Fielded Object Networks and Semiotic Engineering Techniques in Intelligent Systems

Rodrigo Gonçalves
rodrigo@dca.fee.unicamp.br

Ricardo Gudwin
gudwin@dca.fee.unicamp.br

Department of Computer Engineering and Industrial Automation - DCA - Faculty of Electrical and Computer Engineering – FEEC - State University of Campinas - UNICAMP – Brazil

ABSTRACT

This work introduces a first proposal on how to use semiotics in order to improve software engineering methods, when intelligent autonomous systems are targeted. First we investigate the current flaws in software engineering, concerning intelligent autonomous systems. Then we propose a knowledge taxonomy, based on semiotic ideas, aiming a tool to understand the information domain of intelligent autonomous systems. Further, we illustrate on how to use the notion of knowledge types during the development of a simple intelligent autonomous system, emphasizing the relationship between the described types of knowledge aiming at the understanding on how they are organized into an intelligent autonomous system. After that we review the idea of generalized subsistence machine (GSM) proposed by Meystel as a possible tool (not the unique nor the best one) to represent the information domain of intelligent autonomous systems. Further, it is shown how both ideas might be used in the requirement analysis step of any software engineering method when an intelligent autonomous system is targeted. Finally, the design and implementation steps are discussed. As a conclusion, we discuss the future works and trends of semiotic intelligent autonomous systems engineering.

KEYWORDS: *artificial intelligence, semiotics, artificial life, autonomous systems, software engineering, UML, knowledge unit, knowledge space, knowledge taxonomy*.

1 INTRODUCTION

During the early years of the computer era (1950 to around of 1965), software development was an art that was virtually unmanageable. There was very few systematic software development methods and they used to be very specific and with a limited distribution. Software design, by this time, was an implicit process of one's head, and documentation was often nonexistent [Pre92]. As soon as general-purpose hardware became a commonplace and multiprogramming and multi-user systems were introduced, the second computer era had started (around mid-60's to the late 1970). Many other new and powerful concepts, like real-time computing and database, had been introduced. The complexity of the applications and the large scale in which they started to be sold introduced a dark cloud on the horizon: the software maintenance. This fact fired the software crisis that started the third computer era (late 70's to late 80's). This era introduced many software engineering paradigms and tools. Now, with the appearance of objected-oriented paradigm, powerful desktop computers, Internet, multimedia and artificial intelligence (AI), we are living the fourth computer era.

Since the beginning of the computer science artificial intelligence was an aspiration. In fact, the wish for intelligent machines came before the idea of computers, as they are today (an approximation to Turing machines). AI techniques and methods have been emerging and getting better since early 60's when the mathematical logic was first applied to achieve artificial intelligence and an AI sub-field has emerged: the autonomous intelligent systems. This field emphasized the use of AI techniques applied to autonomous systems, i.e., systems capable of taking decisions in an autonomous way.

Nowadays we have many powerful theories, methods and paradigms related to AI but, unfortunately, we still lack of systematic AI systems development methods able to provide a formal way to fully understand the knowledge processing and communication inside them. Additionally, AI researchers and engineers have another problem: current software representation models are often inadequate for autonomous systems. Due to the fact that, for most of autonomous systems, it is not possible to predetermine *a priori* all possible states, modeling tools based on finite state automata theory, e.g. state charts and Petri nets, become inadequate. This problem is especially important in the development of evolutive autonomous systems. In this case not only the representation models are inadequate. Analysis and design paradigms, e.g. objected oriented, lack concepts that allows dynamic data structures definition as well as a way to formally deal with evolutive functions and methods (e.g. genetic programming).

This paper introduces some ideas based on semiotics in expectation to help overcoming some of those problems. But, why semiotics? Why should software/systems engineers concern about semiotics? The answer is that semiotics provides a new analysis domain for AI problems, in a way comparable to when frequency analysis was first introduced in the domain of sign processing: a new perspective for analysis of the same problems.

The ideas introduced in this paper are a preliminary study on this direction, aiming at achieving new insight for the AI analysis

domain. The approach introduced in this paper is a first step toward an autonomous system engineering method based on semiotics and software engineering.

The approach introduced in this paper is a first step toward an autonomous system engineering method based on semiotics and software engineering.

2 AIMING A GOOD INFORMATION DOMAIN MODEL

Computational systems are dynamic systems that can be modeled by a Turing Machine. But for the pragmatic task of programming computers, we don't use artifacts given by Turing Machine theory. Humans have developed high level languages and engineering methods to make the development of complex systems possible and affordable within computers. Software engineering methods can be divided into three parts: Analysis, Design and Construction (implementation). The analysis step intends a good problem understanding for consistent following steps. The quality, maintenance and expandability of the software are grounded in this step. Anywise, this step can also be divided into two other ones: Requirement Analysis and Domain Analysis. The requirement analysis aims at understanding the problem scope and the domain analysis aims for the identification and partition of software roles and tasks. To help us understand, let's consider a reference model [Lar98]: a business organization. If our goal is to design a software for the management of a business organization, during the requirement analysis we should ask ourselves: "What are the business concepts and processes (e.g. making sales, paying employees, etc)?" In the same way, during domain analysis we would ask: "What are the employee roles?" The domain analysis builds a conceptual model that is not a description of software components; it represents concepts in the real-world problem domain, focusing in data processing aspects (information flow, structure and content). This problem domain with this focus is called "information domain". This work introduces a semiotic based strategy to analyze the information domain of autonomous intelligent systems aiming for a consistent analysis task. In other words, a tool to help in a consistent understanding and partitioning of the system specification targeting a computer implementation of autonomous intelligent system.

An autonomous intelligent system (IS) can also be seen as a set of concepts and processes aiming for an adequate information treatment and behavior generation. But, although we may use current software engineering methods to focus on the information domain, in this case they do not provide a formal way to understand the knowledge transforming and communication needed by any autonomous IS. In other words, in this case, current software engineering methods still lack of an adequate methodology for knowledge representation. The appropriate understanding of information domain is important to achieve a good allocation of information transforming methods into each system module. The lack of such methodology may result in systems that will be more complicate, will need more time to be developed and also get the risk of becoming unmanageable as they grow in complexity. The use of software engineering techniques is prescribed here as a way of getting those systems manageable and affordable. The lack of an adequate information structure may lead to difficulties in the use of multiresolutional approaches (what is sometimes vital concerning intelligent autonomous systems) and in the system scalability, understandability and maintenance. All those reasons encouraged us to propose new techniques that seem to be more adequate when intelligent autonomous systems are concerned. These techniques explore the semiotic nature of knowledge and its use in construction of intelligent autonomous systems. In next section we present a knowledge taxonomy based on C.S.Peirce's and Morris' semiotics and how it can be used in order to obtain a good information domain model.

3 A KNOWLEDGE TAXONOMY

Peirce's semiotics introduced a signical taxonomy, where different kinds of signs (e.g. rhemes, dicents, arguments, icons, indexes, symbols, qualisigns, sinsigns, legisigns) were proposed, addressing different characteristics of its structure and signic function. From Peirce's taxonomy, Gudwin [Gud96] derived a taxonomy of types of knowledge, where each type of knowledge addresses a different way in which phenomena from the world can be modeled. This taxonomy can be summarized in Figure 1 and will be explained in the following sections. In this figure R means *Rhematic*; D means *Dicent*; Ic means I*conic*; Ob means O*bject*; Sp means *Specific*; G means *generic*; Sy means S*ymbolic*; In means *Indexical*; Se means *Sensorial*; Oc means *Occurrence*.

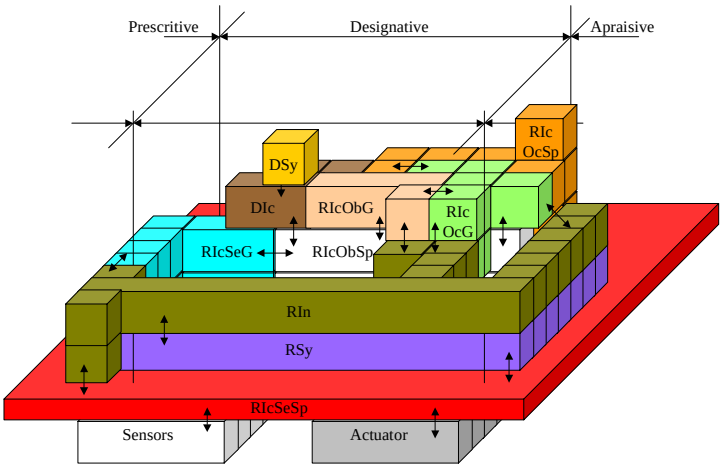

Figure 1 - Knowledge taxonomy

All arrows refer to argumentative knowledge (both analytic and synthetic). From this point forward, we will use the Figure 1 notation between brackets to specify the different kinds of knowledge. For example, {RIcSeG} means a rhematic iconic sensorial generic knowledge.

In this taxonomy a knowledge is first classified according to its functionality (designative, apraisive, prescriptive) and to its

structure (rhematic, dicent). As we will see, argumentative knowledge is a special case and it is both a functional and a structural. Here we will present the structural classification before discussing about the functional one.

3.1 RHEMATIC KNOWLEDGE

A rhematic ({R}) knowledge is the semantic that can be assigned to isolated words in a natural language. Usually, it is associated to a representation of environmental phenomena like sensorial experiences, objects and events. Sensorial experiences can be represented by adjectives, objects by substantives and events by verbs. In a last analysis, though, all of them are connected to perceptual data. The rhematic knowledge can be represented an icon (rhematic iconic - {RIc}) or an index (rhematic indexical - {RIn}). A {RIc} knowledge is a direct representation of the phenomenon that it represents. A {RSy} knowledge is a name that refers the phenomenon. A {RIn} knowledge is an indirect reference to the phenomenon.

The {RIc} knowledge can be divided into three different classes: sensorial - {RIcSe}, usually modeled by adjectives; object - {RIcOb}, usually modeled by substantives; occurrence - {RIcOc}, usually modeled by verbs. Those three knowledges can be divided into two classes: specific (Sp) and generic (G). For example, a {RIcSe} knowledge might be a sensorial information like an image or a temperature sensor output. A {RIcSeSp} knowledge might be a particular instance of a sensorial pattern, e.g. a specific image, or the temperatures in a given time. A {RIcSeG} knowledge might be a generic knowledge of all time occurrences of some sensorial input. The knowledge that the outside temperature is 28 degrees Celsius is a specific sensorial knowledge but the knowledge of what is a high temperature is a generic sensorial knowledge.

The {RIcOb} knowledge is the knowledge related to a real world object (existent or nonexistent). The {RIcObSp} knowledge is the knowledge of a specific occurrence of a specific object. It assumes an existence of an object model. This model is a {RIcObG} knowledge.

The {RIcOc} knowledge corresponds to the semantic of verbs. Usually, it associates an attribute to an object, or model the changes in one or more attributes in the object (or objects) as an effect of time. e.g. a property of an object, it's creation or destruction, etc. The {RIcOcSp} knowledge is related to a specific occurrence in time, e.g. the knowledge that a traffic light, in a specific time changed from red to green. The {RIcOcG} knowledge is related to a generic occurrence, e.g., the change of a traffic light from red to green, without specifying a particular instance.

3.2 DICENT KNOWLEDGE

The dicent ({D}) knowledge is a proposition. The difference of a proposition and a term is that the proposition has a truth-value associated with it. Usually, this truth-value represents the belief of the proposition and it can vary from false to true using a multivalored logic or not (e.g. fuzzy logic). Compared to rhematic knowledge, we would say that, in a natural language, if rhematic knowledge represents the meaning of a single word, dicent knowledge represents the semantic of phrases.

A proposition is formed to the association of a term (or a set of terms) to a truth value. Propositions can also be formed by the association of more primitive propositions, linked by logical connections. Examples: the knowledge that "A" is true; the knowledge that "A∧B" is false; the knowledge that "IF A∧B THEN C".

A dicent knowledge can be iconic ({DIc}) or symbolic ({DSy}). The {DSy} knowledge is a name for a whole phrase, and consequently, has attached a truth value, e.g. a label for a first order logic sentence. The {Dic}knowledge turns explicit its composing rhematic pieces of knowledge, associating to them a measure of the belief of this phenomenon (occurence).

3.3 DESIGNATIVE KNOWLEDGE

Until now we presented the structural classification of the knowledge. Now we start discussing the functional one. The first functional category is the designative knowledge. Designative pieces of knowledge are employed to represent the world. An alive organism usually initiates its life with almost none designative knowledge.

A designative knowledge can be at the same time rhematic and designative as depicted in Figure 1.

3.4 APRAISIVE KNOWLEDGE

Apraisive knowledge is used to make a judgment according to some objective. In an alive system this objective can be innate or learned. In this case the apraisive knowledge can be, for example, love, hate, pain, pleasure, desire, etc… An apraisive knowledge is usually classified as rhematic iconic.

3.5 PRESCRIPTIVE KNOWLEDGE

Prescriptive knowledge is used to plan, predict and actuate in the real world through actuators. In the same manner of apraisive knowledge the prescriptive knowledge is usually classified as rhematic iconic.

3.6 ARGUMENTATIVE KNOWLEDGE

Before introducing the argumentative knowledge it is necessary to introduce the concepts of knowledge unit and knowledge space. A knowledge unit is an atomic structure of information carrying some meaning, i.e. modeling at a particular level of resolution, a phenomenon from the world. A knowledge space is a space that stores knowledge units. In this context, only the structural knowledge taxonomy is relevant because the concepts of knowledge unit and space are related only to knowledge's structure, not to its functional purpose.

Argumentative knowledge is related to knowledge processing and transformation. It may be seen as an algorithm that creates or transforms knowledge units within a knowledge space. It is both a structural and functional classification. It is structural in the sense that it is a knowledge unit like any rhematic or dicent knowledge but its structure holds a program code instead of data. It is functional because it has an explicit functional role. A good metaphor for understanding what an argumentative knowledge is, the representation of machine instructions within a computer memory. Those instructions can be seen both as data (a sequence of bytes) and code (processor instructions). In the same way, argumentative knowledge is both data (structure) and code (process).

The argumentative knowledge ({Ar}) can be synthetic ({ArSt}) or analytic ({ArAn}). An {ArAn} knowledge unit is like a piece of code that creates new knowledge units that does not contain any new information - it only turns explicit an information that was implicit in the knowledge space. In other words, it only performs an "analysis" of existing knowledge units, making explicit something that was modeled into a more compact form. The most important {ArAn} is the "modus ponens". Different from the {ArAn}, the {ArSt} knowledge creates knowledge units that contain new information. In other words, it synthesizes new knowledge content. There are two kinds of {ArSt}: inductive ({ArStId}) and abductive ({ArStAb}). The {ArStId} knowledge makes small modifications in the premises knowledge units to produce a new one. In this sense, one can say that it is a constructive argument. One example of {ArStId} is the generalization. The {ArStAb} selects candidate units as true or false (at any degree) according to the preexistent knowledge units in the knowledge space. The candidate units can be generated either by an {ArStId} knowledge or by any random method. Different of the {ArStId} knowledge, it is a destructive argument.

4 AN INFORMATION DOMAIN PARTITION

Any software engineering method has three well defined phases: analysis, design and implementation. Now we will address the analysis phase. The other ones will be discussed in the next section.

During the requirement analysis step of any software engineering method, our goal is to outline the scope of the system. We do this by finding its restrictions, objectives, inputs and outputs. After that, in the domain analysis, an information domain model is built focusing on "what" not on "how" [Pre91]. In the semiotic point of view, the model should make clear the classification and distribution of all knowledge units of the system except the argumentative ones.

The knowledge taxonomy previously presented can be used to help us understand the information domain of the system. This approach may help us achieve better distributed and more representative models. Classifying the inputs, outputs, objectives and restrictions is the first step to achieve a good information domain model. As we said before, this classification leads to a better understanding of the information domain. At this step the focus should be on the information structure, not on the information functionality. Aiming a better understanding on how the information can be classified, lets consider the Terzopoulos' artificial fishes (Figure 2) as a case of study [Ter94]. This example is very adequate because it is not too complex, it is ludic and it was not originally designed with the help of semiotics.

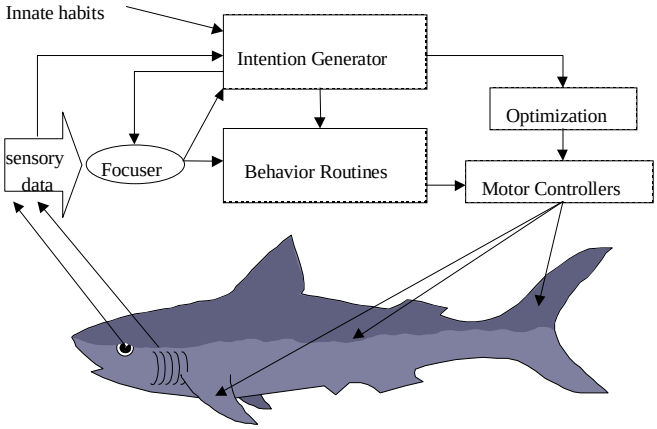

Figure 2 - Artificial fish model

The artificial fishes have innate knowledge units that determines whether or not it is male or female, predator or prey, if it likes schooling or not, if it likes brightness or darkness, cold or warmth, and so forth. Using the knowledge taxonomy we can say that these knowledge units are apraisive, more specifically {RIc}. Other apraisive knowledge units present in the artificial fishes model are the mental state variables: hunger (H), libido (L) and fear (F). Note that the mechanism used to calculate them is due to argumentative knowledge units. The behavior of an artificial fish is generated using intention generation and behavior generation modules based on predefined rules. Those rules correspond to prescriptive knowledge units ({RIc}). The artificial fishes learn how to coordinate their actuator to produce a coherent movement. This knowledge is prescriptive too.

Once understood the information structure (from the semiotics point of view), it is possible to build an information domain model, just by finding the mapping between the information domain and any previously known AI architecture. We can say that this architecture plays a role of a pattern [Lar98] [Szy98] and, in this case, the model is the result of the process of applying it over our problem. To do that, both the pattern and the problem must be understood from the semiotics point of view.

It is important to note that any AI architecture can be used, in fact we are working in our own, but to illustrate the idea, we will recall to a well known architecture: the general subsistence machine (GSM). Meystel [Mey96] has proposed the GSM architecture as "*a system which is unified by a goal to exist as an entity. In pursuit of this goal, GSM can perform tasks which has been developed internally or submitted externally*". The GSM architecture is very

adequate to autonomous systems modeling but it would be naive to imagine that it is the only or the best architecture. In this work, the GSM structure is adopted due to the fact that it is generic and promising. Any other model can be used instead of the GSM. For that, it is only necessary to understand the functionality and information distribution of such model in the semiotic point of view. The GSM architecture (in the nested format) can be seen in the Figure 3.

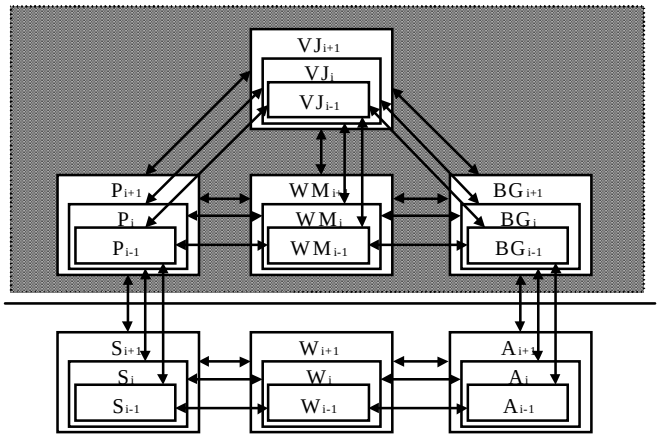

Figure 3 - Multiresolutional GSM structure

The GSM modules are perception (P), world model (WM), value judgement (VJ), behavior generation (BG), "S" means sensors, "W" means world and "A" means actuators. Any of the GSM modules can be regarded as GSM too (embedded GSM).

In a first glance, this definition seems to be very adequate for autonomous agents and the like but not for others IS as, for example, medical diagnosis expert systems. It is not true and it is easy to see that GSM structure bears for all intelligent systems. In a medical diagnosis system for example, its "goal to exist as an entity" is related to its success in performing good diagnostics. Its actuators might be a GUI that make queries for the user and the sensors might be a GUI that receives his answers.

Based on the fact that the GSM maps intelligent systems, we can use its structure as a starting point for a domain analysis model. This approach leads to a multiresolutional representation of the IS that might be very interesting in most cases but it's necessary to certificate that the IS mapping into the GSM will be adequately done. To make it possible it is necessary to know how to distribute our system elements into the GSM parts, otherwise the system model might turn to be confusing and unmanageable.

To ensure a properly autonomous IS information domain mapping into a GSM structure, two points should be observed: the function and objective of each GSM component and which kind of knowledge each of them supports. The first point should be carefully observed to avoid mistakes. When the engineer is working over a GSM-based IS information domain model he must remember that any GSM part can be a nested GSM. In a nested GSM the meaning of each GSM part changes according to its location. For example: a perception module is responsible for a sensory processing. So, a world model of a GSM nested in a perception module may contain a model of the sensorial space and the actuators may be something like a focus system. For a complete description of each GSM part please refer to [Mey96].

The second point to be observed turns clear the importance of semiotics in the information domain analysis. GSM parts usually hold the same kind of knowledge, no matter if it is embedded or not. Thus, a prior knowledge of the knowledge distribution in the GSM structure helps us to build consistent information domain models. Table 1shows the taxonomy of the knowledge contained by each GSM part. The notation "X $\rightarrow$ Y" means an argumentative knowledge that transforms a knowledge type X to a knowledge type Y.

Let's recall the artificial fishes example again. In this case we can see that both the intention generator, behavior generator and optimization modules hold prescriptive and designative knowledge units. This suggests that they can be combined into one single behavior generation module in different hierarchical levels (nested GSM). The focuser acts into the sensorial space. It uses some prescriptive knowledge to change the designative knowledge generation. This fact suggests that the focuser is part of an embedded GSM in the perception module. The apraisive and some designative knowledge used by the intention generator to control the focuser can be grouped into a world model module, completing a GSM (Figure 4). The model obtained using the semiotics analysis structurally differs from the original model depicted in the Figure 2 but it is functionally equivalent. You might wonder why the second model is better if both of them are functionally equivalents. The main reason is that the first model may not exist, in other words, creating a model like the one depicted in the Figure 2 for autonomous systems is not a trivial task – that's why people study architectures for autonomous systems. Using a generic architecture for autonomous intelligent systems as a start point is a good idea. The semiotic and more specifically the proposed knowledge taxonomy help us to find out how a specific system maps in a generic architecture like the GSM. Again it is important to note that the GSM is a possible architecture, not the architecture. Any other autonomous intelligent systems architectures can be used once fully understood the information exchange and processing in the semiotics point of view.

Another important point is that the resulted model should be built in conformance with the analysis, design and implementation paradigm adopted. For example, if the analysis method is the *structured analysis* the GSM structure should be represented as a data flow diagram (DFD). In this case, many DFDs in different detail levels can represent the GSM. In an *object-oriented analysis* (OOA) the GSM structure can be easily mapped into an object-oriented structure. We will introduce in future works how this method integrates with a specific software engineering method called UML (Unified Modeling Language) [Lar98]

Knowledge	P	VJ	WM	BG
Apraisive		X		
Designative	X		X	
Prescriptive				X
{RIcSe}	X	X	X	X
{RIcO}	X		X	
{RIcOc}	X		X	
{D}			X	X
{RIcSeSp} → {RIcSeG}	X		X	
{RIcSeSp} → {RIcOSp}			X	
{RIcSeG} → {RIcOSp}			X	
{RIcSeG} → {RIcOG}			X	
{RIcOSp} → {RIcOG}			X	
{RIcSeSp} → {RIcOcSp}			X	
{RIcSeG} → {RIcOcG}			X	
{RicOcSp} → {RIcOcG}			X	
any {RIc} → {RIcSeSp}				X

Table 1

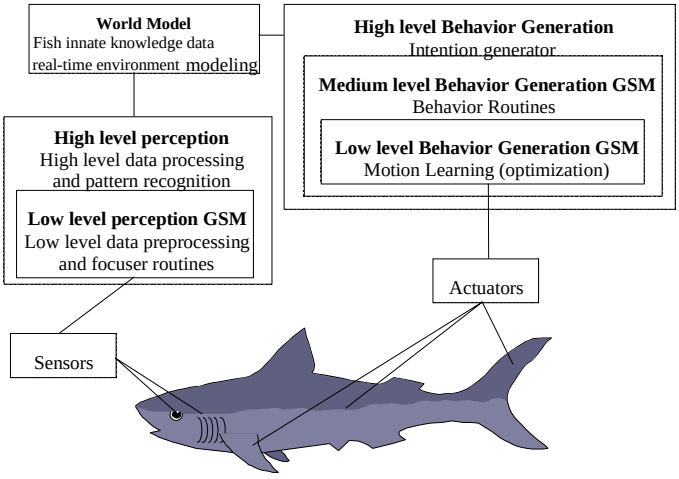

Figure 4 - GSM based artificial fishes

5 A KEY TO IMPLEMENTATION

After built a domain model for our problem, we go forward to the next steps of the software engineering process: design and implementation. To find the domain model of the system, during the analysis step, we used the idea of *patterns* supported by the semiotics analysis. An analog process can be done into the design and implementation step using the idea of framework [Lar98] [Szy98]. The main principle is quite simple: if we know how a GSM (or any other pattern) can be implemented using any framework, we know how to implement anything modeled as a GSM. Here we propose a computational tool to act as a framework in the design and implementation steps. This tool is called Fielded Object Network (FON), and is based on the Object Network (ON) first proposed by Gudwin as a knowledge representation tool [Gud96] [Gud97].

5.1 FIELDED OBJECT NETWORK

The main concept of *Object Networks* and *Fielded Object Networks* relies in the *object* definition. An object represents a logical or physical entity in a certain level of abstraction. The *Fielded Object Network* shares the same theoretical basis with Object Network, so due to space limitations, these basic concepts will not be described in detail here. The interested reader is referred to a complementary reading [Gud96] or [Gud97].

Considering the *Fielded Object Networks*, an object is an individual and atomic entity composed by five elements: *attributes*, *functions*, *field*, *connections* and *life cycle*.

5.1.1 Attributes and functions

Attributes are those predicates that describe the objects. For example, a color might be an attribute of an object that represents a space ship. On the other hand, an object that represents a window is not an attribute of the space ship object, but one of its parts. It is important to note that *attributes* and *parts* are different concepts. In the Fielded Object Network scope, if an object must describe one of its parts, this part will be also an object, and this object is stored into the former object's *field* (see section 5.1.2). The values of attributes in a particular instant of time constitutes an object's state.

Functions (sometimes called *methods*) are another important concept in all object-oriented approaches. In the Fielded Object Network a function is represented by an algorithm that returns some information and changes the state of the object (based in its own state and in the function parameters).

5.1.2 Field

Fielded Object Network enhances the traditional concept of object, used by the most popular object oriented analysis (OOA) and design (OOD) methods like UML, OMT or Booch [Lar98] by introducing three new concepts associated to an object: *field*, *connections* and *life cycle*.

A *field* is considered as a place where the object places the objects that it contains, like a box or a container. The *field* might have any kind of spatial structure and dimension. It can be continuos or discrete, ordered or unordered, finite or infinite.

The *field* might impose rules that affect a subset of attributes of the contained objects. We call those attributes *physical parameters* and the rules that change them *physical laws*. The field might impose physical laws *or not*. If it does, any object into the field must have physical properties compatible with the field structure and its physical laws. It is important to note that in the same way that the field space might have any dimension and ordering scheme, the physical laws can be any kind of rule. For example, if a field is a cartesian 3D-continuous space, the physical laws might be similar to gravitational or electromagnetic fields and forces. If a field is a 2D discrete space, the physical laws might be simple

rules between neighbor objects. In this last case, the field predisposes its contained objects to work as a cellular automata. Figure 5 depicts an example of this concept.

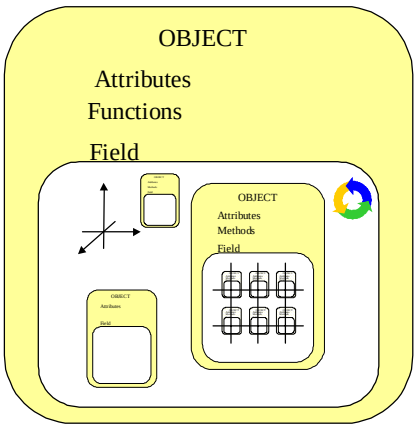

Figure 5 – Field and Embedded Nature

The physical laws are processed into an independent dynamical system called *embedded nature*. We can think the *embedded nature* as an independent process stream (or thread). In fact, in a digital computer, it might be implemented in this way.

If an object has a *field* but does not have an explicit *embedded nature*, we consider this dynamical system to be static. As we will see, an object has many independent asynchronous processing streams. When the *embedded nature* detects a physical (happening within the field) interaction between two or more objects (like a collision), it might warn all objects using a special asynchronous function called *hit function* (see Figure 6). In this case a *communication connection* (see section 5.1.3) may be opened. Later in this paper, we will explain all possible interactions between objects and the synchronization scheme used to make them possible in a digital computer. As we can see, this approach introduces a non-deterministic factor that might be very important on intelligent systems engineering.

In a Fielded Object Network, there is only one object that is not into any field. This object is called *universe object*, being the source of containership of all other objects. In the other hand, objects that do not contain any objects in their field are called atomic objects. The object that contains another one is said to be its *container object*.

5.1.3 Connections

An object has direct access to its attributes and functions, as well as its contained objects attributes and functions. In both cases the communication between those objects are simple and reliable. As we have already seen in the last section, the Fielded Object Network is topologically ordered. This fact imposes some topological restrictions in all communication cases where this access is not due to a containment relationship. Thus, it is not possible for an object to communicate with another one out of its field without a *connection*. A *connection* is a communication line between two or more objects, wherever they are.

It is important to note that objects can interact in a *logical* or *physical* fashion. The embedded nature and the *hit function* (as explained in the last section) perform the *physical* interaction between objects. A process called *assimilation* is responsible by the *logical interaction* between objects. As we will see, there are three different assimilation types. In all of them, an object might assimilate another one if and only if there is a *connection* between theirs container's field. In this sense, we consider that if an object is within a field, there is a natural connection from it and its container. Figure 7 shows an example. The connections are graphically represented as links between objects. The connection might be directed or undirected.

As we will see in the next section, all objects, in the worst case, have at least two independent processing streams (*embedded nature* and *life cycle*). When implementing a FON into a digital computer, this fact imposes that some care should be taken to ensure its data and structural consistency. All communication mechanisms are defined as atomic operations and are protected by binary semaphores to properly deal with race conditions and to avoid deadlocks. Due to practical reasons, to allow this safe communication mechanism in the FON, an additional restriction is applied: an object must explicitly allow its assimilation before

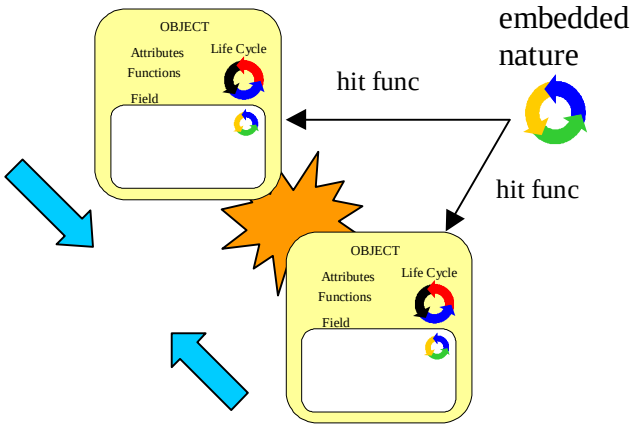

Figure 7 – Physical communication by means of the hit function

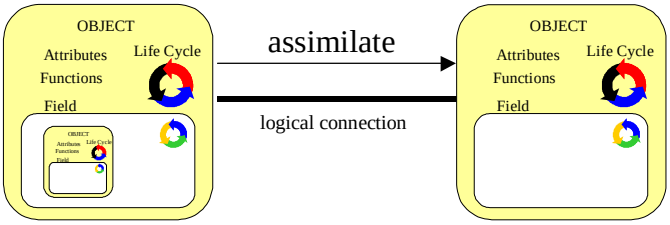

Figure 6 – Logical communication by means of a connection

being assimilated and it should specify the allowed assimilation type.

There are three different assimilation types. The first one is a *destructive assimilation* (Figure 8). In this case, the object is moved from the source field to the destination one. The second assimilation type is the *non-destructive* (Figure 9) where the object is cloned (copied) in the destination field. The latter and more complex assimilation mechanism is the *message passing* (Figure 10). In the message passing mechanism the assimilation is performed with a help of a third object called *messenger* that goes to the source field, physically attach itself to the assimilated object, modify its state based on assimilated object state and then return to the original field. This is necessary when we do not want to get a whole object, but only part of its structure.

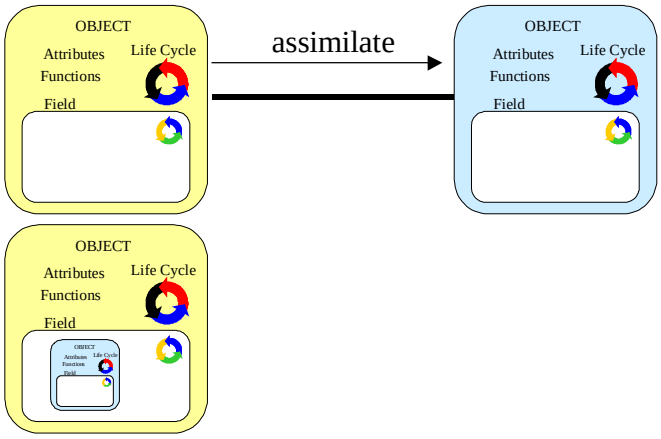

Figure 8 - Destructive Assimilation

5.1.4 Life cycle

The last and maybe the most important FON object characteristic is the concept of *life cycle*. A *life cycle* is an independent and parallel dynamic system that is ruled by some kind of algorithm

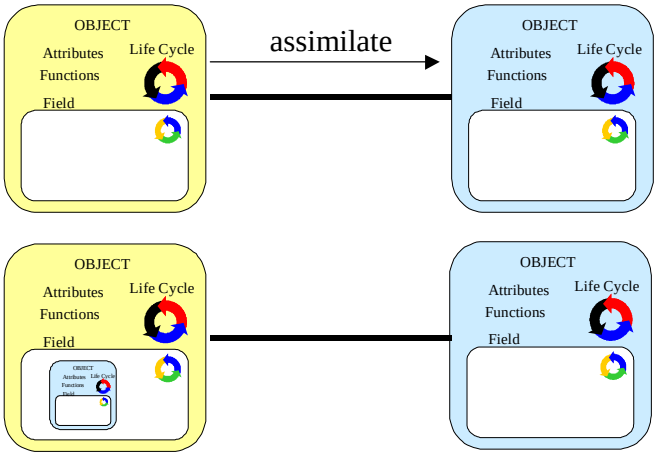

Figure 9 - Non-destructive Assimilation

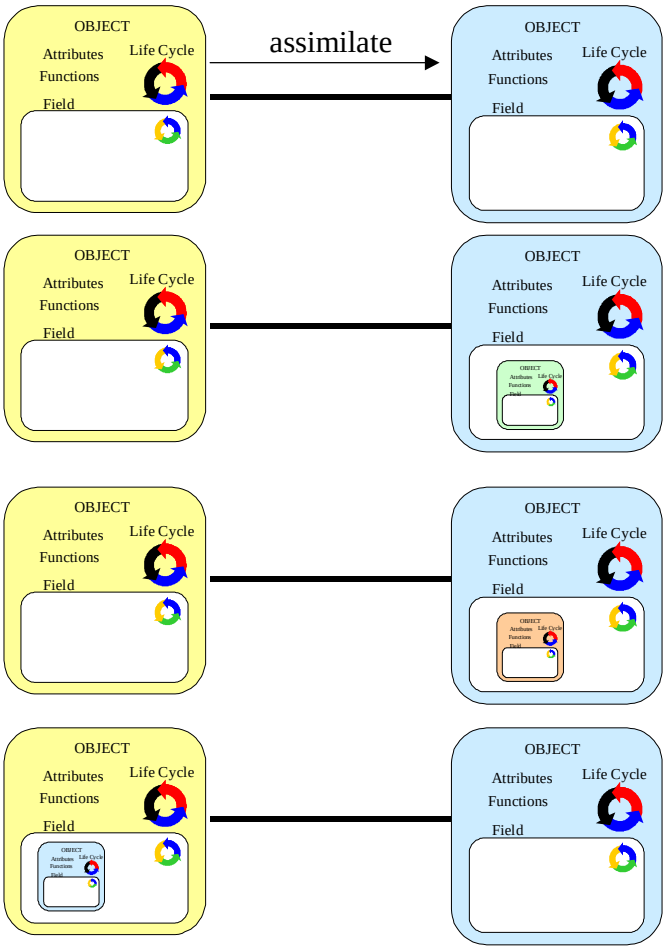

Figure 10 - Message Passing

while the object exists. If the *life cycle* does nothing, or does not exist, the object is called *passive*, otherwise it is called *active*. A *life cycle* can be assigned dynamically to an object. In this sense, an object can initially be passive and become active, or be active and suddenly become passive. We say, when we assign a *life cycle* to an object, that we are giving life to it.

5.1.5 Why and when Fielded Object Network?

Distributed Artificial Intelligence (DAI) corresponds to the intersection of Distributed Computing and Artificial Intelligence. In Distributed Computing, several processors share data, but not control. It focuses on low-level parallelization and synchronization issues [Sto97]. In DAI the intelligent control as well as data is distributed and it focuses on problem solving, communication and coordination. Recently, people tend to break DAI into two classes: Distributed Problem Solving (DPS) and Multiagent Systems (MAS) [Sto97] [Wei96], despite that the boundaries between them is still not clear. In DPS a complex task is decomposed and distributed. A solution is synthesized by collecting all subtasks partial results. DPS focuses on information domain management. Usually in DPS domain, strong assumptions are performed to

ensure compatibility of different problem-solving entities. Those restrictions normally do not apply on MAS systems.

As we already said, an object has, in the worst case, two independent processing streams: the *embedded nature* and the *life cycle*. In a digital computer implementation, those processing streams might be independent and the connection communication mechanism can be implemented aiming a distributed communication (e.g. using TCP/IP, CORBA or PVM). It makes the FON a Distributed Computing tool oriented for Distributed Artificial Intelligence engineering, more specifically, for Multiagent Systems. This paper will focus on how the Generalized Subsistence Machine (GSM) [Mey96] can be mapped using the FON architecture aiming a DAI architecture grounding. Another interesting research theme associated to FON is Artificial Life. This is to be addressed in a future paper

6 A FON APPROACH TO GSM

Taking the advantage of FON parallelism and scalability a GSM can be implemented as shown in Figure 11.

Sometimes the Value Judgement (VJ) element (object) can be incorporated into Behavior Generation and World Model elements (objects). It is important to note that using the FON approach the multiresolutional characteristic of the GSM structure is improved by another one: the parallelism. One of the GSM claims is that the world is hierarchical, thus, a hierarchical modelling would achieve better results. Following this thought, we claim that the world is massively parallel, thus, a parallel approach is important too (this is one of the DAI claims) but it would be naive to think that a multiresolutional and parallel approach is adequate for all cases. This fact turns evident one of the most important FON advantages: the objected-oriented encapsulation concept. Using this concept an MAS can be conceived with GSM and non-GSM agents together. More than that, an agent can be constructed with GSM and non-GSM modules that work together in a cooperative or competitive fashion and each one can use different knowledge representation and processing approaches.

7 CONCLUSION

This work introduced important issues concerning the development of a semiotic oriented software engineering methodology, aimed for the analysis and design of autonomous intelligent systems. First of all, we emphasized the utility of discriminating among different types of knowledge, when considering the domain of intelligent autonomous systems as a development platform. The hierarchy developed by Gudwin is a good starting point, in this case, that allows us to discriminate on the different functions performed within an autonomous intelligent system and also point out concepts of different nature that need to be treated in these systems. Differently from common software systems, intelligent autonomous systems concepts and processes

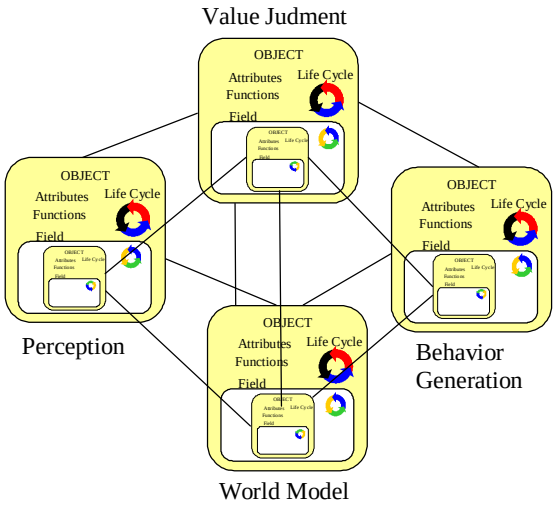

Figure 11 - GSM as a FON

are directly related to knowledge and knowledge processing. At this point, the introduction of the concepts of knowledge unit, in its structural and functional forms proves to be very useful during the targeted analysis development phase. In future works we will address the relevance of this concept in others development phases.

The definition of knowledge space allows the determination of an architecture suitable to handle the particularities of such class of software systems. The semiotic nature of different types of knowledge is the guideline on the whole process. Understanding the subtleties among the different knowledge types, and the knowledge types required in order to build an intelligent autonomous system, allows a better comprehension of its behavior, and also makes more easy its development because it allows an easier application of known autonomous intelligent systems architectures.

Most of the work in autonomous intelligent systems architectures are top-down in the sense that they first propose a model, showing their capabilities and characteristics and then try to apply these models in real problems. At this point they have problems because the low level, in other words, the real problems specifications, often have excess of information and it is difficult to understand the mapping from the low-level to the high-level (from the real problem specification to the autonomous intelligent systems architecture). The work presented in this paper is a first step toward a methodology to allow the usage of autonomous intelligent systems architectures in more reliable way.

This is a work in progress. Next publications should include a more in depth discussion concerning the concept of knowledge space and knowledge unit and address others software engineering phases as development and construction. Some work regarding a development the development phase is in progress. As a preliminary results we can point out the development of the Object

Networks (ON) [Gud95] and the Fielded Object Networks (FON) [Gon98]. They are both frameworks to implement intelligent systems and will be used in a further development phase development. In this paper we presented the main concepts of the Fielded Object Networks. The FON intends to be a tool in an autonomous intelligent systems engineering methodology. This methodology is still a work in progress. This approach is akin to the most recent developments in the literature of intelligent systems, and shows a promising methodology to modeling, analysis and design of such class of systems. At the same time, the FON object-oriented flavor also suggests a direct methodology for the computational implementation of these systems and the development of fourth generation software tools for autonomous intelligent systems.

As we said before, this work is the first step towards a semiotic engineering method and does not adopt any specific software engineering methodology. Future works should introduce how these concepts apply in a specific method such UML.

8 REFERENCES

[Alb91] - Albus, J.S., *Outline for a Theory of Intelligence*, IEEE Transactions on System Man and Cybernetics, Vol. 21, No. 3, May/June 1991.

[Col96] - Coleman D., *Object-oriented development: the fusion method*, Prentice-Hall Inc, New Jersey, 1996.

[Gon98] – Gonçalves, R., Gudwin, R., Gomide, F., *Fielded Object Networks as a Framework for Computer Intelligence*. Proceedings of the ISAS'98 – Inteligent Systems and Semiotics: A Learning Perspective –Gaithersburg, MD, USA – Semptember, 1998.

[Gud96] – Gudwin, R., *Contribuições ao Estudo Matemático de Sistemas Intelifentes – PhD Thesis – DCA-FEEC-UNICAMP, May 1996 (in portuguese).*

[Gud97] – Gudwin, R. R., Gomide, F., *An Approach to Computational Semiotics*. Proceedings of the ISAS'97 – Inteligent Systems and Semiotics: A Learning Perspective –Gaithersburg, MD, USA – 22-25 Semptember, 1997.

[Lar98] – Larman, C., *Applying UML and Patterns: an introduction to object-oriented analysis and design*, Prentice Hall, 1998

[Mey96] – Meystel, A. M., *Intelligent Systems: A Semiotic Perspective*, International Journal of Intelligent Control and Systems, Vol. 1, No. 1, pp. 31-57, 1996.

[Pei60] – Peirce, C. S., *Collected Papers of Charles Sanders Peirce*. Edited by Charles Hartshorne and Paul Weiss – Belknap Press of Harvard University Press – Cambridge, Massachussets, 2nd printing, 1960.

[Pre92] – Pressman, R. S., *Software Engineering a Practitioner's Approach – third edition*, McGraw Hill, 1992

[Szy98] – Szyperski, C., Component Software: Beyond Object-Oriented Programming, Addison-Wesley,1998

[Sta97] – Starks, S.A., Kreinovich, V., Meystel A., *Multi-Resolution Data Processing: It is Necessary, It is Possible, It is Fundamental*. Proceedings of the ISAS'97 – Inteligent Systems and Semiotics: A Learning Perspective - Gaithersburg, MD, USA – 22-25 September, 1997.

[Sto97] – Stone, P., Veloso, M., *Multiagent Systems: A Survey from a Machine Learning Perspective*. Carnegie Mellon University CS technical report number CMU-CS-97-193.

[Ter94] – Terzopoulos D., *Artificial Fishes*, Artificial Life, Volume 1, Number 4, MIT Press, 1994

[Wei96] – Weiβ, G., Sen, S., Adaptation and Learning in Multiagent Systems. Springer Verlag Inc., Berlin, 1996.