Building Object-Agents from a Software Meta-Architecture

Analía Amandi and Ana Price

Universidad Nac. del Centro de la Pcia. de Bs. As. - Fac. de Cs. Exactas - ISISTAN -
San Martin 57 - (7000) Tandil - Bs. As.
Argentina
email: amandi@exa.unicen.ar

Universidade Federal do Rio Grande do Sul - Instituto de Informática
Caixa Postal 15064 - Porto Alegre - RS
Brasil
email: anaprice@inf.ufrgs.br

Abstract. Multi-agent systems can be viewed as object-oriented systems in which their entities show an autonomous behavior. If objects could acquire such skill in a flexible way, agents could be built exploiting object-oriented techniques and tools.

There are several ways for building agents from objects: defining common interfaces and behavior in abstract superclasses, wrapping objects with agent behavior using composition techniques, etc. However these ways present problems for becoming objects in agents and for adapting the behavior assigned to agents, especially whether it is required in a dynamic way.

This paper analyzes these problematic alternatives and presents another one in which agent characteristics (such as perception, communication, reaction, deliberation and learning) can be dynamically added, deleted, and adapted to objects using a particular computational reflection form achieved by meta-objects.

Keywords: software architectures, agent-oriented programming.

Building Object-Agents from a Software Meta-Architecture

Abstract. Multi-agent systems can be viewed as object-oriented systems in which their entities show an autonomous behavior. If objects could acquire such skill in a flexible way, agents could be built exploiting object-oriented techniques and tools.

There are several ways for building agents from objects: defining common interfaces and behavior in abstract superclasses, wrapping objects with agent behavior using composition techniques, etc. However these ways present problems for becoming objects in agents and for adapting the behavior assigned to agents, especially whether it is required in a dynamic way.

This paper analyzes these problematic alternatives and presents another one in which agent characteristics (such as perception, communication, reaction, deliberation and learning) can be dynamically added, deleted, and adapted to objects using a particular computational reflection form achieved by meta-objects.

Keywords: software architectures, agent-oriented programming.

1. INTRODUCTION

Agent-oriented and object-oriented programming work on cooperative entities. These entities have limited competence, which is defined by a set of actions that are able to execute. However, the following definitions show an important difference between the entities that each one of the paradigms work on.

An agent is an autonomous entity that has action capabilities, which are used in a non-deterministic behavioral way and internal knowledge for supporting their decisions and the execution of its behavioral capabilities. An object is an entity that has action capabilities defined in a deterministic way and internal knowledge for supporting the execution of its behavioral capabilities.

Objects and agents define a well-bounded behavior but there is one difference: objects use it in given ways and agents can decide the way they take. Thus, additional characteristics must be designed for achieving object-agents: communication among agents, perception of changes in the environment, and decision processes related to what to do next.

Communication capabilities determine how agents of different types can communicate among them. Examples of communication performatives: achieve (aGoal), askIf (aQuery), askAll (aQuery), etc.

Perception capabilities allow an agent to know facts without receiving any information from communications. Perception capabilities allow an agent to observe activities performed by other agents (a conversation among agents, transference of information between agents, destruction of agents, agents changing of communities, etc.) and environment changes (i.e. external sensors becoming enabled, locking of internal structures, etc.).

Decision capabilities determine the course of action that an agent takes in a given point of time. Such course of action can be characterized as reactive or deliberative [Demazeau 91]. Reactive behavior allows agents to act without thinking before act. Deliberative behavior shows a meditative selection of a particular set of actions.

We can recognize three behavioral levels in agents (see Figure 1). The lower level shows us what an agent can do in terms of actions. An agent robot, for example, could execute the following actions: go, turnLeft, turnRight, etc. This level represents the characteristic behavior of objects.

The next level represents social capabilities inherent to agents. Objects only can pass messages, they can not make a real communication as agent can do. Agents communicates with other agents requesting help to achieve a given goal, giving notice of a trouble, advising about a problem, giving information, etc. Also, agents can observe events happened in their environment.

The third level represents the set of decision capabilities that are used for agents to determine actions to execute. To do this, the observed events and the received communications are considered basic information for such decision processes.

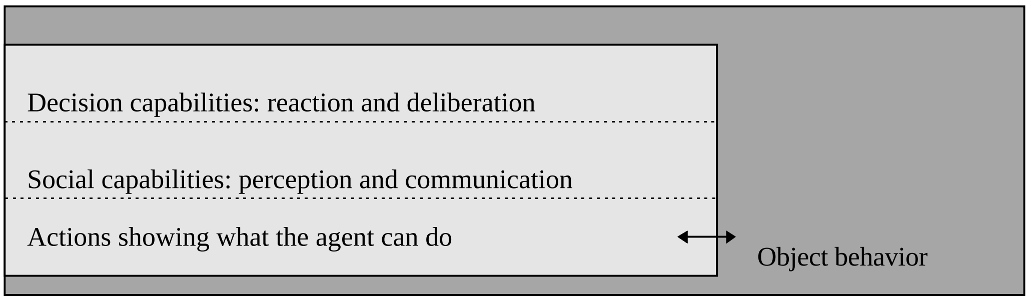
Decision capabilities: reaction and deliberation
Social capabilities: perception and communication
Actions showing what the agent can do
Object behavior

Figure 1 - Agent behavioral levels.

A multi-agent system can in summary be considered as an object-oriented system in which object-agents has associated extra capabilities. This idea is not new, [Gasser 92] [Shoham 93] have shown the potential characteristics of the object-oriented paradigm for building agents. However, there are several alternative ways in which these extra capabilities are associated to simple objects.

This paper analyses several alternatives concluding with the presentation of a software architecture named Brainstorm. This architecture allows objects be the main support of agents and meta-objects be the flexible way for adding the mentioned agent-specific capabilities.

The breakdown of the paper is as follows. Section 2 identifies the advantageous and troubles of several agent design alternatives based on object-oriented concepts. Section 3 presents an architecture based on the meta-object concept. Section 4 relates results and experiences with the architecture. Sections 5 and 6 expose related works and conclusions.

2. AGENTS AS OBJECTS

An obvious way for designing different agent types in an object-oriented context is the usage of classes representing each type of agent. This design way has been used in the design of several agent-oriented languages such as AgentSpeak [Weerasooriya 95], Daisy [Poggi 95] and CooL [Kolb 95].

An example of an object-agent is a robot. It can be designed through a class Robot which has defined methods for acting, communicating, perceiving, reacting, deliberating, learning and managing mental attitudes.

Using inheritance for designing agents, the common behavior of any agent type could be grouped in a class named by Agent. We can produce different types of agents by the specialization of this class. However, if inheritance is the only design technique used, the agent design presents several problems.

The main problems are (i) the explosion of the number of classes and (ii) duplicated code. The cause is that specific agents combine particular communication languages, particular decision algorithms and ways of behavior (reactive, deliberative or hybrid) in different ways. Communication languages, decision algorithms and ways of behavior must be duplicated in different point of the hierarchy (simple inheritance case) or complex hierarchies must be managed (multiple inheritance case). Static relationships among classes stated by inheritance produce such problems.

By introducing composition, dynamic alterations of structure and behavior of agents become possible. For example, the decision mechanisms used by agents can be flexibly designed through composition, avoiding code duplication and a large number of classes.

Figure 2 shows as agents can be dynamically built combing parts. Each part adds new functionality to a basic agent. In the Figure, a particular communication language, reaction functionality, deliberation functionality associated with a particular set of decision mechanisms are wrapping agent basic functionality.

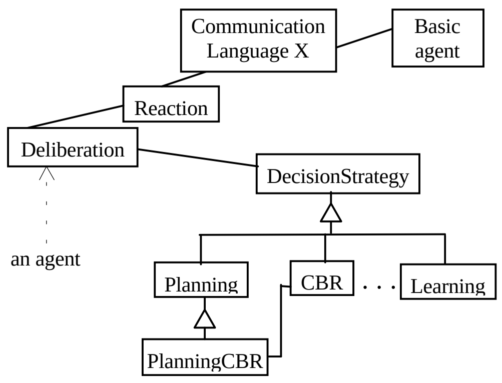

Figure 2 - An agent design using inheritance and composition.

Each agent can use different decision mechanisms any time and combine these mechanisms by composition. The design of such mechanisms achieves major advantageous combining inheritance and composition. Inheritance for adding new mechanism by class specialization. Composition for linking agents with decision strategies in a dynamic way.

Regarding perception design, we have to allow object-agents observing changes produced on their environment. This imply that agents must observe messages being sent among agents and objects.

As simple objects can not interfere in the sent messages, a simple solution is that agents and objects inform to the interested agents each one of their moves by messages. In this way, the natural observation of the environment looking for changes of interest become in a controlled announce of changes where the involved entities inform the changes they produce.

With this agent design, it is possible dynamically to alter agent behavior. However, the design presents some troubles: agent developers must manage each relation among agent parts and the perception design forces changes in code of the object. The administration of parts involves control of the activation order (i.e. react before deliberate) and implicit delegation of execution responsibility. The perception solution imposes changes in the basic code of agents because of the announces, avoiding the full possibility of dynamic changes in the observation targets.

Another alternative, which is the support of our agent architecture, is the usage of computational reflection by meta-objects. Computational reflection is an activity carried out by a computational system when it executes computations on its own computation [Maes 87]. From this point of view, a reflective system is divided in two levels: an object level and a reflective level. The object level manages the actions and data of the application domain. The reflective part, located in a meta-level, manages actions and data of the object level.

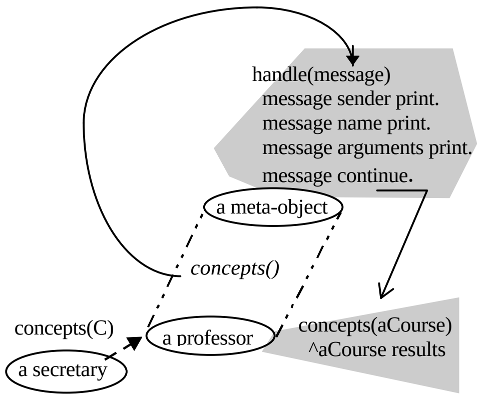

Figure 3 - Meta-objects.

Figure 3 shows an example of computational reflection from an object-oriented point of view. In the object-oriented approach, the concept of meta-object materializes the concept of computational reflection. The example shows as a meta-object interferes in the computation of the messages received by a given professor. The meta-object prints the sender, the name, and the arguments of each message received and so returns the control to the professor.

Coming back with the agent design, the problems found using only inheritance and composition can be solved using computational reflection techniques. Meta-objects can interfere in the passive behavior of objects turning it an autonomous behavior.

Therefore, a multi-agent system can be considered as an object-oriented system which has associated to it a meta-system responsible for a typical agent behavior. The next section present a solution for designing agents from object in this way.

3. THE ARCHITECTURE BRAINSTORM

A software architecture [Shaw 96] describes software components, patterns of component composition and restrictions on these patterns, which allow systems to be built. An agent architecture allow multi-agent systems to be built in term of given components and interfaces among them.

We have developed an agent architecture denominated Brainstorm for extending objects to support agent behavior. In such way, agent characteristic behavior has been encapsulated in components and specific interfaces have been defined for connecting those agent components.

In the Brainstorm context, a multi-agent system is defined as a reflective system, which is causally connected to an object-oriented system defined in the base level. The causal connection is established when an object receives a message: it is intercepted by the meta-level, which decides what to do with it. For example, an agent can decide to deny a specific request of another agent.

The architecture has been built using a meta-object approach. An object and a set of meta-objects define an agent. Each meta-object incorporates agent functionality to simple objects.

The functionality that can be added to objects from meta-objects can be observed in Figure 4. We can see two object-agents. The agent A has associated seven meta-objects and the agent B has four. Requirements of each agent originates the necessity of different quantity and types of meta-objects. For example, the agent A uses a particular communication language (by a communication meta-object), a particular set of mental attitudes (by a knowledge meta-object) and functionality related to their hybrid behavior (reaction (by a reactor meta-object), deliberation (by a deliberator meta-object) and learning (by a learner meta-object)), and perceive changes in its environment originated by the agent B and one given object (by two perception meta-objects).

In the Figure, we can see agent A observing agent B by a perceptor meta-object. In addition to perceptive connections, agents can interact among them sending messages using a communication language.

Also, cooperative decisions can be made using an special object (*cooperative element* in the Figure) which can be acceded by a given set of agents in contrast of the rest of the private components.

Several reflective levels composed by meta-objects can be observed in each agent. Each level is connected to the lower level by reflection. In other words, each level intercepts all event of its lower level and changes its limited behavior.

The first meta-level is composed by a communication meta-object, a knowledge meta-object and a set of perception meta-objects. These meta-objects works on messages received by the base level (the object base). This level is responsible for recognizing messages that belong to the communication language, for managing the mental state, and observing messages that can represent interesting situations for the agent.

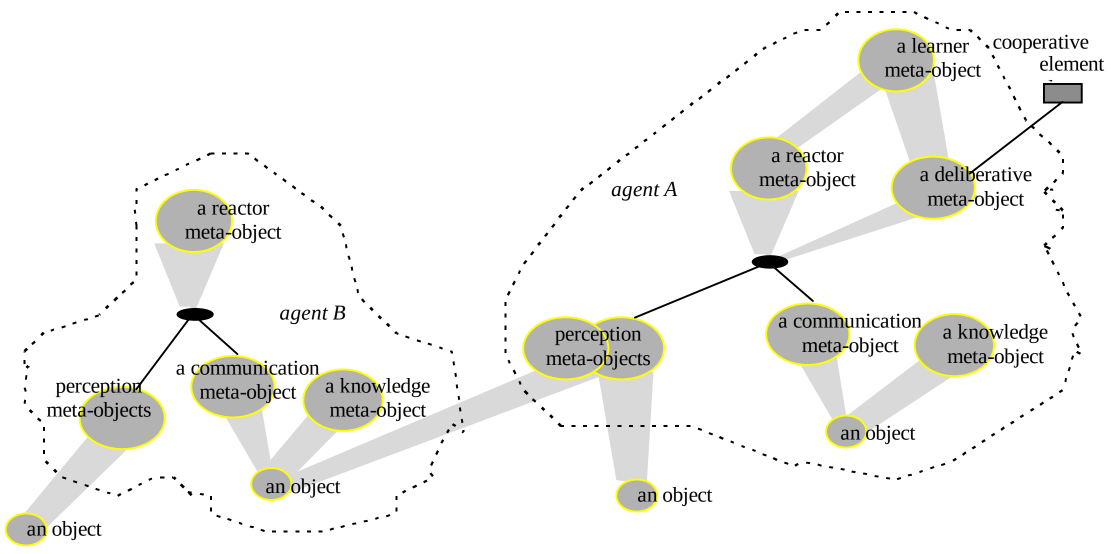

Figure 4 - Brainstorm agents.

The second meta-level is responsible for acting. The possible ways for acting are reaction and/or deliberation. Therefore, it is possible have in this level a reactor meta-object and/or a deliberator meta-object. These meta-objects work on communications processed by the communication meta-object and on interesting situations discovered by the situation manager. A situation manager (black circle in Figure 4) analyzes whether a situation of interest is present. For it, it uses as data the received communications and the messages observed by perceptors.

The third meta-level is optional. Learner meta-objects are responsible for learning from experiences and for using these experiences in future decisions. These meta-objects can work on reactions and on decision algorithms used by deliberators.

The mentioned meta-objects are the main architectural components in terms of agent instances. In terms of agent class there is n architectural component named MetaAgent which is materialized in the meta-level of a class. This component is responsible of the process of meta-object instanciation. When a class instanciation message is received by an agent class, its associated meta-agent intercepts it and generates the meta-level of the new object-agent. Meta-agents create the meta-objects needed for satisfying the agent specification. Thus, it is possible to define different types of agents in the same system.
The following sections present in detail each component of the architecture Brainstorm.

3.1 The communication component

Communication meta-object classes defines communication protocols which may be adopted by agents. Agents that want to interact have associated instances of the same communication meta-object class.
When an object-agent receives a message *m* that belong to its communication language (see Figure 5), this message is intercepted by its agent communication meta-object by a classical meta-object mechanism. This mechanism actives the methods *handleMessage()* of the first meta-level with the intercepted message as argument.

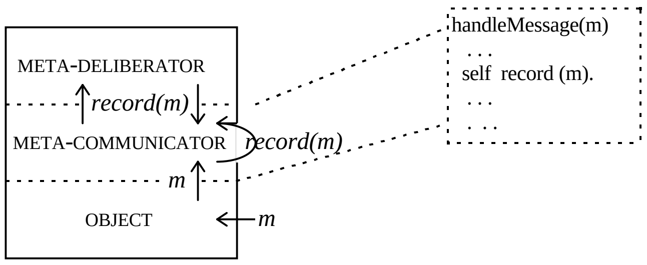

Figure 5 - Communication.

That message is then analyzed by the communication meta-object for deciding what to do with the message. If the message has been defined either in the object class as an acting capability or in its communication meta-object class as part of its communication language, it is registered on a communication message base using an architectural protocol *register()*. The priority of managing that message is lower for default.
In this occasion, the deliberator meta-object can intercept the message *record()* to define other message priority. For doing so, the deliberator uses its own mental state. For instance, if a request sender is the boss of the receiver, it has the highest priority of

attending. Then, the control returns to the meta-communicator and it records that message in the priority defined by the meta-deliberator.

Each one of the meta-components has defined independent responsibilities. These meta-component work together showing a transparent connection by a reflection mechanism.

3.2. The perception component

Agents can observe events taking place in their environment. Agents and objects compose the environment in Brainstorm, and so the message passing is the only source of events. These events can draw a situation of interest for agents.

To perceive events, agents shall include definitions of perception meta-objects (instances of MetaPerception class). These meta-objects observe other agents or objects, reflecting their received messages. Both agents and objects can be observed without they know they are observed in contract of the classical announce of changes where the observed object must announce its movements.

Using meta-objects for percepting events in the environment, agents have two advantageous. One is that the transparency of the perception, the observed agent or object is not conscience of the observation. The other advantageous is that agents can in runtime move their observation targets.

This architectural component use the common protocol for meta-objects, the method *handleMessage()* for intercepting observed messages.

3.3. The situation manager component

The situation manager is responsible for searching situations of interest. For doing so, it evaluates its perceptions, received messages and the mental state. Figure 6 shows the static connections established with the situation manager. Perceptor meta-objects informs to the situation manager all observed event, the communicator meta-object informs to the situation manager all received communication. The situation manager can see and use the knowledge and believes encapsulated in the component named brain.

Here we present one architectural component, brain. This component is responsible for the mental state of the agent. Details about this component are showed in the next section.

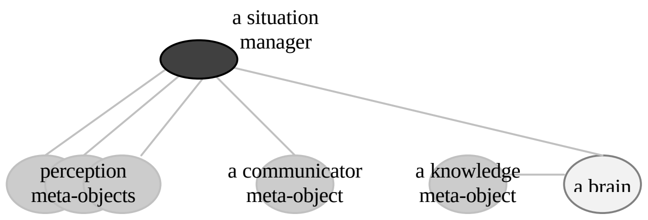

Figure 6 - Connection to the situation manager.

The situation manager uses information about perception, communication and mental attitudes for detecting interesting situation. Whenever an interesting situation is detected, a method using a protocol *newSituation()* is invoked on the own situation manager. Transparently this message is reflected by the meta-objects MetaReactor and MetaDeliberator of the next meta-level. These meta-objects decide how to manage such new situation.

For example, an agent Agenda may want to know when given people take particular types of commitments. The agent wants to be more careful on the evaluation of accepting of commitments that involve people that are related to rivalry groups. The agent could ask to each agent agenda for such information (through a message), but a better solution would be to put sensors on the agendas to observe their commitment management.

The implementation of these sensors is very simple: the agent agenda associates a perception meta-object to the another agendas to observe the reception of the messages *canAccept(aCommitment)* and *accept(aCommitment)*. With information about intercepted messages and the mental state, the agenda' situation manager can analyze whether such commitment is a interesting fact. If it is the case, it sends the message *newSituation()* to itself, which is reflected by both the reaction and deliberation meta-objects that evaluates what to do with such new situation. The reactor can produce a reaction. The deliberator can analyze what actions to take.

3.4. The knowledge component

This architectural meta-component permits objects managing logical structures for mental attitudes. Object-agents need an instance of this component.

The architecture prescribes objects using modules composed by logic clauses for managing mental attitudes. The type of clauses this component can manage is selected among a big set of possibilities since there is not a general agreement about both the set of mental attitudes and the relationships among them that should shape agent mental states. Several formal frameworks have been proposed [Cohen 90] [Rao 91] [van Linder 96] [Huang 96] in order to elucidate relations among mental components. Any extension of classical logic programming that applies these theoretical concepts can be adopted by particular agents.

The object uses logic modules in the base level. This usage is reflected by the meta-knowledge and managed by the component named brain. The component Brain has defined as its interface the needed protocol for maintaining and consulting a knowledge base. The Brainstorm architecture prescribes that Brain is an specialization of the classical logic programming.

From the programming point of view, this capability is achieved since an integration of an object-oriented language with a logic language is possible [Ishikawa 86] [Fukanaga 86] [Mello 87] [Vaucher 88]. Extensions of classical logic programming (such as [Costa 95] [Amandi 97a]) for managing mental attitudes are used as a base of the component Brain.

The extensions to a Prolog interpreter can be easily added as specialization of Brain component since the knowledge meta-object administrates the integration with objects. This meta-object permits the combination of different logic modules referred by instance variables and methods and makes it temporally available for use.

3.5. The reaction component

The reaction meta-object has the functionality of reaction to given situations. A reaction itself is composed by a situation and a sequence of actions that are executed when this situation is detected. So this component would in summary be to use for quick answers to particular situations.

The reaction meta-object of an agent manages immediately each detected interesting situation whether a reaction was defined for such situation. After that, the deliberator also can act.

The Brainstorm architecture prescribes a meta-object with reflective connections with the situation manager in the lower level and with the learner meta-object in the upper level. This prescription has the flexibility of changing the behavior of agents as main advantageous.

3.6. The deliberation component

The deliberation component is responsible for the intelligent and autonomous acting form of the agent. The architecture prescribe a meta-object for this component. This meta-object is connected by reflection with the situation manager, the meta-communicator and the meta-learner (if there is one).

An agent acting form is determined by the manner in which it has to act when: (a) it perceives a situation of interest, (b) it receives a message, and (c) it decides to do something. Different acting forms are defined in different subclasses of the MetaDeliberator class or its parts. Brainstorm architecture defines two important parts for abstracting decision behavior. Figure 7 shows those components.

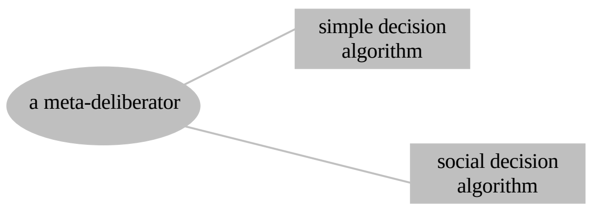

Figure 7 - Deliberation component and its decision subcomponents.

When an agent perceives a situation (by its situation manager), it must decide what to do in such situation. For doing so, the meta-deliberation intercepts the detected situation and delegates it to one of its simple decision algorithm.

When an agent receives a message, it have attend the message. For doing so, first the meta-deliberation intercepts the received message for defining a priority of attending. Second when the message is attended it delegates the communication message to one of its social decision algorithm. Such algorithm analyzes the communication considering the social state and decide what is the answer.

An agent also act without the necessity of any communication or perception of situations. A quantum of time is given to each agent for acting freely if parallel process is not possible. Agents use this time for treating pending communications, making plans to achieve their goals (using the available planning algorithms), analyzing their old plans, goals and intentions, and executing actions in their environment. Therefore, it is necessary to have defined priority rules for guiding the decisions about what to do next.

3.7. The learner component

The learner component is defined as a meta-object by the Brainstorm architecture. It intercepts reactions and deliberations. The reactions are analyzed permitting future changes on themselves. Deliberations can be altered incorporating new points of view on the object on analyses.

For example, an instance of this component is a case-based reasoning engine. This element can intercept, for example, a deliberation process that use a planning algorithm, deciding the resulting action plan.

4. RESULTS AND EXPERIENCES

The Brainstorm architecture has been formally specified using the Z language. For these schemes, seven properties were proved. We have been proved that all message received by an object-agent take a right way from an agent behavior point of view. Detail of these proves can be found in [Amandi 97b].

In addition to the formal results, two experiences were made. First, a multi-agent systems for simulating robots in a room was developed using the Smalltalk language. Second, a multi-agent systems for managing personal agendas was implemented in the same language, reusing several decision algorithms.

These experimental cases showed that the abstractions defined in Brainstorm can be directly used and helped to construct a base of reusable decision algorithms and communication languages.

5. RELATED WORK

Several agent architectures have been proposed in the literature (i.e. [Ferguson 92] [Müller 94] [Hayes-Roth 95]) describing the main components and interfaces of software agents. These agent architectures prescribe a computational solution for agents construction.

In the other hand, lots computational solutions of software agents have been built on object-oriented languages and lots agent specific languages have been developed on object-oriented concepts [Fisher 94] [Kolb 95] [Poggi 95]. In spite of that, the existent agent architectures have not considered in their design the usage of object-oriented concepts.

6. CONCLUSIONS

Brainstorm is a software architecture that prescribes an object-oriented agent solution, allowing thus the usage of object-oriented languages in the agent implementations. The object-oriented base has two advantageous: old object-oriented systems can be extended for becoming multi-agent systems and all support for object-oriented systems can be used for building agents.

The Brainstorm architecture prescribes several reflective components materialized by meta-objects. This prescription makes possible the dynamic modification of the structure and behavior of agents. Such flexibility also allows old objects become agents.

7. REFERENCES

[Amandi 97a] Amandi, A.; Price, A. An Extension of Logic Programming for Manipulating Agent Mental States. In: Proceedings of the Artificial Intelligence and Soft Computing. Canada, 1997. p.311-314.

[Amandi 97b] Amandi, A. Programação de Agentes Orientada a Objetos. Doctoral Thesis. Porto Alegre: UFRGS, 1997.

[Cohen 90] Cohen P., Levesque H. Intention is Choice with Commitment, Artificial Intelligence, 42(2), 1990.

[Costa 95] Costa Mora, M.; Lopes, J.; Coelho, H. Modeling Intentions with Extended Logic Programming. Proceedings of the Brazilian Symposium on Artificial Intelligence. 1995. P.69-78.

[Demazeau 91] Demazeau Y., Muller J.P. From reactive to intentional agents. In Decentralized Artificial Intelligence 2. pp.3-14. 1991.

[Ferguson 92] Ferguson I. TouringMachines: Autonomous Agents with Attitudes. Computer, 25(5), pp.51-55, May 1992.

[Fisher 94] Fisher M. A Survey of Concurrent METATEM-The Language and its Applications. Temporal Logic. 1994. pp.480-505. (LNAI 827).

[Fukanaga 86] Fukunaga, K; Hirose, S. An Experience with a Prolog-based Object-Oriented Language. SIGPLAN Notices, 21(11), pp.224-231, Nov. 1986.

[Gasser 92] Gasser, L.; Briot, J.P. Object-Oriented Concurrent Programming and Distributed Artificial Intelligence. In: Distributed Artificial Intelligence: Theory and Praxis. Kluwer. 1992.

[Hayes-Roth 95] Hayes-Roth B. An architecture for adaptive intelligent systems. Artificial Intelligence, 72(1-2), pp.329-365, Jan. 1995.

[Huang 96] Huang Z.; Masuch, M.; Pólos, L. ALX, an action logic for agents with bounded rationality, Artificial Intelligence, 82(1), 1996, pp.75-127.

[Ishikawa 86] Ishikawa, Y; Tokoro, M. A Concurrent Object-Oriented Knowledge Representation Language Oriente84/K: Its Features and Implementations. SIGPLAN Notices, 21(11), pp.232-241, Nov. 1986.

[Kolb 95] Kolb M. A Cooperation Language. Proceeding of the International Conference on Multi-Agent Systems. 1995. pp. 233-238.

[Kolodner 93] Kolodner, J. Case-Based Reasoning. 1993.

[Maes 87] Maes, P. Concepts and Experiments in Computational Reflection. Proceedings of the Object-Oriented Programming Systems, Languages, and Applications Conference, 1987. p.147-155.

[Mello 87] Mello, P; Natali, A. Objects as Communicating Prolog Units. Proceedings of the European Conference on Object-Oriented Programming. 1987. pp.181-191.

[Müller 94] Müller J., Pischel M. Modeling Interacting Agents in Dynamic Environments. Proceeding of the European Conference on Artificial Intelligence, 1994. p.709-713.

[Poggi 95] Poggi A. DAISY: An Object-Oriented System for Distributed Artificial Intelligence. Intelligent Agents. 1995. p.341-354. (LNAI v. 890).

[Rao 91] Rao A.; Georgeff M. Modelling Rational Agents within a BDI-Architecture, Proceedings of Knowledge Representation and Reasoning (KR'91), pp.473-484, April, 1991.

[Shaw 96] Shaw, M.; Garlan, D. Software Architectures. Prentice Hall. 1996.

[Shoham 93] Shoham, Y. Agent-Oriented Programming. Artificial Intelligence. v.60, n.1, p.51-92, Mar. 1993.

[van Linder 96] B. van Linder, W. van der Hoek, Meyer, J. Formalising Motivational Attitudes of Agents: On Preferences, Goals, and Commitments. Intelligent Agent II,1996.

[Vaucher 88] Vaucher, J; Lapalme, G; Malenfant, J. SCOOP: Structured Concurrent Object-Oriented Prolog. Proceedings of the European Conference on Object-Oriented Programming, 1988. pp.191-211. (LNCS 322).

[Weerasooriya 95] Weerasooriya, D.; Rao, A.; Ramamohanarao, K. Design of a Concurrent Agent-Oriented Language. In: Wooldridge, M.; Jennings, N. (Eds.). Intelligent Agents. Berlin: Springer-Verlag, 1995. p.386-401. (LNAI, v. 890).