Reuse in Agent-based Application Development

Rosario Girardi
Department of Computer Science, Federal University of Maranhão,
Av. Dos Portugueses s/n - Campus do Bacanga, São Luis - MA , Brazil
Phone: +55(98)9967-1908
Email: rgirardi@deinf.ufma.br

ABSTRACT

The agent computational model has the potential to improve the theory and practice of modeling, designing and implementing software applications. Agent-based Software Engineering should consider advances and also lessons learned in software reuse, particularly, the requirement of effective reusable software abstractions. This work proposes a model for developing reusable software using the agent paradigm. Reusable software abstractions generated through Agent-based Application Engineering are analyzed considering both their abstraction level and domain dependence.

Categories and Subject Descriptors
D.2 [Software] {Software Engineering}

General Terms
Design

Keywords
Software Reuse; Agent-based Development; Agent Software Abstraction.

1. INTRODUCTION

Software reuse techniques have contributed for significant improvements to reduce both time and cost of software development, and the object-oriented paradigm has offered the mechanisms to spread their practice.

In spite of these contributions, the complexity of software applications is constantly increasing and appropriate techniques for the development of distributed, open and adaptable software systems are needed.

The agent computational model [4, 14, 17] has the potential to improve the theory and practice of modeling, designing and implementing software applications. Also, it is considered the best approach for the development of distributed systems of great complexity, mainly because an agent is a useful software abstraction to the understanding, engineering and use of this kind of systems [16, 36].

An active research on the systematization of methods and methodologies for agent-oriented development is being conducted [6, 7, 15, 23, 27, 28, 32, 33]. However, most development techniques approach the development of specific software applications. Agent-based Software Engineering should consider advances and also lessons learned in software reuse, particularly, the requirement of effective reusable software abstractions [10, 11].

This article proposes a model to exploit reuse in agent-oriented development and analyzes reusable software abstractions in Agent-based Application Engineering. Section 2 briefly reviews main concepts of reusable software abstractions. Section 3 introduces a first model for Application Engineering and Application Development. Generated products in the development phases of requirement engineering, global and detailed design are analyzed considering their abstraction level and domain dependence.

2. SOFTWARE ABSTRACTIONS

All approaches to software reuse use some form of abstraction for software artifacts. A software abstraction is a high-level, succinct, natural and useful specification that corresponds to one or more realizations in a more detailed level of representation. The specification describes what the abstraction does and the realization how that is done [20]. Abstractions play a central role in reuse activities. Concise abstractions are needed to efficiently locate, understand, compare, and select the appropriate software artifacts from a collection. Specialization of a generalized software artifact corresponds to choosing an abstraction realization from the variable part of an abstraction specification. Finally, to compose a set of software artifacts, users must understand the artifact interface, which must be clearly described in the abstraction specification.

An abstraction is composed of a fixed, a variable and a hidden part. Only the fixed and variable part are visible in the abstraction specification. The fixed and variable parts represent, respectively, the invariant and variable features of the abstraction. The hiding part, not visible in the abstraction specification, describes details of a realization.

Considering software productivity, the effectiveness of a reuse technique can be measured in terms of the cognitive distance: the intellectual effort that is required to take a software system from one stage of development to another. Moreover, a reuse technique should reduce the cognitive distance between the conception of a system at the highest level and its final executable implementation.

This can be accomplished by both moving the abstraction specifications closer to the abstractions used to reason about the application domain, and partially or fully automating the mapping from abstraction specifications to executable abstraction realizations [10].
3. MULTI-AGENT APPLICATION ENGINEERING

MaAE ("Multi-Agent Application Engineering") is a research project approaching software complexity and productivity through the construction of techniques and tools promoting software reuse in Multi-Agent Software Engineering.

We are working on a model for Multi-Agent Application Engineering. Application Engineering is the process of creating reusable software abstractions, and Application Development, the one of constructing an specific application using reusable software abstractions [12].

We expect to construct a software development environment composed by a set of development tools and libraries of high-level reusable software abstractions for agent-based application development. Experiments are being conducted on the information access application domain.

Sections 3.1 and 3.2 introduce a first model for Agent-based Application Engineering and Agent-based Application Development. In both models, development is approached through successive refinements from abstract to concrete agent software abstractions.

3.1 Developing multi-agent specific applications

In MaAE, a multi-agent specific application is obtained through the composition of a set of reusable agent frameworks available in the library of the development environment (Figure 1). These frameworks are realizations of high-level software abstractions in the library.

3.2 Developing high-level software abstractions

Figure 2 illustrates a first model for Application Engineering in MaAE, showing their activities and generated products. The reusable agent-based software abstractions are illustrated and described considering their abstraction level (from abstract to concrete) and their dependence level from the application or user domain (from domain dependent to domain independent): domain models, user models, agent-oriented architectural styles, agent-oriented design patterns, agent-oriented frameworks and software agents.

The mapping from specifications to realizations to select appropriate frameworks from domain and user models is conditioned by the design decision of using a particular style of agent architecture.

3.2.1 Requirement Analysis

Requirement Analysis in Agent-based Application Engineering produces the requirement specification of a family of similar systems in an application domain. The specification should include both the required functionality of the domain and features of end users as well. A domain, a user model and the interactions between them constitute the reusable software products generated by the Requirement analysis phase of Agent-based Application Engineering.

The domain model - application domain dependent and specified at a high level of abstraction - represents the formulation of a problem, knowledge or activity of the real world. The formulation is generic enough to represent a family of similar systems. The representation language of the domain model establishes the vocabulary and semantics for the elements, processes and relationships in the systems.

Figure 1. A model for Agent-based Application Development

Particular requirements of a multi-agent specific application are used to map the specification level of a domain model into a realization satisfying such needs. The realization should have associated a set of frameworks, which are agent-based solutions to those requirements. Requirement analysis in Agent-based Application Development should also consider particular preferences of users of the multi-agent specific application. Therefore, these user profiles are used to map the specification level of a user model into a realization satisfying these preferences or user needs. The realization should have associated a set of frameworks, which are agent-based solutions to those needs.
A user model specifies features, needs, preferences and goals of end users. Both the number and variety of information systems users is increasing the practice of user-centered software development for the construction of more effective interactive applications adaptable to particular users [9, 19].

3.2.2 Design
The design phase of Agent-based Application Engineering produces a reusable design specification for a family of similar systems in an application domain. The design specification is composed by a set of agent-based architectural styles, software patterns and frameworks.

Software architects use a number of commonly recognized “styles” to guide their design of system structures. An architectural style is a set of design rules that identify the kinds of components and connectors that may be used to compose a system or subsystem, with local or global constraints indicating the way the composition is done [26]. Software architectural styles are domain independent and are specified at a high level of abstraction.

Multi-agent software applications have used basically the following architectural styles [13, 20, 31]:

- **blackboard**, where agents communicate indirectly through a blackboard;
- **message passing**, where agents communicate directly through message passing;
- **federative**, where agents are grouped into federations and a facilitator agent in each federation manages send and received messages out of the federation;
- **master-slave** coordination, where working agents are coordinated by a manager which distributes tasks and controls their results;
- **market** coordination, where agents are classified as providers and consumers trying to maximize profits through negotiation.

A software pattern is recurring problem and solution; it may address conceptual, architectural or design problems, associated to architectural styles, domain and user models. A pattern is described in a set format to ease its dissemination. The format states the problem addressed by the pattern and the forces acting on it. There is also a context that must be present for the pattern to be valid, a statement of the solution, and any known uses. Software patterns are abstract entities either dependent or independent of the application domain. Agent-based patterns have already been proposed approaching agent-based designs both at architectural and detailed levels [2, 18].

Both agent patterns and architectural styles should provide the main design guidelines for building agent-oriented frameworks [3, 5, 22]. Agents in the framework should be selected from a bank of software agents. This bank contains reusable software agents offering functionality either dependent or independent from an application domain.

3.2.3 Agent component designs
We are considering four types of solutions to design individual agent components [25]:

- **reactive** agents (simple reflexive agents and reflexive agents with internal state), and
- **deliberative** agents (goal-based agents and utility-based agents)

Deliberative agents have a symbolic reasoning model. They have a plan to be pursued and negotiate with other agents to achieve their goals.

Reactive or reflexive agents do not have a reasoning model. They act as a response to an environment stimulus. Even though reactive agents do not use traditional artificial intelligent representations, they can exhibit some kind of intelligent behavior through the interaction with other agents in a multi-agent system.

A simple reflex agent works by finding in its memory, a rule whose condition matches the current situation (as defined and interpreted by the current percept) and then doing the action associated with that rule. The agent memory is then composed of a set of condition-action rules (also called production rules or situation-action rules) of the form “if <condition> then <action>”. Humans also have this type of behavior, some of which are learned responses (such as braking when a car in front is braking) and some of which are innate reflexes (such as blinking when something approaches the eye).

A reflexive agent with internal state also works by finding a rule whose condition matches the current situation and then doing the action associated with that rule. However, the current situation is defined by the current percept and an internal state stored in the agent memory. This internal state is updated with two kinds of knowledge: information about how the world evolves independently of the agent, and information about how the agent’s own actions affect the world.

In goal-based agents, the agent memory has goal information, which describes situations that are desirable and the decision making of what action will be executed will depend both on the current situation and on the fact that the action will allow to achieve the goal. Sometimes this will be simple, when goal satisfaction results immediately from a single action; sometimes, it will be more arduous, when the agent has to consider long sequences of twists and turns to find a way to achieve the goal. Search and planning are sub-fields of Artificial Intelligence giving solutions to finding action sequences that do achieve the agent’s goals.

Utility-based agents are similar to goal-based agents. However, goals alone are not enough to generate high-quality behavior. There are many action sequences that will allow the agent to achieve its goal, but some can be, for instance, quicker, safer, more reliable, or cheaper than others. Utility provides a way in which the likelihood of success can be weighed up against the importance of the goals. It is a function that maps a state onto a real number, which describes the associated degree of utility. When there are conflicting goals, only some of which can be achieved (for example, speed and safety), the utility function specifies the appropriate trade-off.

4. CONCLUDING REMARKS AND FURTHER WORK
We have described a new approach for reuse-based software engineering using the agent paradigm.

An initial development model has been described for Agent-based Application Engineering and Agent-based Application Development. The model specifies main activities to be performed (requirement engineering, global and detailed design) and generated high level agent-based software abstractions (domain and user models, architectural styles, design patterns, frameworks and agent designs).
Our final goal is to elaborate an effective model for Agent-based Application Engineering composed of a methodology and supporting tools for developing high-level software abstractions.

Further work is being done in two main complementary directions. First, we are analyzing and experimenting with current approaches for agent-based software engineering in the domain of information access [1, 27, 30]. This work will give us knowledge and experience on the development of multi-agent specific applications in these applications domains. Second, we are working on techniques for the acquisition, specification and implementation of high-level agent-based software abstractions (domains and users models, patterns and frameworks) [8, 21, 24], and on mechanisms for mapping abstraction specifications to realizations in all abstraction levels of the proposed model. For that, both compositional and generative approaches are being explored.

5. ACKNOWLEDGMENTS

This work is supported by CNPq, an institution of Brazilian Government for scientific and technologic development.

6. REFERENCES


