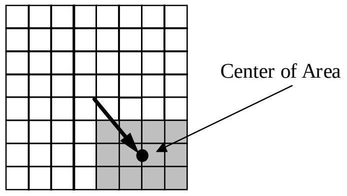


Figure 28 - Center of Area Computation

More than that, the function performs a compensation due to the vehicle's movement. This compensation is shown in figure 29:

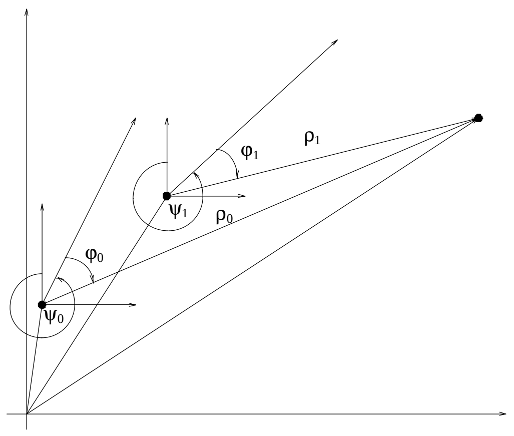


Figure 29 - Compensation for Vehicle's Movement

Basically what we desire here is that, being the vehicle at position (x_0 , y_0 , ψ_0), and having visual control coordinates (ρ_0 , φ_0), when the vehicle changes its position to (x_1 , y_1 , ψ_1), the visual control coordinates (ρ_1 , φ_1), sets that the visual control focus points to the old position. The equations for (ρ_1 , φ_1), are the following:

$$\varphi_1 = \psi_1 - \mathrm{arctg}\left(\frac{y_0 - y_1 + \rho_0 . \mathrm{sen}(\psi_0 - \varphi_0 - 3\pi/2)}{x_0 - x_1 + \rho_0 . \cos(\psi_0 - \varphi_0 - 3\pi/2)} \right) - \frac{3\pi}{2}$$

$$\rho_1 = \sqrt{\left(y_0 - y_1 + \rho_0 . \mathrm{sen}(\psi_0 - \varphi_0 - 3\pi/2)\right)^2 + \left(x_0 - x_1 + \rho_0 . \cos(\psi_0 - \varphi_0 - 3\pi/2)\right)^2}$$

Function output is obtained adding both center of area and compensation for vehicle's movement. The result is sent then to IVC.

The class associated to AA4 is tAA4:

```
class tAA4
{public:
   GenerateAbducedVisualControl(void);
};
```

The function GenerateAbducedVisualControl simply generates a random set of coordinates (ρ_1 , φ_1), having as ρ_1 a value that is compatible with the vehicle’s limits and a φ_1 from $-\pi/4$ to $\pi/4$. The generated object is sent to RVC.

The deductive argument in DA9 decides from sensor data if it will use the object in IVC or RVC. If the system is focusing in some object, it uses the object in IVC. If it is searching for an object, it uses the object in RVC.

The class associated to DA9 is tDA9:

```
class tAD9
{public:
   tVC VisualControl(tVC ivc, tVC rvc, tSS ss);
};
```

Function VisualControl does the task of deciding between the two control suggestions, sending its decision to VC.

8.6 Output Interface Module

In this module, motor control and visual control are integrated, resulting a prescriptive knowledge, that is sent to actuators.

The network section corresponding to such module is indicated in figure 29:

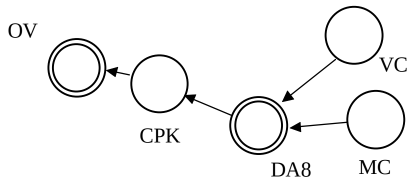


Figure 29 - Output Interface Module

The class associated to CPK (control prescriptive knowledge is tCPK:

```
class tCPK
{double vel;
 double dpitch;
 double sclength;
 double scangle;
};
```

The class associated to DA8 is tDA8:

```
class tDA8
{public:
   tCPK GenerateCPK(tVC vc, tMC mc);
};
```

The function GenerateCPK takes the objects in VC and MC, corresponding to the partial determination of visual control and motor control, and integrates them in a single object of type tCPK. This object is then sent to CPK.

The class associated to OV is tOV:

```
class tOV
{public:
   void SendToActuators(tCPK cpk);
}
```

The deductive argument in OV simply takes the object corresponding to the control prescriptive knowledge in CPK and send the right values to system’s actuators.

9. Coordination of System Objects

To develop an object network we must specify the object classes and the selection functions for each object as well. The right choice of selection functions is the key point to achieve coordination among the different objects composing the network.

There are many forms in which selection functions can be determined. In some cases (as shown in part I of the paper), we assign independent selection functions for each active object, expecting the system to achieve coordinationa autonomously. In other cases, we may enforce some sort of coordination designing selection functions specifically for this purpose. We discuss this issue next.

The first objects to be triggered are the visual control objects (i.e. those in the visual control module). Initially, the system uses the random visual control (i.e. the objects in RVC), until an image capturing an environment object is detected by the visual sensors. At this instant, the system starts a focusing task (by taking objects from IVC instead form RVC), to get the best position to perceive the object. During random visual control and focusing, the perceptive system (i.e. the argument in IA1) remains disabled. When visual control finishes focusing (i.e. the center of area for the image stays below a given threshold), the perceptive system is enabled. Then, the argument in IA1 generates a model for the focused object and the argument in DA1 tries to integrate such model to others that are already in EM. Once perception is terminated, the visual control system returns to random control, repeating the previous steps. This is the coordination at the level of visual control and perception.

Motor control, has an independent coordination. Initially, from sensorial input, the argument in AA1 generates origin and target points. These points are checked by the arguments in DA2 and DA3 going to VSP. After, the argument in AA2 attempts to produce an arc from origin to target. If it succeeds, a plan is generated in PL. If not the argument in AA1 produces new points starting from the origin point. First, it uses GenerateNewPoint2, and next GenerateNewPoint3. The argument in AA2 then tries to find an arc from each of these points to the target. If it succeeds, a plan is generated in PL by the argument in DA6. Otherwise, each of these points are used to generate new points until an arc to the target is produced, generating a plan in PL. At this time, execution of Points and Arcs Generation Module ceases, and the arguments in IA2 and AA3 are triggered. IA2 first optimize the plan, putting it in PL1 and AA3 performs a second optimization, sending the plan to PL2.

The argument in DA7 works in parallel with the other objects. From a plan in PL2, the argument produces a motor control and send it to MC. The output interface module is also independent. At each time instant, it collects the objects in VC and MC and provides the prescriptive control knowledge, the actual control decision, that is sent to actuators.

It should be stressed that there are four parts of the object network working in parallel. The first is the visual control and perception, the second, the plan generation, (comprising points and arcs generation, planning and plan optimization), the third is the generation of motor control, from the optimized plan, and the fourth is the generation of prescriptive control knowledge. Each of these parts, however, does have a well established sequence. The selection functions of the objects consider such a sequence to guarantee that the object network, as a whole, work as desired.

10. Simulation Results

In this section, simulation results are presented to illustrate the performance of the control system developed within the framework of computational semiotics.

The experiment is depicted in figure 30. The first frame shows the vehicle position when simulation starts. In second frame, the vehicle chooses a plan, based on its current environment model (at this time it is none) it chooses a plan, that is, it decides to proceed straight to the target. After the walls hidding the target are perceived by the visual system, the vehicle looks for an alternative path. The third frame shows the vehicle at a later time instant, when now it has an updated environment model, as indicated by the rectangles with dark contours. Using the current model, an alternative plan is generated, but it will be found to provoke a collision with environment walls currently unknown by the vehicle. After a number of steps travelling, a good environment model is constructed and the system easily develop a plan to achieve the target, without any collision with environment’s walls.

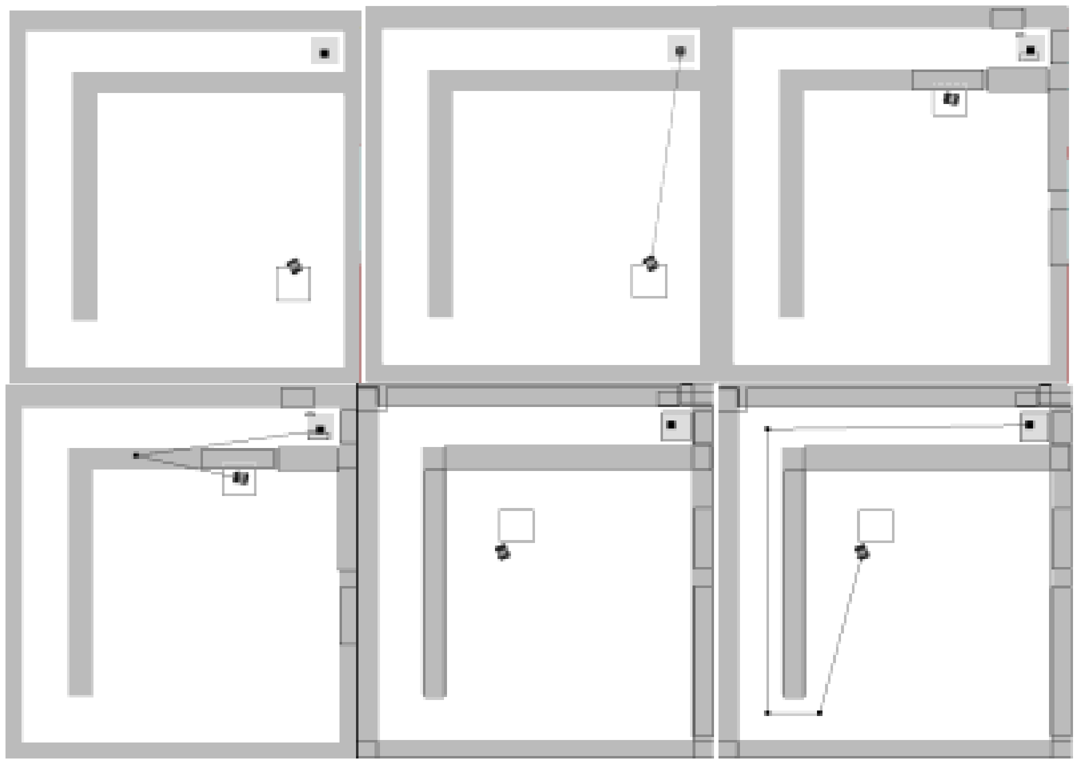


Figure 30 - Simulation Results

The specific environment used in the example may, at first glance, looks simple. But it should be noted that the main aim of the control system is to find a target without colliding with obstacles within an unknown environment. For this purpose, the goals to be achieved may become antagonic. For instance, in figure 30, to effectivelly reach the target, the vehicle must first move in a direction opposed to the one where the target is, as the sixth frame of figure 30 shows. This is not the case when there is no obstacle between the current vehicle position and the target. This is critical for sensor based, pure reactive (intuitive) approaches because if decisions are made based on local information only, the vehicle can be either trapped, or remain in a dynamic behavior which will never lead it to the target position. This is shown in figure 31 (Fabro, 1996)

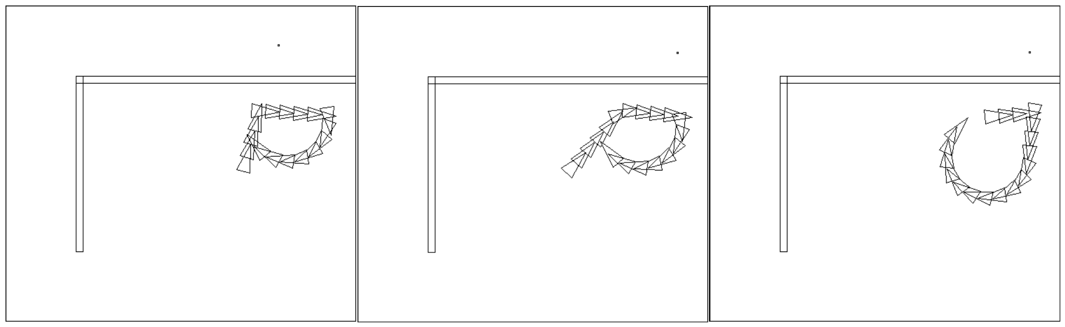


Figure 31 - Simulation using a Pure Reactive System

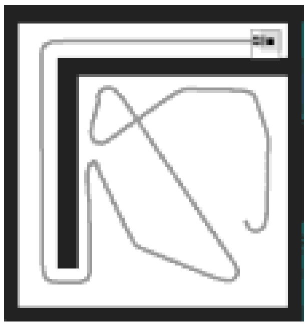


Figure 32 - Complete Trajectory using Object Network Control

The complete trajectory due to the objects network based control is shown in figure 32.

A number of different environment topologies have been used in a series of simulations experiments. The intelligent control here implemented via the object network achieved a quite good performance reaching the target without collision in all of them.

11. Conclusions

In this paper we completed the formalization of the semiotic concepts introduced in part I and an application example was detailed not only to clarify the main issues, but also to show the potential of the aproach proposed.

The main contributions of this work are the following: the elaboration of a semiotic taxonomy of knowledge, the introduction of a formal model of objects, the development of a new tool, the object network, as a paradigm for complex systems, the formalization of different types of knowledge by means of object networks, the identification of object networks as a suitable modeling tool to develop intelligent systems and an example of an intelligent control for an autonomous vehicle.

Despite these contributions, the approach herein developed clearly have limitations. For instance, the taxonomy of knowledge types presented is only partial. Originally, Peirce identifies more than 100 different types of signs, which eventually may imply in many and different types of knowledge. Nevertheless, the knowledge types organization addressed here does include the most frequently used in building practival knowledge-based systems.

The scheme presented as a formalization of objects, does not aim at a general theory for objects, appears as a foundation for such a theory. Some extensions are clearly needed here too, e.g. to handle the asynchronous interaction among objects.

The object network developed from the formal concept of objects, despite its power, still have many limitations. For example, analysis tools still are underdeveloped when compared to other modeling tools (e.g., Petri Nets). Indeed, very few systems have been modeled whith this formalism so far. We did not provide a formal representation for the types of knowledge not covered here. As new types of knowledge arise, new formal definitions will be demanded.

For continuing the consolidation of object networks as a key tool for modeling intelligent systems, it is still necessary to solve a broader class of problems to emphasize its benefits and to identify opportunities for extensions.

12. References

(Albus, J.S. 1991) - “Outline for a Theory of Intelligence” - IEEE Transactions on Systems, Man and Cybernetics, vol. 21, n. 3, May/June 1991.

(Beom, H.R.; Cho, H.S. 1995) - “A Sensor-Based Navigation for a Mobile Robot Using Fuzzy Logic and Reinforcement Learning”, IEEE Transactions on Systems, Man and Cybernetics, vol. 25, n. 3, March 1995.

(Bolc, L. 1987) - “Computational Models of Learning” - Springer-Verlag, Berlin

(Brooks, R.A. 1991) - “Intelligence Without Reason” - Proceedings of the Twelfth International Conference on Artificial Intelligence, Vol. 1 (1991) 569-595.

(Bylander, T.; Allemang, D.; Tanner, M.C.; Josephson, J.R. 1991) - “The Computational Complexity of Abduction” - Artificial Intelligence 49 (1991) 25-60.

(Carbonell, J.G. 1983) - “Learning by Analogy: Formulating and Generalizing Plans from Past Experience” - in Michalski et.al. *Machine Learning - An Artificial Intelligence Approach* - Morgan Kaufmann Publishers, Inc. pp 83-134.

(Chen, C.X.; Trivedi, M.M. 1995) - “Task Planning and Action Coordination in Integrated Sensor-Based Robots”, IEEE Transactions on Systems, Man and Cybernetics, vol. 25, n.4, April 1995.

(Chung, Sheng-Luen ; Lafortune, S. 1992) - "Limited Lookahead Policies in Supervisory Control of Discrete Event Systems"- IEEE Transactions on Automatic Control, vol. 37, n. 12, December 1992.

(Cohen, B.;Murphy,G.L. 1984) - "Models Of Concepts" - Cognitive Science 8 (1984) pp. 27-58

(Collins, A. ; Michalski, R. 1989) - "The Logic of Plausible Reasoning : A Core Theory" - Cognitive Science 13, 1-49 (1989).

(Edelman, G. M. ; Mountcastle, V. B. 1978) - “The Mindful Brain : Cortical Organization and the Group-Selective Theory of Higher Brain Function” - Cambridge : MIT, 1978

(Edelman, G. M. 1987)- “Neural Darwinism - The Theory of Neuronal Group Selection” - Basic Books, Inc, 1987

(Fabro, J.A. 1996) - “Neuronal Groups and Fuzzy Systems : An Autonomous Vehicle Application” MSc Thesis - DCA-FEE-UNICAMP, Fev. 1996 (in portuguese).

(Fabro, J.A. ; Gomide, F. 1996) – “Self Organizing Neurofuzzy Control of Complex Systems” – Applied Mathematics and Computer Science, vol. 6, n. 3, pp. 581-594.

(Fan, K.C.;Lui, P.C. 1994) - “Solving Find Path Problem in Mapped Environments Using Modified A* Algorithm” - IEEE Transactions on Systems, Man and Cybernetics - vol. 24, n. 9, September 1994.

(Gudwin, R.R.; Gomide, F.A.C. 1994a) - "Genetic Algorithms and Discrete Event Systems : An Application" - *Proceedings of The First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, 26 de Junho a 2 de Julho de 1994, Orlando, Florida, USA.

(Gudwin, R.R.; Gomide, F.A.C. 1994b) - "Context Adaptation in Fuzzy Processing" - *Proceedings of the Brazil-Japan Joint Symposium on Fuzzy Systems*, 19 a 22 de Julho de 1994, Campinas - SP - Brasil.

(Gudwin, R.R.; Gomide, F.A.C. 1997) – “Computational Semiotics : An Approach for the Study of Intelligent Systems – Part I : Foundations” – Technical Report RT-DCA-1997.

(Gudwin, R.R. 1996) - “Contribuições ao Estudo Matemático de Sistemas Inteligentes” - Tese de Doutorado - DCA-FEEC-UNICAMP, Maio 1996.

(Krozel, J.; Andrisani II, D. 1995) - “Intelligent ε-Optimal Path Prediction for Vehicular Travel”, IEEE Transactions on Systems, Man and Cybernetics, vol. 25, n.2 - February 1995.

(Lippman, R.P. 1987) - "An Introduction To Computing with Neural Nets" - IEEE ASSP Magazine April 1987

(Michalski, R.S. 1983) - “A Theory and Methodology of Inductive Learning” - in Michalski et.al. *Machine Learning - An Artificial Intelligence Approach* - Morgan Kaufmann Publishers, Inc. pp 83-134.

(Michalski, R.S. 1987) - “Learning Strategies and Automated Knowledge Acquisition - An Overview” in Bolc. R.- *Computational Models of Learning*, Springer Verlag, Berlin.

(Michalski, R.S.; Carbonnell, J.G.; Mitchell, T.M. 1983) - “Machine Learning - An Artificial Intelligence Approach” - Morgan Kaufmann Publishers, Inc.

(Michalski, R.S.; Carbonnell, J.G.; Mitchell, T.M. 1986) - “Machine Learning - An Artificial Intelligence Approach - Volume II” - Morgan Kaufmann Publishers, Inc.

(Oliveira, M.; Figueiredo, M.; Gomide F. 1994) - “A Neurofuzzy Approach for Autonomous Control” - 3rd International Conference on Fuzzy Logic, Neural Nets and Soft Computing, IIZUKA’94, 1994, Fukuoka, Japan

(Ram, A. ; Leake, D. 1991) - “Evaluation of Explanatory Hypotheses” - Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society, Chicago, IL, USA August, 1991.

(Rao, N.S. 1995) - “Robot Navigation in Unknown Generalized Polygonal Terrains Using Vision Sensors”, IEEE Transactions on Systems, Man and Cybernetics, vol. 25, n. 6, June, 1995

(Shafer, G. 1976) - “A Mathematical Theory of Evidence” - Princeton, N.J. - Princeton University Press, 1976.

(Spence, R.; Hutchinson, S. 1995) - “An Integrated Architecture for Robot Motion Planning and Control in the Presence of Obstacles with Unknown Trajectories” - IEEE Transactions on Systems, Man and Cybernetics, vol 25, n. 1 - January 1995.

(Verschure, P.F.M.J.; Kröse, B.J.A.; Pfeifer, R. 1992) - “Distributed Adaptive Control : The Self-organization of Structured Behavior” - Robotics and Autonomous Systems 9 (1992) 181-196.

(Verschure, P. 1993) - "Formal Minds and Biological Brains - AI and Edelman's Extended Theory of Neuronal Group Selection" - IEEE Expert, October 1993, pp. 66-75

(Wang, F.; Kyriakopoulos, K.; Tsolkas, T.; Saridis, G.N. 1991)- “A Petri Net Coordination Model of Intelligent Mobile Robots” - IEEE Transactions on Systems, Man and Cybernetics - vol. 2, n. 4, July/August 1991.

(Yen, J.; Pfluger, N. 1995) - “A Fuzzy Logic Based Extension to Payton and Rosenblatt’s Command Fusion Method for Mobile Robot Navigation”, IEEE Transactions on Systems, Man and Cybernetics, vol. 25, n. 6, June, 1995.

(Zurada, J.; Marks II, R.J.; Robinson, C.J. 1994) - “Computational Intelligence - Imitating Life” - IEEE Press.