Autonomous Objects: a framework for Intelligent Software Engineering Environment

Janete P. Amaral  
E-mail: jpa@di.ufpe.br  

Jaelson F. B. Castro  
E-mail: jbc@di.ufpe.br

Departamento de Informática  
Universidade Federal de Pernambuco  
Caixa Postal 7851 - CEP 50732 - Recife-PE - Brasil  
Fone: (081)-2718430 - Fax: (081)-2714925

Abstract

The object-oriented paradigm (OOP) has been applied on the construction of intelligent systems using the metaphor of cooperating intelligent agents. The iteration and the cooperative work, essential characteristics of the software development process, make the modeling of software engineering environments (SEE) suitable for this approach. This paper describes the architecture of a software development assistant based on researches on OOP and Distributed Artificial Intelligence. This assistant was implemented using autonomous objects that constitute an intelligent computational unit which controls its own behaviour, possesses the necessary knowledge to accomplish its activities, acquires knowledge and communicates with objects and users in order to fulfill its tasks. Multiparadigm resources were used for the implementation of these objects. Through this assistant, the feasibility of this framework for construction of intelligent SEE was demonstrated.

Key-words: Object-Oriented Paradigm, Multiparadigm resources, Software Engineering Environment, Distributed Artificial Intelligence, Multiagent Systems.

1 Introduction

The software development process consists of coordinated execution of several activities. In these activities users, managers, analysts, designers, and programmers are involved, possibly geographically dispersed. Each one of this professional performs a specific role, and through the cooperative work of this group, a software is developed. One effective strategy to produce software of the better quality is to provide the highest iterativity in this process.

Current researches on Software Engineering Environments (SEE) aim at detection of structures which allow the construction of open, flexible, and active environments, offering conditions for the execution of software development activities, as accomplished on conventional way [10]. In this context, open means that the environment can be adapted for different models of development, enabling an easy inclusion of tools. Flexible means that the environment can be adapted dynamically to changes in a development model. Active means that the environment can understand the actions done by the users, and can use this information to provide context-sensitivity help, driving the users through the development process [7].

Software Engineering has been an interesting area of research for the Artificial Intelligence (AI) community. The reason for that has been the recognition of the complexity of the activities of the
software development in-the-large, which requires a great amount of knowledge and intelligence. The software engineering problems include questions studied in isolation by the AI, such as human-machine interface, computer-support cooperative work, and knowledge representation.

The Distributed Artificial Intelligence (DAI), using the metaphor of intelligence based on intelligent social behaviour, looks for strategies for cooperative problem-solving. Studies on DAI emphasize the actions and interactions of intelligent elements (cognitive agents) and non-intelligent elements (reactive agents) of a society [4]. This approach is suitable for complex problem solving, which requires knowledge of several domains and sometimes involves distributed data management. For system implementation, the DAI approach has suggested several computational models, such as blackboard architecture and the actor model. The object-oriented paradigm (OOP) through concurrent object-oriented languages, constitutes a new horizon for these researches. The OOP characteristics of knowledge modularization, the extensibility allowable by classes specialization, and the communication strategy, in which is used the metaphor of intelligent cooperative agents, have been the motivation for these researches.

The cooperative work in software development and the maximum interactivity required in this process, makes SEE an application suitable for this approach. Using OOP in intelligent SEE construction, where the intelligence is modeled by DAI approach, each object will be specialized on particular activity of software life cycle (e.g. analysis, design, and implementation). Every object works cooperatively to meet the development purpose. In this environment the knowledge is stored on a knowledge base partitioned on objects, enabling the reasoning capacity to be distributed among these objects. The suggested approach allows the construction of intelligent SEE which are open, flexible, and active. Therefore, a SEE evolves during runtime as result of interactions among users and the environment, providing effective help for the software development activities.

In this paper we draw results from DAI and SE research areas, to define an object model to be used as structure for construction of intelligent SEE. In section 2 we describe some experiences, which represent the background for this research. The section 3, presents the main characteristics of the object model proposed. In section 4, details of the implementation of the model using multiparadigm resources are showed. Section 5 describes a software development assistant, designed to show suitability of the proposed model. Finally, the conclusion with future directions of this research is presented in section 6.

2 Related work

Recent work has explored the inclusion of special mechanisms to the object model to represent knowledge and to add autonomy and reasoning capacity on objects. There have been proposals from the DAI researchers for the implementation of intelligent systems. In some of these experiences multiparadigm resources were used, where the features of OOP were deployed in combination with logic programming, rule-based, or access-oriented resources.

The proposed models have been used on applications such as behavioral simulation, intelligent diagnostic, expert systems shells, and cooperative-distributed problem solving. The mechanisms used aimed at particular characteristics of these applications. In the sequel, we present a short description of some recent experiences.

Tokoro [14], using the Knowledge Objects (KO), presents a multiparadigmatic model for construction of knowledge systems, composed of several cooperating subsystems. Each KO consists of a behaviour part, a knowledge part, and a monitor part. These objects store knowledge and have a reasoning capacity, characteristics provided by logic programming resources. The behavioural
methods of these objects make inference on their knowledge, include new knowledge or update knowledge dynamically. The object oriented language Orient84/K has been designed to implement these objects.

The Knowledge, dissemination, and manipulation ObjectS (KNOs) were proposed by Tsichritzis [15]. These objects were projected for construction of assistants that perform tasks which involve cooperation, negotiation, and knowledge acquisition. Their experiences address office-automation area. A Kno is composed by a structure and a behaviour which is determined by the operations it can perform. Each Kno operation is composed by a set of production rules. Knos communicate with each other by posting messages on a blackboard, that is managed by the object manager.

Maruichi [9] has proposed a model for distributed cooperative problem-solving, where the autonomy of the object is provided by an intelligent message interpreter. To experiment this proposal, the concurrent object-oriented language Pandora II was implemented.

A number of researches in this subject have been presented. For the sake of simplicity, we do not mentioning all here. In those researches two trends are perceived. On one hand, researchers in Distributed Artificial Intelligence searching from Cognitive Science characteristics for modeling agents, where aspects such as behaviour, autonomy, adaptability, exchanged knowledge and society behaviour of the agents are studied. On the other hand, researchers in concurrent object-oriented languages are searching for mechanisms to implement these aspects studied by DAI. We consider that these two trends will provide insights to a suitable approach for modeling and implementing of intelligent SEE.

3 The object model proposed

We propose an object model that enables the inclusion of special mechanisms in the object model of OOP, endowing objects of intelligent behavior (as regarded by researchers on DAI). This project intends to use these objects as structure for the implementation of Intelligent SEE, considering the main requirements of this application, that are incorporate knowledge and present intelligent behaviour, to effectively assist the users on their tasks fulfillment. The main characteristic of this project is the development of an object model with few mechanisms and of ease implementation. It will be used as a kernel for construction of more sophisticated models.

The actual implementation of the proposed model aims at exploring the possibilities of this structure as a framework for construction of Intelligent SEE. Through this implementation, we also explore object- oriented software development methods and requirements in SEE.

For the reminder of this section, we present the main characteristics of the object model proposed, as well as aspects of its operation. In the next section, we give details of the implementation, in which we used multiparadigm resources.

3.1 Characteristics of the objects model

We define objects with an internal structure similar to intelligent systems. An application will be thought of as a set of cooperating intelligent systems. These objects possess characteristics considered essential in order for system exhibit intelligent behaviour [5]. These characteristics are:

- representing declarative and procedural knowledge on the application domain;
- having autonomy on tasks fulfillment;
presenting flexible behaviour, that is being able to adapt to the environment conditions;
permitting dynamic modifications on their behavior, as a result of the knowledge acquired during runtime; and
interacting with the external environment, providing assistance and report on their activities.

The proposed model is composed of two types of objects: autonomous and servers. These objects are defined on classes, and classes hierarchy are structured. Every communication among objects will be done by message passing.

3.1.1 Autonomous object

The autonomous objects are active, present intelligent behavior, and acts as expert on particular area. These objects possess knowledge about their tasks and autonomy on their actions. They cooperate with others objects to meet the system goals and acquire knowledge. The autonomous objects classes have all components defined to these objects and they are subclasses of a root class, that hold their basic characteristics. These objects behave like a system process. Messages to autonomous objects are objects as well. An autonomous object is composed by an identification, a knowledge- base, a knowledge acquisition mechanism, and a communication mechanism. The Fig. 1 illustrates the integration of these components.

![Autonomous object components](image)

Figure 1: Autonomous object components

The role of each one of these components is:

a. Identification: internal identity of the object.

b. Knowledge base: represents an object memory, where the declarative, procedural, and behavioural knowledge is stored. Knowledge is provided by the class definition, inherited by subclasses, and acquired during the life cycle of the object.

- Declarative knowledge: contains information about object properties, such as: attribute values, relationships, default values, and restrictions.
- Procedural knowledge: contains procedures, describing how the actions are executed. Correspond to methods found in the object oriented paradigm.
- Behavioural knowledge: describes what the object have to do, identifying the conditions for object behaviour activation. It enable objects not only to react to external events, but also to act on recognition of exceptional states in their knowledge.
c. Communication mechanism: responsible for object communication. It starts the object activities, receives and reply requests from objects, asks for the cooperation of other objects, and interacts with users. It is composed of:

- Message queue: area for storage of messages with requests for object cooperation. The message queue acts as an object mailbox.
- Message manager: executes procedures to start activities, to insert and remove messages from the message queue, and to reply to message requests.
- User Interaction component: executes procedures for communication with users, providing helps, warnings, explanations, and trace of activities performed.

![Communication mechanism diagram](image)

Figure 2: Communication mechanism

d. Knowledge acquisition mechanism: responsible for the maintenance of knowledge to be stored on the object knowledge-base. This mechanism enables knowledge acquisition through users as well as by the object itself.

![Knowledge acquisition mechanism diagram](image)

Figure 3: Knowledge acquisition mechanism

At *autonomous object* creation, its components are automatically generated. After that, the object can accomplish tasks, receives and reply messages, acquires knowledge, and provides help. The *autonomous objects* are aware of the objects they communicate with. Therefore, this knowledge is provide on knowledge-base of this objects.
The action to be performed when the message is sent to this object is only the inclusion of this request on its message queue. The sender object does not wait for message reply. The receiver decides, if and when, it has to perform a reply action. For execution of its activities, the autonomous object queries the behaviour knowledge, which establishes the conditions for its execution.

The autonomous objects can acquire knowledge as result of their activities. Thus, they can modify their behaviour dynamically through inclusion of the declarative, procedural, and behavioural knowledge in their knowledge-base. The objects also can acquire knowledge from users. In this case, the communication mechanism is used.

3.1.2 Server objects

The server objects are passive and are used to help the autonomous objects tasks. They execute only their tasks when requested by others objects, i.e. they a passive, and possess the same characteristics of OOP objects, as presented on Smalltalk/V.

4 The object model implementation

In order to implement the proposed objects a language which permits the exploration of OOP and logic programming resources was required. The logic programming will be used, because it caters for:

- the representation of behavioural knowledge of the autonomous objects, through rules in backward chaining, enabling the inference on the stored knowledge;

- the fulfillment of declarative knowledge aspects, not explicitly provided in class definition, neither deduced through OOP representation model; and

- the acquisition of declarative knowledge in runtime. In this case, features of database manipulation provided by Prolog will be used.

The language Smalltalk/V with feature of logic programming provided by Prolog/V was chosen. The use of logic programming, in Smalltalk environment, was attractive to this research, because, besides this environment already provides facilities required, others features available in the Smalltalk/V environment can be used. In sequel, we describe the decisions made on the implementation of the object model.

In Smalltalk/V, the logic programming resources are implemented in the Prolog class, subclass of the Logic class. For construction of classes taking advantage of these resources, these classes need to be subclass of Prolog class. The Fig. 4 presents this hierarchy.

In order to simulate the existence of active objects, we implemented the Scheduler class. This class has an object identification of its subclass. The objective is to transfer the thread-of-control to each object which needs to become active to carry out one cycle of its activities. The Autonomous class contains the basic characteristics of the autonomous objects, which will be inherited by application classes. The server objects are instance of others classes provided by Smalltalk/V environment.

The characteristics of the autonomous objects are translated to resources provided by Smalltalk/V and Prolog/V languages.
Figure 4: Class hierarchy of the model

a. Identification – An internal identification of an object is provided by an address stored in object pointer table.

b. Knowledge base – The declarative knowledge is represented through instance variables and through methods coded in logic programming syntax, as fact about the object. The declarative knowledge acquired during runtime is provided through a special variable, denominated \textit{kb}, whose function is to store the identification of the auxiliary databases. Facts can be included or removed from these databases. The procedural knowledge is represented by methods on Smalltalk/V syntax. The behavioural knowledge is represented as follows in logic programming: \textit{rule}: ("procedure") :- condition 1, condition 2, ..., condition n. Where \textit{procedure} is a Smalltalk/V method selector, and \textit{conditions 1 through condition n} are predicates, which obtain results of queries on declarative knowledge, receives results of procedural methods execution, and returns the evaluation of these results.

c. Communication mechanism – The \textit{message queue} is implemented as an object variable instance. This variable is initialized with one instance of \textit{OrderedCollection} class of the Smalltalk/V. With this procedure, the variable behaves as a queue to receive instances of the \textit{Message} class. The incoming messages are included at the end of the message queue, in the arrival order. The \textit{message manager} functions are performed by methods on the Smalltalk/V syntax. These methods include messages in the message queue, consult the behaviour rules, invoke messages and delete replied messages. The \textit{user interaction component} is formed by methods on Smalltalk/V syntax, to present trace of activities executed, to provide assistance to users, and to exhibit the behavioural and procedural knowledge of the objects.

d. Knowledge acquisition mechanism – It implements functions of creation and maintenance of the object knowledge-base. This base receives information provided by users or generated by the system.

Messages sent to autonomous objects are server objects, instance of the \textit{Message} class (Fig. 4). The message passing to autonomous objects consists of two steps. First, it is created one instance of the \textit{Message} class, and then, it is provided the arguments for execution of its methods \textit{sender:}, \textit{selector:}, and \textit{arguments:}. This instance is sent to target object, as follows:

\begin{verbatim}
Msg := Message new.
Msg sender: [anObject]; selector: [aSelector]; arguments: [anArgument].
[anAutonomousObject] put: Msg.
\end{verbatim}
Messages to *server objects* follow the same message pattern of the Smalltalk/V objects.

In the next section, features of the proposed assistant is described emphasizing its structure, functions, and mechanisms.

5 Assist/DS: The Software Development Assistant

This assistant was designed to test the feasibility of the use of *autonomous objects* as structure to SEE, as well as to analyze possibilities provided by this approach. Its objective is to drive the software development process, providing interaction with the users, help on the execution and coordination of their tasks, and validation of the steps of the chosen development method. In this assistant, users and environment work cooperatively.

The knowledge stored in this assistant includes the development method that the developer must follow, characteristics of the users, and some heuristics about the domain of the system under development. The chosen method comprises of several phases and an object-oriented method is used for all phases. The development process is decomposed of individual activities, identifying total and partial dependencies within them. The knowledge about the users is stored to permit an effective assistance on their tasks fulfillment. The Assist/DS stores the development history through register of the activities, providing a software evolution memory, which is dynamically updated as sub-product of development. The history of development can be feed back to project managers in order to monitor development progress.

5.1 Structure

The Assist/DS structure is composed of an hierarchy of *autonomous objects* classes. This hierarchy is formed by stages, phases, activities, techniques, documents, and others elements of the software development process. The top level of this hierarchy is shown in Fig. 5. The Assist/DS root class is called *Assistant* and it is implemented as subclass of the *Autonomous* class. This class is further decomposed into several subclasses. The others classes are *server objects*, and thus, do not belong to the application root class. In the Assist/DS, information about this components, the actions to be performed, and the conditions in which these actions will be performed, form, respectively, the declarative, procedural, and behavioural knowledge of the *autonomous objects*.

The software development process is distributed within objects that form the Assist/DS. As the stages, phases, and activities are *autonomous objects*, the behavioural knowledge of each one of these objects indicates the pre-requisites to a particular stage, phase or activity to be done. Therefore only those resources that are directly necessary for the current task are deployed. The behavioural knowledge of the *autonomous objects* establishes the Assist/DS action plan.

5.2 Functions

The Assist/DS should provide complete life cycle coverage. Thus, possess functions for coordination, planning, development, and operation. These functions correspond to stages of the chosen software development life cycle.

a. Coordination – This function is concerned with activities executed by the Assist/DS coordinator, and by the project manager. Information provided by this function concerns users and
their responsibilities, projects managed by the environment, available resources, diagramming techniques, document models, and the work method.

b. Planning – it is related to activities inherent to systems planning stage of the organization. For these activities, graphical and text editors are provided.

c. Development – it is concerned with activities of software development process. These activities are grouped in phases, composed of specific procedures. For each activity tools provided by Assist/DS.

d. Operation – it is related to activities executed after the software development and after the software release, such as evaluation and maintenance.

Figure 5: Assist/DS class hierarchy

5.3 Mechanisms

In this section, it we describe details of the mechanisms provided by the Assist/DS. These mechanisms are supported through user-interface, assistance, tools, and method provided.

Find in Fig. 6 an overview of the Assist/DS interface, where it is exhibited, in the left hand side, the trace of the activities executed. At the bottom it is presented messages provided by the assistant. For each phase of development process a specific window is provided. For implementation of the user-interface, Smalltalk/V resources were used.

A peculiar feature of this assistant is related to the assistance provided during the software development. The Users class is implemented for representing the capacity and goals of (a group
of) users. Users of the Assist/DS fall into three categories: coordinator, project managers, and software developers. For each user, an instance of this class is created. The assistance applicable to each user is defined in its model. Fig. 7 exhibits details of this assistance. Three levels of assistance are defined. The Assist/DS can modify the level, according to the interaction performed by user. During the lifetime of the Assist/DS, the user model evolves. These characteristics are provided by knowledge acquisition mechanism of the autonomous objects. The special user, called Coordinator, is responsible for tailoring the environment through modifications of the method used and inclusion of new techniques and documents.

Using the autonomous objects proposed, users can, at any moment, get information about the activation status of the behavioural knowledge of the objects. The message passing mechanism permits the logging of all activities executed as well as the requests held on the message queue of the objects. These information are used to provide assistance and explanation during the activities of software development. Thus, users can, at any moment, ask for Assist/DS the following questions:

- (What can I do?) – To answer this inquire, it is exhibited the state of the behavioural knowledge of the objects responsible for the stage in which the user is.

- (What was done?) – To answer this inquire, it is exhibited the messages replied by the objects, kept in their knowledge-base.

- (What will be done?) – To answer this inquire, it is exhibited the messages kept in the message queue of the objects.

The Assist/DS provides graphical and text editors to be used in software development process. These editors are implemented with autonomous objects also. The suitable editors for each activity are identified in the project planning phase. The Assist/DS Coordinator may, in runtime, include new editors or modify the characteristics of the existing editors. The graphical and text editors are
implemented in the classes *Techniques* and *Document*, respectively (recall to class hierarchy illustrated in Fig. 5). The inclusion of techniques and documents is permitted through specialization of these classes, by methods of the *Coordination* class, and by acquisition knowledge mechanism of the *autonomous objects*.

6 Conclusions

An architecture for the construction of intelligent SEE is provided. This structure is composed by a class hierarchy, structured by software development life cycle. These classes describes *autonomous objects* that represent stages, phases, activities, and tools involved on software development process. The *autonomous objects* constitute an intelligent computational unit that controls their behaviour, possesses the knowledge necessary to task's fulfillment, acquires knowledge, and communicate with other objects and with users.

Benefits of this approach includes better implementation, maintenance, and evolution of the SEE. These facilities are provided by knowledge structuring of the OOP, which allows the knowledge to be separated in independent units, and by the behaviour knowledge representation, which is independent to procedural knowledge. This strategy reduces procedures with complex structure of control, or class with control function. In the Assist/DS the knowledge relative to an activity or technique is located on particular class. The integration within these components is provided through the class hierarchy and by behaviour knowledge of *autonomous objects*. The inclusion of an activity, technique, and document, consists of the class specialization, and the inclusion of the behaviour knowledge of the included element.

Some limitations of the object model proposed have been observed. They are related to the knowledge-acquisition mechanism of the *autonomous objects*, and the lack of mechanisms for concurrency control and distribution. These aspects are currently under investigation.
The prototype was developed only with the basic structure of functioning. The aim was to investigate the feasibility of this structure. Now we are incrementally implementing the others facilities designed for this assistant, such as graphical editors and documents.

One of the benefits provided by this object model is to cater for the construction of intelligent systems, in which the knowledge and the intelligence is located in each entity involved. We hope that the contribution of this research will be a identification of an effective alternative for construction of intelligent SEE.

Acknowledgments
The authors are in indebted to several persons that contribute to this paper, specially Dr. Tom Price (UFRGS-Brazil), that conducted the first steps of this research.

References


782