

An Aspect-Based Object-Oriented Model for Multi-Agent Systems

Alessandro F. Garcia

Carlos J. P. de Lucena

*Software Engineering Laboratory (LES) – TecComm Group
Computer Science Department – PUC-Rio – Brazil
e-mail:{afgarcia, lucena}@inf.puc-rio.br*

Abstract

This paper proposes an agent model from early stage of design that (i) describes structured integration of agents into the object model, (ii) incorporates flexible facilities to build different types of software agents, (iii) encourages the separation of agency properties and capabilities, (iv) provides explicit support for disciplined and transparent composition of agency properties and collaborative capabilities in complex software agents, and (v) allows the production of agent-based software so that it is easy to understand, maintain and reuse. Our model explores the benefits of aspect-based design and programming for the incorporation of agents in object-oriented systems. We also demonstrate our multi-agent approach through the Portalware system, a web-based environment for the development of e-commerce portals.

1. Introduction

Agent technology has been revisited as a complementary approach to the object paradigm, and has been applied in a wide range of realistic application domains, including e-commerce, human-computer interfaces, telecommunications, and concurrent engineering. Software agents, like objects, include a specific set of capabilities for their users. In fact, objects and agents have many similarities [2, 17], but the state of agents is composed of beliefs, goals, plans, and their behavior is driven by a number of agency properties such as autonomy, adaptation, interaction, learning, mobility and collaboration. Moreover, collaborative software agents must incorporate distinct collaborative capabilities to cooperate with other agents in heterogeneous contexts. In practice, a complex application is composed of objects and multiple types of agents, each of them having different agency properties and collaborative capabilities. The introduction of software agents in the object model pose other problems because many properties and capabilities of agents are intrusive and not orthogonal, so that a disciplined approach is required for composition. As a consequence, there is a need for an agent model which supports the manipulation of these properties and capabilities directly since the very beginning of design in order to master the natural complexity of building agent-based object-oriented systems.

Most existing object-oriented software architectures typically incorporate agent models which focus on one type of agent, and do not provide direct support for handling and reusing properties and capabilities separately (e.g. [4], and [14]). The state and behavior of an agent in current proposals generally are encapsulated as an object. Even though it is desirable for an agent to appear as a single object, this approach results in agent design and implementation being quite poor, complex and difficult to understand, maintain and reuse in practice. In addition, it is not often easy to design software agents properly, as the developers of multi-agent systems have to take into account many agency properties at the same time. Ideally, agent system developers should apply special structuring techniques and disciplined ways of associating the different properties and collaborative capabilities of an agent with its core state and behavior. In this sense, Kendall et al. [10] proposes the layered agent architectural pattern, which separates different layers of an agent, such as sensory layer, collaboration layer, and so on. However, this proposal causes object shizophrenia – the agent state and behavior, which are intended to appear as a single object, are actually distributed over multiple objects. In addition, we believe agent design evolution is cumbersome since it is not trivial adding or removing any of these layers; it requires invasive adaptation of the adjacent layers.

In this paper, we present an agent model for object-oriented systems. Our model explores the benefits of aspect design and programming [12] for the incorporation of agents in object-oriented systems. Aspect-oriented design encourages modular descriptions of software systems by providing support for cleanly separating the object's core functionality from its crosscutting concerns. Aspects are the units that modularize these crosscutting concerns and are associated with one or more objects. They are comprised of

pointcuts, advices, and introduction. Join points are principal points in the dynamic execution of an object; pointcuts are collections of join points; advice is a special method-like construct that can be attached to pointcuts; introduction is a construct that defines new ordinary member declarations to the object(s) to which the aspect is attached (such as, attributes and methods). Central to the process of composing aspects and objects is the concept of weaver. Weaver is the component responsible for deviating the normal control flow to an advice, when program execution point is at a join point. AspectJ [13] is a practical aspect-oriented extension to the Java programming language. AspectJ implements a weaver and provides support for programmers defining aspects.

In this context, our aspect-based agent model provides separation of concerns among the different agency properties and collaborative capabilities. The root Agent class specifies the core state and behavior of an agent, and different types of agents are organized hierarchically as subclasses that derive from the root class. Each Agent object is associated with a given number of aspects, and each of these associated aspects introduces the appropriate behavior for an agent’s property or collaborative capability. Our model allows the composition of these agent aspects in a disciplined and non-intrusive manner. As a consequence, we believe it supports the construction of multi-agent software with improved structuring for evolution and reuse of design and implementation solutions. We will also present results applying our model to introduce multiple aspects of agents in Portalware [7], a web-based environment for the development of e-commerce portals.

The remainder of this paper is organized as follows. Section 2 gives provides brief definitions and applications of multi-agent systems. This section also introduces an example which is used throughout this paper to illustrate our approach. Section 3 presents our aspect-based model for designing agent-based applications. Section 4 describes some implementation issues. Section 5 discusses related work. Finally, Section 6 presents some concluding remarks and directions for future work.

2. Multi-Agent Systems: Definitions and Case Study

Software agents are often viewed as complex objects “with an attitude” [3], in the sense of being objects with some additional agency properties. In general, the state of an agent is formalized by knowledge, and is expressed by mental components such as *beliefs*, *goals*, and *plans* [17, 20]. Beliefs model the external environment with which an agent interacts. A goal may be realized through different plans. Software agents carry out plans to achieve their internal goals, and the selection of plans is based on their beliefs. Plans select agent’s appropriate capabilities that could achieve their stated goal(s). There are different kinds of plans, and they are application-specific [10]. The behavior of an agent depends on and is affected by the incorporated *agency properties*. Table 1 summarizes the definitions for the main agency properties. Since such properties are difficult to define precisely in a broad context, we have stated and refined them for the agent domain based on our experience in developing multi-agent systems [7, 19, 21] and previous studies [10, 16, 17].

AGENCY PROPERTY	DEFINITION
Interaction	An agent communicates with the environment and other agents by means of sensors and effectors
Adaptation	An agent should rapidly adapt/modify its state and behavior according to new environmental conditions
Autonomy	An agent is capable of acting without direct external intervention; it has its own control thread and can accept or refuse a request
Learning	An agent can learn based on previous experience while reacting and interacting with its external environment
Mobility	An agent is able to transport itself from one environment in a network to another in order to achieve its goals
Collaboration	An agent can cooperate and negotiate with other agents in order to achieve its goals and the system’s goals

Table 1. An Overview of Agency Properties

There is no agreement on what actually constitutes *agenthood*, that is, how the state and behavior of an agent is described, and what are the fundamental properties of agents that must be incorporated. However, an agent is generally regarded to be an *autonomous* entity that interacts with its *environment*, and adapts its state and behavior based on this interaction [17]. Therefore, autonomy, interaction and adaptation can be considered as

fundamental properties of software agents, while learning, mobility and collaboration are neither a necessary nor sufficient condition for agenthood. There are several kinds of software agents, including information agents, user agents, interface agents, and mobile agents. Each agent type has different application-specific capabilities and agency properties.

Case Study. Figure 1 illustrates the software agents in Portalware [7], a web-based environment for the construction and management of e-commerce portals. Portalware encompasses three agent types: (i) interface agents, (ii) information agents, and (iii) user agents. Each of them has different capabilities and properties, but everyone implements the fundamental aspects defined by agenthood. Figure 1 summarizes capabilities and agency properties for the Portalware agents. For a more complete discussion about this example the reader can refer to [6].

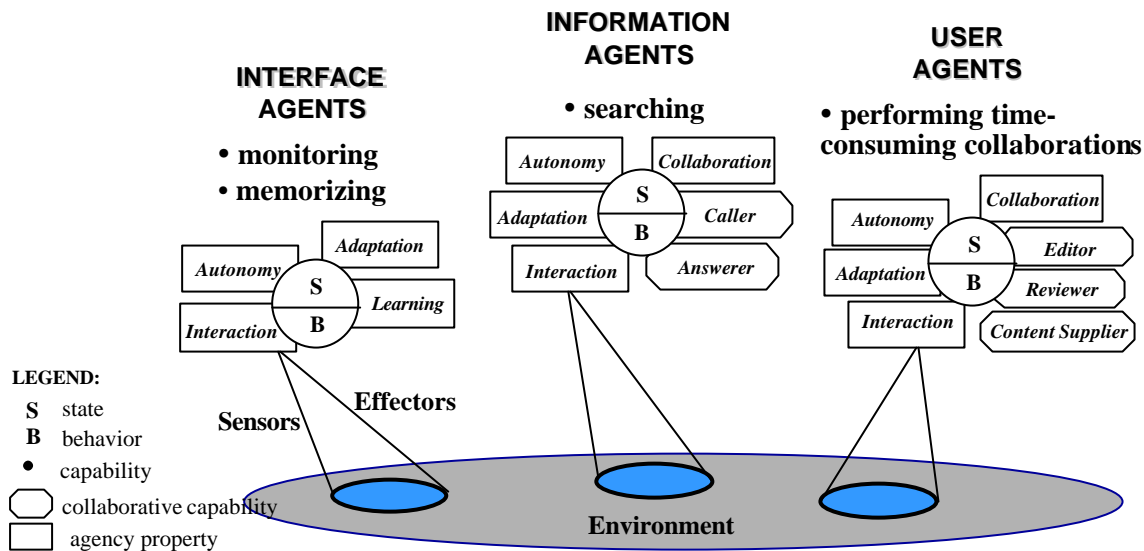


Figure 1. Portalware Agents.

3. An Aspect-Based OO Model for Multi-Agent Systems

In the following, our agent design model is presented as an aspect-oriented extension of the traditional object model. In particular, the proposed agent model is discussed in terms of: (i) *agent's core state and behavior*, (ii) *agent types*, (iii) *agency aspects for agenthood*, (iv) *particular agency aspects*, (v) *collaborative aspects*, and (vi) *agent evolution*. We adopt UML diagrams [1] as the modeling language throughout this paper. The design notation for aspects is based on [11].

Agent's Core State and Behavior. The Agent class specifies the core state and behavior of an agent, and should be instantiated in order to create the application's agents. Since an agent is described in terms of its goals, beliefs, and plans, the attributes of an Agent object should hold references to objects that represent these elements, namely Belief, Goal and Plan objects. Methods of the Agent class are used to update these attributes and implement agent's capabilities.

Agent Types. Our model proposes the use of inheritance in order to create different agent types. Different types of agents are organized hierarchically as subclasses that derive from the root Agent class. The methods of these subclasses implement the capabilities of each agent type. Figure 3 illustrates the subclasses representing the different kinds of agents of our case study (Section 2). For example, the method search(keyword) of the InformationAgent class implements the capability of information agents searching for information according to a specified keyword.

Agency Aspects for Agenthood. Aspects are used to implement the agency properties an agent incorporates. Each agency aspect is responsible for providing the appropriate behavior for an agent's property. An agency aspect introduces an interface related to the agent's property, and implements the advices that crosscuts the core agent's functionality. Figure 2 depicts the agent aspects which define essential agency properties for agenthood: interaction, adaptation, and autonomy. For example, when the Interaction aspect is associated with the Agent class, it makes any Agent instance interactive. In other words, the Interaction aspect extends the Agent class's behavior to send and receive messages. This aspect updates messages and senses changes in the environment by means of sensors and effectors. Since the process of sending and receiving messages occurs quite often in multi-agent systems and cuts across the agent's basic capabilities, the implementation

of this process as an aspect is a design decision that avoids code duplication and improves reuse. The Adaptation aspect makes an Agent object adaptive, it adapts an agent's state and behavior according to new environmental conditions. The AdaptBeliefs(), AdaptGoal(), AdaptPlan() advices are responsible for updating beliefs, goals, and plans, respectively. The Autonomy aspect makes an Agent object autonomous, it encapsulates and manages one or more independent threads of control, implements the acceptance or refusal of a capability request and for acting without direct external intervention. The advice MakesDecision() implements the decision-making process by invoking specified decision plans. The advice PerformsPlan() implements the ability to accept or refuse a capability request.

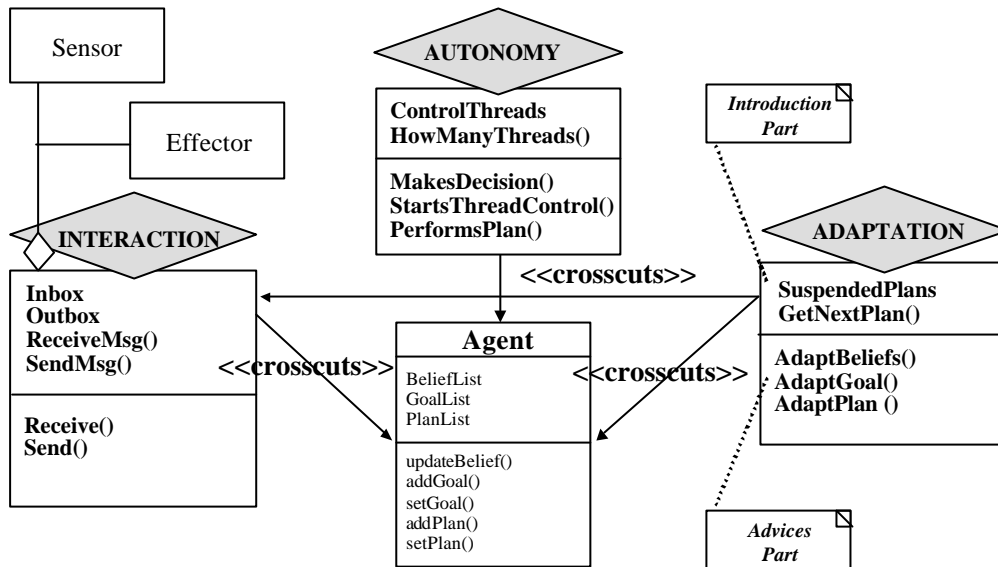


Figure 2. Agency Aspects and the Design for Agenthood.

Particular Agency Aspects. The agency aspects that are specific to each agent type are associated with the corresponding subclasses (Figure 3). Note that the different types of software agents inherit the agency aspects attached to the Agent superclass. As a consequence, the information agent inherits agenthood features and only defines its specific capabilities and aspects.

Collaborative Aspects. Aspects are also used to implement the collaborative capabilities of an agent. A collaborative aspect is a part of an agent which defines the activity of the agent within a set of particular collaborations. As a result, it decouples the agent's basic capabilities from the collaborative capabilities, which in turn improves understanding, reusability and evolution. Since an Agent object needs to include multiple collaborative capabilities, different collaborative agency aspects are associated with this object. Figure 3 pictures the application of our design model for the Portalware collaborative agents.

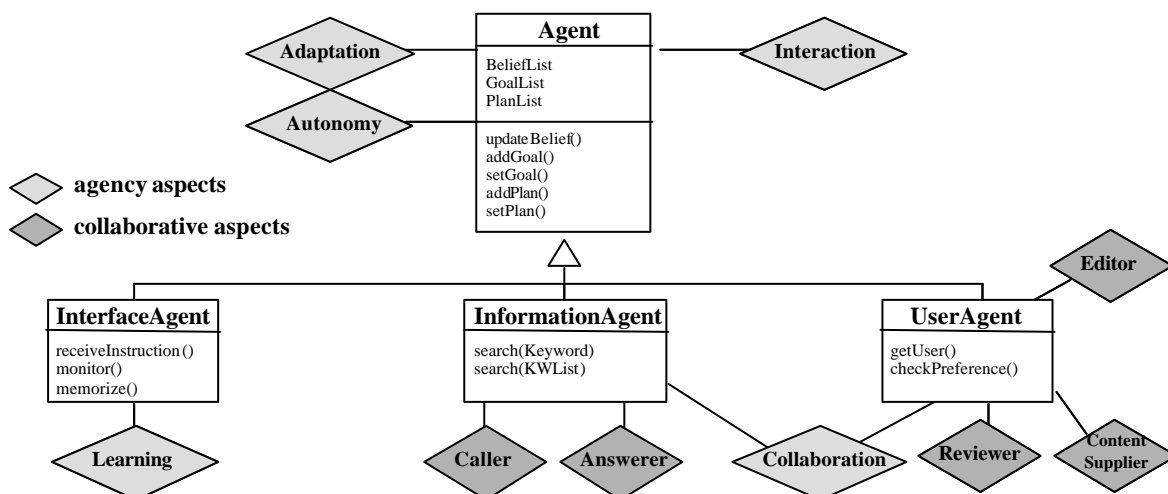


Figure 3. Portalware Agents and their Agency Aspects.

Agent Evolution. The behavior of software agents can evolve frequently to meet new non-functional requirements. Suppose information agents do not need to cooperate with each other in order to find information. Instead, information agents are required to transport themselves from one environment in the network to another in order to achieve the searching goal. As a consequence, they do not need to have the caller and answerer capabilities, but must to be mobile. In our model, this modification is done transparently, since agency aspects can be added to or removed from classes in a plug-and-play way. The Caller and Answerer aspects are disattached from the InformationAgent class without requiring any intrusive modification in other components of the agent. The remaining behavior of the agent is kept. However, it is necessary to associate the Mobility aspect, which introduces to the Agent class the ability to roam the network and gather information on behalf of its owner (Figure 13). This association process includes defining the end of executions of searching methods (search(*)) as a pointcut. At runtime, when the execution of a search() method is finished, the weaver deviates the program control flow to the Mobility aspect. It takes results of searching methods to check if the information agent was not able to find the information. Thus, if the method result is null, this aspect is responsible for migrating the agent to the other host in order to start a new search for the information.

4. Implementation Issues

Our aspect-based agent model was implemented for the case study (Section 2) using AspectJ [13]. The implementation consists of 91 classes, 5 agency aspects, and 5 collaborative aspects. In order to implement the dependency relationship between these different aspects, we used the “dominates” construct of AspectJ. In addition, we have used two aspects which implement application’s general aspects: exception handling and persistence. For instance, the persistence aspect is applied for dealing with the persistence of an agent’s state. In this sense, we have also used TSpaces/IBM [15], a blackboard architecture for network communication with database capabilities. TSpaces provides group communication services, database services and event notification services. It is implemented in the Java programming language and thus it automatically possesses network ubiquity through platform independence, as well as standard type representation for all datatypes. In our prototype, messages captured by different collaborations were implemented as tuples and finally, agents communicated by posting such tuples to the blackboard.

In our system, different instances of information agent can have varying characteristics. They may be collaborative or non-collaborative, they may be static or mobile, they may or may not learn. So, it is desirable to build personalized information agents to attach different aspects to distinct instances of information agents. Our proposed model supports this feature. However, the current version of AspectJ does not provide direct support for associating different aspects with distinct class instances. This feature is currently supported by some meta-object protocols such as Guaraná [18].

5. Related Work

Kendall et al [9] describes the application of aspect-oriented programming to implement role models. This approach is used to represent the different collaborative capabilities (roles) an agent can have. In fact, we have followed their guidelines to implement agent’s collaborative aspects. However, their work does not deal with agency properties, which we believe are the main source of agent complexity. In this sense, our paper presents a unified framework for dealing with collaborative capabilities and agency properties, and their interrelationships.

Research in aspect-oriented software engineering has concentrated on the implementation phase. A few works have presented aspect-based design solutions. In addition, since aspect-oriented programming is still in its infancy, little experience with employing this paradigm is currently available. To date, aspect-oriented programming has been used to implement generic aspects such as persistence, error detection/handling, logging, tracing, caching, and synchronization. However, each of these papers is generally dedicated to only one of these generic aspects. In this work, we provide an aspect-based design model which: (i) handles both agency-specific aspects as well as generic aspects (e.g. persistence), and (ii) encompasses a number of different aspects and their relationships.

6. Conclusions and Future Work

The main contribution of this work is a design model which provides a unified framework for introducing agency aspects to the object model. Moreover, this agent model: (i) incorporates flexible facilities to build

different kinds of software agents, (ii) encourages the handling of each of the agency aspects separately, (iii) provides explicit support for disciplined and transparent composition of non-functional properties and collaborative capabilities in complex software agents, and (iv) allows the production of agent-based software so that it is easy to understand, maintain and reuse.

Design patterns [5] are important vehicles for constructing high-quality software. Architectural patterns define the basic structure of an architecture and of systems which implement that architecture; design patterns are more problem-oriented than architectural patterns, and are applied in later design stages. As presented in this paper, aspect-based design can be used to address the problems of dealing with aspects of agents. We are currently investigating a language of aspect-based design patterns for agent systems, which provide good design solutions for dealing with each of the aspects of agents. An architectural pattern will be proposed in order to specify a high-level description of the agent's organization in terms of its aspects and their interrelationships. Aspect-based design patterns can be used to provide solutions for each of the agent aspects while following the overall structure of the proposed architectural pattern.

References

- [1] G. Booch, J. Rumbaugh. “**Unified Modeling Language – User Guide**”. Addison-Wesley, 1999.
- [2] J. Bradshaw et al. “**KaoS: Toward an Industrial-Strength Generic Agent Architecture**”. In J. M. Bradshaw (Ed.), *Software Agents*. Cambridge, MA: AAAI/MIT Press, 1996.
- [3] J. Bradshaw. “**An Introduction to Software Agents**”. In: *Software Agents*, J. Bradshaw (ed.), American Association for Artificial Intelligence/MIT Press, 1997.
- [4] D. Brugali, K. Sycara. “**A Model for Reusable Agent Systems**”. In: *Implementing Application Frameworks – Object-Oriented Frameworks at Work*, M. Fayad et al. (editors), John Wiley & Sons, 1999.
- [5] E. Gamma et al. “**Design Patterns: Elements of Reusable Object-Oriented Software**”. Addison-Wesley, Reading, MA, 1995.
- [6] A. Garcia, C. Lucena, D. Cowan. “**Agents in Object-Oriented Software Engineering**”. Technical Report CS-2001-07, Computer Science Department, University of Waterloo, Waterloo, Canada, March 2001.
- [7] A. Garcia, M. Cortés, C. Lucena. “**A Web Environment for the Development and Maintenance of ECommerce Portals Based on Groupware Approach**”. Proc. of the IRMA'2001, Canada, May 2001.
- [8] IBM Research: Subject-Oriented Programming Group. “**Subject-Oriented Programming and Design Patterns**”. Available in: <http://www.ibm.research/sop>.
- [9] E. Kendall. “**Agent Roles and Aspects**”. ECOOP Workshop on Aspect Oriented Programming, July, 1998.
- [10] E. Kendall et al. “**A Framework for Agent Systems**”. In: *Implementing Application Frameworks – Object-Oriented Frameworks at Work*, M. Fayad et al. (editors), John Wiley & Sons, 1999.
- [11] M. Kersten, G. Murphy. “**Atlas: A Case Study in Building a Web-Based Learning Environment using Aspect-Oriented Programming**”. Proceedings of the OOPSLA'99, Denver, USA, ACM Press, pp. 340-352, 1999.
- [12] G. Kiczales et al. “**Aspect-Oriented Programming**”. In Proceedings of the ECOOP'97, Finland, Springer-Verlag LNCS 1241, June 1997.
- [13] G. Kiczales et al. “**An Overview of AspectJ**”. Submitted to ECOOP'2001, Budapest, 2001.
- [14] D. Lange, M. Oshima. “**Programming and Developing Java Mobile Agents with Aglets**.” Addison-Wesley, August 1998.
- [15] T. Lehman, S. McLaughry, P. Wyckoff. “**TSpaces: The Next Wave**”. Hawaii International Conference on System Sciences (HICSS-32), January 1999.
- [16] H. Nwana. “**Software Agents: An Overview**”. Knowledge Engineering Review, 11(3): 1-40, September 1996.
- [17] Object Management Group – Agent Platform Special Interest Group. “**Agent Technology – Green Paper**”. Version 1.0, September 2000.
- [18] A. Oliva, L. Buzato. “**The Design and Implementation of Guaraná**”. 5th USENIX Conference on Object-Oriented Technologies and Systems (COOTS '99), San Diego, USA, May 1999.
- [19] P. Ripper, M. Fontoura, A. Neto, C. Lucena. “**V-Market: A Framework for eCommerce Agent Systems**.” World Wide Web, Baltzer Science Publishers, 3(1), 2000.
- [20] Y. Shoham. “**Agent-Oriented Programming**”. Artificial Intelligence, 60(1993): 24-29, 1993.
- [21] TecComm Group. “**Frameworks and New Technologies to E-Commerce**”. URL: www.teccomm.les.inf.puc-rio.br