Ongoing Research on the Software Engineering of Multi-Agent Systems

Carlos Lucena
Computer Science Department, PUC-Rio
Rua Marquês de S. Vicente 225, Rio de Janeiro RJ, Brazil 22451-900, + 55 21 3114-1500
e-mail: lucena@inf.puc-rio.br

Abstract

Multi-agent systems (MAS) are composed of heterogeneous agents with distinct agent properties, such as adaptation, mobility, collaboration, roles, and learning. The agent paradigm per se is not new since it has its roots in the Artificial Intelligent research. However, an important issue in getting the agent technology into mainstream software development is the creation of appropriate software engineering techniques. This paper presents the ongoing research on Software Engineering for Multi-Agent Systems at the Software Engineering Lab (LES) at PUC-Rio.

1. Introduction

Advances in networking technology have revitalized the investigation of the agent technology as a promising paradigm to engineer complex distributed software systems. Agent-based computing is rapidly emerging as a powerful technology for the development of complex software systems, synthesizing contributions from many different research areas including artificial intelligence and software engineering. Nowadays, the agent technology has been applied in a wide range of application domains, including e-commerce, human-computer interfaces, telecommunications, and concurrent engineering. Multi-agent systems (MAS) and their underlying theories provide a more natural support for ensuring important properties such as autonomy, mobility, environment heterogeneity, organization, openness, and intelligence.

A software agent is driven by goals, knowledge, and a number of behavioral properties such as autonomy, adaptation, interaction, collaboration, learning and mobility. MAS features are now being applied to the development of large industrial software systems. Such systems involve hundreds, perhaps thousands of agents, and there is a pressing need for software engineering techniques that allow their complexity to be effectively managed, and principled methods are also required to guide the process of MAS development. Without adequate development techniques and methods, such systems will not be sufficiently dependable, trustable, extensible, difficult to comprehend, and their components will not be reusable.

The complexity associated with MAS is not straightforward and involves numerous facets and dimensions. As the multiple software agents become highly collaborative, new problems appear. It makes their coordination and management more difficult and increases the probability of existing exceptional situations, security holes, privacy violations, unexpected global effects, and so on. Moreover, since users and software engineers delegate more autonomy to their MAS, and put more trust in their results, new concerns arise in real-life applications. Commercial success for MAS applications will require scalable solutions based on software engineering approaches in order to enable reuse and effective deployment. However, many existing agent-oriented solutions are far from ideal;
in practice, the systems are often built in an ad-hoc manner and are error-prone, not scalable, not dynamic, and are not generally applicable to large-scale MAS. This short version paper, based on a keynote presentation delivered at the SBES 2004, addresses ongoing research on software engineering of multi-agent systems at the Software Engineering Lab (LES) at PUC-Rio. In our research we seek to develop a systematic understanding of the nature of agent-based software engineering. We start with the basic premise that an important issue in getting the technology into the mainstream of software development is the development of appropriate methodologies and architectures for engineering agent-oriented software. Throughout our investigation we make every effort to explore the use of agents as first order entities.

2. Software Agents

An agent is an interactive computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives [11]. Some characteristics of software agents include:

- **Autonomy**: an agent is capable of acting without direct external intervention
- **Interactivity**: an agent communicates with the environment and other agents
- **Adaptation**: ability to respond to other agents and/or its environment to some degree
- **Proactivity**: an agent is goal-oriented, i.e. it does not simply react to the environment
- **Learning**: ability to modify its behavior based on its experience
- **Rational**: ability to choose an action based on internal goals
- **Mobility**: ability to transport itself from one environment to another

The multi-agent paradigm offers a decomposition model, an abstraction for software development based on the problem domain, a model of system development as a composition of organizational units, and a decentralization model to reduce the unit coupling.

3. The Research on SE-MAS at LES

The ongoing research on software engineering for MAS at LES is investigating several aspects of the agent paradigm. These include the definition of conceptual frameworks, the development of modeling languages, the relationship between agents and other paradigms, such as objects and aspects, and the support for design and implementation of multi-agent systems. In this talk we will present some of the results of this research.

3.1. The TAO Conceptual Meta-Model

The TAO meta-model [8] provides an ontology to capture the foundations for agent- and object-based software engineering. Below, we introduce the static and dynamic aspects of the meta-model.

3.1.1. TAO Static Aspects. The static aspect of the TAO meta-model captures a rich variety of concepts involved in the structure of a multi-agent system namely, object, agent, organization, object role, agent role, environment and event.

- **Object**: An object is a passive element in the domain whose instances have state and behavior. An object may evolve from state to state. However, it has no control over its
behavior, meaning that it does whatever any other element asks it to do and only when it is asked.

Agent. An agent is an autonomous, adaptive and interactive element in the domain whose instances are expressed through mental components such as beliefs, goals, plans and actions. An agent acts as a processor for plans of actions that are executed, using its beliefs, to accomplish a goal. A goal is an objective the system should meet. And a belief is some knowledge about the system.

Organization. An organization is an element in the domain whose instances group agents, objects and other organizations (sub-organizations). An organization has goals, beliefs (as agents) and axioms. An axiom characterizes global constraints that agents must obey. An organization also is responsible for defining the roles that will be played by agents, sub-organizations and objects. At least one organization must inhabit the environment. We call this organization main-organization. The main-organization is the unique organization that does not play any role since it is not defined as a member of any other organization.

Object Role. An object role guides and restricts the behavior of an object through the description of a set of features that are viewed by other elements. An object role may restrict access to the state and behavior of an object instance. But it may also add information, behavior and relationships to the object instance that plays the role.

Agent Role. An agent role guides and restricts the behavior of an agent through the description of a set of goals, beliefs and actions. An agent role defines duties, rights and protocols that restrict an agent instance. A duty defines an action that must be executed by an agent instance; a right defines an action that may be executed by an agent instance; and a protocol defines an interaction between an agent role and other elements.

Environment. An environment is an element in the domain whose instances are the habitat for agents, objects and organizations. An environment can be passive as an object, or active as an agent.

The TAO defines relationships in which the above concepts may be involved. These are:

Inhabit. Specifies that the element that inhabits is created and destroyed in the habitat and may leave and enter habitats, respecting the habitat permissions. Inhabit is applied to environments and agents, environments and objects and environments and organizations.

Ownership. Specifies that an element is defined in and must obey a set of constraints defined by another element. The member element does not exist outside of the scope of its owner. Ownership is applied to roles (members) and organizations (owners).

Play. Specifies that an element playing a role assumes its properties and relationships. The behavior of the element is guided by and restricted to the scope of the role. Every agent or sub-organization plays at least one role in an organization. Objects also can play roles.

Specialization (Inheritance). Defines that a sub-element that specializes a super-element inherits all the state and behavior associated with the super-element. A sub-element also may add and redefine the properties and behavior associated with the super-element. Specialization may be used between objects, agents, organizations, object roles and agent roles.

Control. Defines that a controlled element must do anything that a controller element requests. An agent role can control another agent role or an object role. Object roles only can control another object role.

Dependency. Defines that an element (client) may be defined to be dependent upon another one (supplier) to do its job. In other words, the client cannot completely do its job
unless it asks the supplier. An agent role can depend on another agent role and an object role can depend on another object role.

- **Association.** Defines how one element interacts with another, indicating that these elements know each other. Associations may be used between (i) roles (object or agent roles), (ii) environments, (iii) objects, (iv) agents and objects and (iv) organization and objects.

- **Aggregation.** Defines that an element is part of an aggregator. The aggregator may use the functionality available in its parts. This relationship may be applied between object roles, agent roles, objects, agents and organizations.

### 3.1.2. TAO Dynamic Aspects

The dynamic aspects of the TAO meta-model describe the relationships between its static elements. They can be classified as primitive (elementary) dynamic processes and high-level dynamic processes.

Primitive processes describe the most basic domain-independent interactions that exist between elements. The processes of creating and destroying the elements of MAS are characterized as primitive processes. These processes define the actors, preconditions, execution steps and post conditions involved. They encompass processes for object, object role, agent, agent role, organization and environment creation and destruction.

High-level dynamic processes are more complex domain-independent behavior that are described based on primitive and other high-level dynamic processes. They derive from the characteristics of the relationships between entities that are associated with domain-independent behavior: ownership, play and inhabit relationships. They encompass processes for an agent entering or leaving an organization, an organization entering or leaving another organization, and an agent or an organization entering or leaving an environment.

### 3.2. The ANote

The ANote modeling language [1, 2] was developed to offer a standard way to describe concepts related to the multi-agent system modeling process. ANote’s basic purpose is to provide users with an expressive visual modeling language to develop and exchange meaningful agent models.

ANote offers an agent-oriented structuring of the software, which means that, in ANote, the agent is the main model element. The agent is the basic structural unit that encapsulates behavior through decisions and plans. Each agent in a multi-agent system contains properties that allow it to accomplish its goals.

ANote provides a set of models for multi-agent system specification since no single model is sufficient. Every non-trivial, large-scale, system is best approached through a small set of nearly independent models or views. A view is a partial specification that provides a certain abstraction of a system aspect under consideration. It enables the software designer to concentrate on a single set of properties each time, as he only needs to write one view at a time. In addition, it enables him to consider only those features that are important for a particular context. The ANote views are:

- **Goal View:** specifies the system goals. A goal defines a service or functionality that some user expects to get from the system.

- **Agent View:** specifies the agent types that exist in a multi-agent application solution and their relationships, thus defining the system structural base.
– **Ontology View**: identifies the non-agent components of the system, and it defines the world (i.e. environment) where the agents will execute to achieve their goals. This view provides a structural description of the agent environment resources.

– **Scenario View**: captures agent behavior in specific contexts. The main model element in this view is the scenario. A scenario is a description denoting similar parts of possible agent behaviors limited to a context, i.e., to a purposeful state where actions and interactions take place among two or several agents.

– **Planning View**: specifies the execution states, or actions, an agent has to perform to compute an action plan.

– **Interaction View**: used to represent the set of messages the agents exchange while computing an action plan.

– **Organizational View**: defines the structure of a multi-agent system. This view specifies the system organizations and their relationships.

### 3.3. The MAS-ML

MAS-ML [10] is a modeling language that makes additive extensions to UML to provide support for multi-agent systems modeling. By augmenting the UML meta-model, new modeling capabilities were incorporated into UML, contributing to its evolution. The UML meta-model extensions proposed by the MAS-ML are based on the concepts defined in the TAO.

#### 3.3.1. MAS-ML Static Aspects

The definitions of the object and event concepts of the TAO meta-model are similar to those of the Class and Event UML meta-classes; thus their notation is preserved in MAS-ML. However, it is necessary to create new meta-class definitions for the other TAO concepts. MAS-ML has included the AgentClass, OrganizationClass, EnvironmentClass, ObjectRoleClass and AgentRoleClass meta-classes to the UML meta-model. In addition, some stereotypes and meta-classes were created extending the UML meta-model to represent goals, beliefs, plans, actions, axioms and messages sent and received by agents.

MAS-ML extends the Class diagram to include modeling information about agents. This diagram shows the association, aggregation and the specialization relationships of TAO. Moreover, MAS-ML defines two other diagrams: Organization and Role diagrams.

The Organization diagram models the system organizations identifying their habitats, the roles that they define and the elements – objects, agents and sub-organizations – that play those roles. This diagram shows the ownership, play and inhabits relationships of TAO. The Role diagram is responsible for clarifying the relationships between the agent roles and object roles. This diagram shows the control, dependency, association, aggregation and specialization relationships of TAO.

#### 3.3.2. MAS-ML Dynamic Aspects

MAS-ML proposes an extension to the UML Sequence diagram to model the dynamic aspects based on the TAO meta-model. MAS-ML extended Sequence diagram has three new elements to represent the agent, the organization and the environment concepts. Furthermore, it proposes new ways to define pathnames that identify element instances. Sequence diagrams illustrate (i) agents and organizations committing to roles and changing their roles, (ii) agents and organizations sending and receiving messages, (iii) agents and organizations executing actions, (iv) elements calling methods of an object, (v) objects executing methods and (vi) the creation and destruction of elements.
To model the dynamic processes presented above, MAS-ML extended Sequence diagram defines new stereotypes and extends the definition of the existing UML create and destroy stereotypes. The new stereotypes are [10]: role_commitment, role_cancel, role_change, role_activate and role_deactivate.

3.4. The Aspect-Oriented Agent Architecture

The aspect-oriented architecture [3, 4, 5, 9] is based on the assumption that an agent is an object with added features. The notion of architectural aspects is used to enrich objects with agent properties in a transparent way. Objects are then considered the basic abstraction for building the agent kernel. However, since a single agent is a much richer abstraction than an object, our architectural approach emphasizes the definition of architectural aspects for representing the different agent properties.

The software architecture provides components for handling each crosscutting agent concern as an individual aspect. The components of our aspect-oriented agent architecture represent agenthood and additional concerns. Agenthood concerns comprise interaction, adaptation and autonomy. Additional concerns include mobility, collaboration, roles, and learning.

The aspect-oriented architecture is composed of two kinds of architectural components: (i) the Kernel component that modularizes the agent's basic concerns or the intrinsic knowledge, and (ii) aspectual components that separate the crosscutting agent concerns from each other and from the Kernel component.

The Kernel component is the agent component that implements the services provided for the agent's clients. This component is responsible for modularizing the elements of the agent's intrinsic knowledge, such as actions, plans, goals, and beliefs. This component realizes an interface that makes available services implemented by the agent. The Kernel component should be implemented as a set of classes at the detailed design level. This component can alternatively represent an existing object, which needs to be transformed into an agent.

The aspectual components, or architectural aspects, represent aspects at the architectural level. Aspectual components are used to improve the separation of crosscutting concerns in agent architectures. They modularize the agenthood properties – interaction, adaptation, and autonomy – and the additional properties – mobility, collaboration, and learning. These agent properties are isolated from the agent kernel and from each other in order to facilitate their composition and the construction of different agent types and heterogeneous agent architectures. An aspectual component is refined as a set of aspects and auxiliary classes, which are also part of the crosscutting concern at the detailed design level.

3.5. Further Research

The research reported here is still in progress. Much remains to be done to create and validate software engineering techniques for MAS. Other ongoing research, developed by the same group, include the development of a generative approach for MAS development [6], law governing for agent interaction, the development of agent architectures (based on components [7] and on the MAS-ML notation), tool support for system development (based on the ANote and MAS-ML notations), and the development of case studies in order to build a body of experience.
4. Conclusion

Agent-based systems are likely to provide new approaches to dealing with the complexity of developing and maintaining modern software. Nevertheless, developing robust large-scale agent-based systems will require new software engineering approaches. Currently there are many methods and techniques for working with individual agents or systems composed of only a few agents. Real large-scale multi-agent systems are still at their infancy and new software engineering techniques need to be developed to cope with the complexity of large MAS.

In this paper we have described some very recent research results on multi-agent systems modeling, design and implementation that have been produced at LES. A new generation of researchers at LES is currently following up on the research results reported here.

Acknowledgements

This paper summarizes the contributions of three PhD Theses, by Alessandro Garcia, Ricardo Choren and Viviane Silva, developed in our laboratory under the author’s supervision. The work reported here was partially supported by the ESSMA Project under grant 552068/2002-0 (CNPq, Brazil).

References

