MAGE: An Agent-Oriented Software Engineering Environment

Zhongzhi Shi, Haijun Zhang, Yong Cheng
Yuncheng Jiang, Qiujuan Sheng, Zhikung Zhao
(Key Laboratory of Intelligent Information Processing, Institute of Computing Technology,
Chinese Academy of Sciences, Beijing 100080, China)

Abstract: Agent-oriented programming is an important technology for autonomic computing and grid computing. Advances in agent technology depend on improving frameworks for building and supporting agent societies. Experience suggests that first generation multi-agent systems focus only on some aspects and fall short of providing a rapid prototyping development environment for the systematic analyze, design, construction and deployment of agent-oriented applications. This paper presents MAGE, an agent-oriented programming environment, with complete tools to support agent-oriented requirement analysis, design, development and deployment. It also gives some comparison between MAGE and other multi-agent platforms with particular focus on methodology. At last, this paper gives some applications on MAGE.

Key words: MAGE, multi-agent systems, agent-oriented programming, software engineering, methodologies

1. INTRODUCTION

Expansions and performance improvements in hardware, data storage facilities, and telecommunication has led to ever more ambitious software engineering projects, which are approaching the point where they are so complex that they are unmanageable. Traditional software engineering development techniques based on functional analysis and data flow, or on object-oriented modeling, are in many cases not sufficient, in themselves, to capture needed dynamism and flexibility of some of the current development tasks that are undertaken. Researchers are now seeking new methods and approaches that can help software engineers grapple with some of these problems. One of the new approaches that have been proposed is agent-oriented software engineering (AOSE).

Recently, natural evolution of Multi-Agent Systems (MAS) has led them to migrate from the research laboratories to the software industry. This migration is good news for the entire MAS community, but it also leads to new expectations and new questions about multi-agent system methodologies, tools, platforms, reuse, specification, and so on. This introduction of Software Engineering techniques into Multi-Agent Systems gains more and more interest in both the research area and the industry. The best example is the brunch of announcements about new multi-agent platforms, such as OAA, RETSINA and etc.

However, experience suggests that first generation multi-agent systems focus only on some aspects of MAS and AOSE, and thus fall short of providing a rapid prototyping development environment for the systematic analyze, design, construction and deployment of agent-oriented applications.

So we feel that there is an urgent need for multi-agent frameworks to cover the entire AOSE process and to provide systemic and powerful tools for analysis, design, implementation and deployment of agent-based applications. This philosophy led to the development of MAGE.

The breakdown of the rest of this paper is as follows: Section 2 briefly presents MAGE introduction and its architecture; Section 3 describes MAGE toolkit; in Section 4, MAGE will be compared with some other MAS frameworks; Some applications on MAGE will be given in section 5; a short conclusion and future work will be given in Section 6.

2. Overview of MAGE

2.1 MAGE introduction

MAGE (Multi-AGent Environment) is a multi-agent environment with a collection of tools supporting the entire process of agent-oriented software engineering and programming. It is designed to facilitate the rapid design and development of new multi-agent applications by abstracting into a toolkit the common principles and components underlying many multi-agent systems. The idea was to create a relatively general purpose and customizable toolkit that could be used by software users with only basic competence in agent technology to analyze, design, implement and deploy multi-agent systems.

This project is supported by High-Tech Programme 863 (2001AA113121, 2003AA115220) and National Natural Science Foundation of China (90104021).
MAGE toolkit has the following features:

- The toolkit utilize standardized technology and standardized specification wherever feasible, for examples our MAGE is FIPA compliant, and agent interaction is based on FIPA-ACL.
- The toolkit provide agent developers with all the domain-independent functionality so they need only implement the domain-specific problem solving abilities of the agents they want to define.
- The toolkit provide friendly and easy-to-use human-computer interface through visual programming paradigm and pick-and-choose principle.
- The toolkit support an open and flexible design so that it is easily to extend.
- The toolkit support the entire development process of agent-based software engineering and programming: AUMP (Agent Unified Modeling Platform) for system modeling, VAgentStudio (Visual Agent Studio) for implementation, and MDeployer for deployment and management.

MAGE has been used by a number of companies and academy groups, such as Legend Corporation. Further details and documentation can be found at [http://www.intsci.ac.cn/en/research/mage.html](http://www.intsci.ac.cn/en/research/mage.html).

2.2 MAGE architecture

Multi-Agent Systems are complex software. Building such complex system software requires using adequate engineering methods. Traditionally, software engineering distinguishes three stages of construction: Analysis, Design and Development. But for systems like Multi-Agent Systems this is not sufficient, and describing the entire MAS creation process, from early design to a running application, a fourth stage after the development stage must be distinguished, that is known as deployment [12].

We propose to use the following definitions for these four stages which are shown in Figure 1:

- Analysis: the process of developing an understanding of the system and its structure.
- Design: the aim of a “classical” design process is to transform the abstract models derived during the analysis stage into models at a sufficiently low level of abstraction that they can be easily implemented. This is not the case with agent-oriented design, however. Rather, the aim of agent-based design is to transform the analysis models into a sufficiently low level of abstraction that traditional design techniques (including object-oriented techniques) may be applied in order to implement agents [17].

- Development: the process of constructing a functional solution to the problem. Practically it consists of coding the solution with a particular programming language. Here we divide development phase into three steps: building behaviour, building agent and building system.

- Deployment: the process of actualizing the solution to the real problem in the given domain and managing the runtime environment.

The two former stages are more involved with methodology, and the two latter addresses more technical aspects.

2.3 Deployment architecture

MAGE is designed to be compliant with FIPA, and MAGE framework in which MAGE agents exist and operate is comply with FIPA Agent Management Specification. Figure 2 is the MAGE deployment architecture, it is a logical reference model for the creation, registration, location, communication, migration and retirement of agents [5].

![Figure 2. Deployment Architecture](http://example.com/figure2.png)

From Figure 2 we can see that MAGE platform consists of the following components, each representing a capability set.

**Agent Management System (AMS)** is a mandatory component of MAGE. The AMS exerts supervisory control over access to and use of MAGE platform. The AMS offers white pages services to other agents.

**Directory Facilitator (DF)** is a mandatory component of MAGE. The DF provides yellow pages services to other agents. Agents may register their
services with the DF or query the DF to find out what services are offered by other agents.

**Message Transport Service (MTS)** is the default communication method between agents on different agent platforms.

**Agent** is the fundamental actor in MAGE which combines one or more service capabilities into a unified and integrated execution model that may include access to external software, human users and communications facilities.

**Software** describes all non-agent, executable collections of instructions accessible through an agent.

Moreover, two auxiliary modules are provided to support designing agent systems: Agent Library and Function Component.

### 2.4 Agent architecture

When defining an agent architecture, there is a tradeoff between flexibility and capability. Some times, users hope the agent model is powerful enough to handle things such as co-operation, negotiation and mobility; but other times, they may hope to customize agents with their own defined components, such as their own co-operation engine. To solve this problem, we define an extensible agent architecture, with a kernel and some plug-in like components. Figure 3 is the extensible architecture of a generic MAGE agent which shows our philosophy.

![Agent Architecture](image)

**Figure 3. Agent Architecture**

Agent kernel consists of the following parts. **Sensor** perceives the outside world. **Function Module Interface** makes an effect to the outside world. **Communicator** handles communications between the agent and other agents. **Co-Ordinator** makes decisions concerning the agent’s goals, and it is also responsible for co-ordinating the agent interactions with other agents using given co-ordination protocols and strategies, e.g. the contract net protocol or different kinds of auction protocol. **Scheduler** plans the agent’s tasks based on decisions taken by the Co-ordination Engine and the resources and task specifications available to the agent. **Resource Database** maintains a list of resources that are owned by and available to the agent. **Task Database** provides logical descriptions of tasks known to the agent. **Plug-In Manager** manages the components provided by MAGE or by users that can be plugged into agent kernel.

### 3. MAGE toolkit

As we just introduce, MAGE toolkit support the entire development process of agent-based software engineering and programming. This toolkit consists of three parts: modeling tool AUMP, development tool VAStudio, and MDeployer for deployment and runtime management.

#### 3.1 Modeling tool AUMP

Multi-agent systems are complex software. Building such complex systems software requires using adequate engineering methods. We divide agent-based system modeling into the following four steps:

1. **First step is system function modeling**, we use Use Case Model to describe system functions, then map Use Cases to goals which drive agents to implement system function.
2. **Second step is behaviour modeling** which is used to resolve how to achieve the Use Cases. There are six types of agent behaviour model: Activity Model, StateChart Model, Interaction Protocol Model, Plan Model, Inference Model and Reactive Rule Model.
3. **Third step is agent modeling** which is used to demonstrate the agent system structure. Here we use Agent Class Model to describe internal structure of agent class, use Organization Relationship Model to describe cooperation relationship between agents and use Ontology Model to describe knowledge structure.
4. **Last step is system deployment modeling**. We use Deployment model to describe how agents are distributed and how the system runs.

![An example of using AUMP](image)

**Figure 4. An example of using AUMP**
According to this agent-based system modeling method, we build AUMP. Figure 4 demonstrates an example which shows how AUMP works.

3.2 Development tool VAStudio

VAStudio is a visual and integrated development environment for developing MAS, and also it is an agent-based programming environment. It has the following features:
1. It provides plenty of behaviour components which can be used to build up agents.
2. It provides plenty of agent templates: Users can select an agent template and then add needed components to this template.
3. It provides many tools: such as behaviour editor, protocol editor, ontology editor and strategy editor. Behavior editor can help users to specify what an agent can do and how to do. Protocol editor is used to describe how this agent interacts with other agents. Strategy editor helps users to customize their negotiation strategies. Visual Multi-Agent Studio also provides plenty of behavior templates (such as Sequence Behavior, Cyclic Behavior, etc), protocol templates (such as English Auction protocol, Contract Net Protocol, etc), ontology templates (such as e-Commerce ontology, etc) and strategy templates, users can directly use these templates to develop their applications.
4. It can directly load the result of design process of AUMP and generate the corresponding agent or MAS.
5. Has a friendly and easy-to-use interface.

Figure 5 demonstrates the GUI of VAStudio.

3.3 MDeployer for deployment and runtime management

MDeployer is responsible for system deployment and management. It provides powerful functions for users to use, such as Start New Agent, Kill Agent, Send Message and etc. The most distinguished feature of MDeployer is that it provides a tool through which users can deploy and manage the whole system through web.

4. Comparison between MAGE and some multi-agent platforms

Here some multi-agent platforms are selected to compare with MAGE: AgentBuilder®, JackTM and Zeus. These platforms have in common:
- to be popular and regularly maintained (for bug fixes and extra features),
- to be grounded on well-known academic models,
- to be developed with industry-like quality standards,
- to cover as many aspects as possible of Multi-Agent Systems, including agent models, interaction, coordination, organisation, etc.,
- to be simple to set-up and to evaluate. This includes good documentation, download availability, simple installation procedure, and multi-platform support.

AgentBuilder is an integrated tool suite for constructing intelligent software agents. It is grounded on the AgentO and Placa BDI models. AgentBuilder’s documentation covers almost all the stages, from analysis to development. This is a good point, even if the analysis and design parts are quite succinct. They deal more with what to do rather than with how to do, which is more difficult to determine but is more helpful to the developer. Another good point are the software tools, which cover almost all the aspects of the stages, and establish links between them. The disadvantage of such a frame is that it limits the versatility of the tool.

Jack is described as an environment for building, running and integrating commercial JAVA-based multi-agent systems using a component-based approach. It is based on the BDI model dMARS developed at the Australian Artificial Intelligence Institute (AAII)[4]. Jack focuses mainly on the development stage. Analysis and design steps are only mentioned in [1]. Jack is particular by its strong agent-programming orientation[7]. This leads to a high versatility, agent’s architecture can range from simple Java-coded reactive behaviours to full BDI, using the provided architecture or another one. Unfortunately, the documentation provided is very technical, and doesn’t cover the methodological aspects, especially for the analysis and the design.

Zeus is an integrated environment for the rapid building of collaborative agents applications. The Zeus documentation is abundant, and puts a strong emphasis on the importance of the methodological aspect of Zeus ("The agent creation methodology is vital to the use of the Zeus toolkit"[5]). The main particularity of Zeus is its
complete integration of all the stages from design to deployment. It provides theoretical and practical tools, uses actual techniques of software engineering, and its methodological documents focus on the how to and not only on the what to do\(^3\). However, even if Zeus is modular, there is only one agent model supported, which limits the range of possible designs of multi-agent systems.

From the methodological point of view, we observe that there exist big differences between platforms, from the Zeus’s vital methodology, to Jack’s quasi-absence of methodological material and tools. Like Zeus, MAGE integrates all the stages from analysis to deployment, and has its own methodology. MAGE tools cover all the aspects of the stages. In analysis and design phases, Zeus only has documents, however besides documents, MAGE also has a graphic tool AUMP to help users to analyze and design system.

From the technical point of view, different solutions have been chosen: AgentBuilder and Zeus are more like “BDI editors”, whilst Jack is an agent programming language. Each kind of solution has its advantages and drawbacks. BDI editors are very efficient because they allow programming in a high level of abstraction, but are committed to a single agent architecture. On the other hand, agent programming languages are more versatile, but at the expense of more code writing, because the level of abstraction is lower. MAGE is a modular platform, and gives the programmer a rich library of agent models, where each model has its own dedicated editor. As introduced, VAStudio provides plenty of agent templates for users, and also provides many tools which are like “BDI editors” of Zeus. This implementation combines good versatility and high-level programming, but brings on some complexity.

For further and detailed comparison between these agent platforms, we give four qualities that seem relevant to us to all the stages, and which address as many aspects as possible of practical development pitfalls. These are:

- Completeness: The degree of coverage the platform provides for this stage.
- Applicability: the range of possibilities offered, the restrictions imposed by the proposed stage.
- Complexity: The difficulty to complete the stage.
- Reusability: The quantity of work gained by reusing previous works.

According to these four criteria, the comparison between MAGE, AgentBuilder, Jack and Zeus is listed in Table 1.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Agent Builder</th>
<th>Jack</th>
<th>Zeus</th>
<th>MAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td>★★★</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Complete*:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applicability:</td>
<td>★★★</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Complexity:</td>
<td>★★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Reusability:</td>
<td>★★★</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Design</td>
<td>Complete*:</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Applicability:</td>
<td>★★★</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Complexity:</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Reusability:</td>
<td>★★★</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Development</td>
<td>Complete*:</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Applicability:</td>
<td>★★★</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Complexity:</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Reusability:</td>
<td>★★★</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Deployment</td>
<td>Complete*:</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Applicability:</td>
<td>★★★</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Complexity:</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Reusability:</td>
<td>★★★</td>
<td>★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
</tbody>
</table>

Table 1. Comparison between AgentBuilder, Jack, Zeus and MAGE

In Table 1, for a given phase and a given criteria, the more “★” one agent platform has, the more advanced this agent platform is about the given criteria at the given phase; vice versa. For examples, at the analysis and design phases, Zeus has good document support, but has no tool support, so at these phrases the completeness of Zeus is basically ok, and Zeus has three “★” about the completeness. And MAGE at these phrases has good document support and good tool support, so MAGE has four “★” about the completeness at these phrases.
5. Applications

5.1 e-Business

We have developed a prototype of e-business. Figure 6 shows the architecture of agent-based e-business system. Four layers are constructed and each layer has its isolated functions.

The first layer includes all kinds of basic resources, such as database, models, and so on. Those resources are public and can be accessed by agent. For example, all data about the storage of e-business system are stored in a database, which is managed by a kind of DBMS such as Oracle or SQL Server. The database can be accessed by applications (agent) through standard SQL sentence.

The second layer is MAGE agent platform, which is the core of the system. It can be divided into two sub-layers, agent platform management and application services. Agent platform management is bases of the MAGE platform. It is in charge of the management of agent platform and management of all agents’ life cycle. AMS, DF and RMA are included in this sub-layer. Another sub layer is application services. It is constructed by many kinds of agents and each agent has its particular functions for given application, such as SearchAgent and BargainAgent. SearchAgent provides the service of database access. BargainAgent can employ different strategies to bargain with other buyer agent, so it can act as seller agent.

The third layer lies in Web server and it provides many kinds of Web application. It bases on the Web and establishes a friendly interface between users and application systems. In this layer, many services are provided through web pages and can be implemented by JavaBean, JSP, Servlet, etc..

When a user login this web, an interface agent will be generated corresponding to the user. Also the interface agent will register in MAGE platform if the user wants to search for some goods, the query information will be generated by the interface agent and be sent to the SearchAgent. After process the query, SearchAgent will send the search results to the interface agent. Finally, the user can get the results from web page which is generated by the interface agent.

Another interface agent will be generated if the user wants to buy something on the web. It acts as buyer agent. The user can setup the bargain strategies and models freely.

5.2 Parallel Web spider

Web information collection is a huge and complex work. An entity with intelligence, auto-adaptability and cooperating is needed in such kind of work. Since agent has the ability of autonomy, sociality, intelligence, auto-adaptability and collaboration, it is a good solution to employ agent to do such kind of work. Based on MAGE, we construct a distributed model for web information collection, i.e., Web Spider. Figure 7 shows the architecture of the Web Spider. It contains four kinds of agent: managing agent (DF), URL agent, spider agent and index agent.

---

Figure 6. e-Business system
DF is the center of management and communication in the system. It is a special agent that is used to manage the resources and other agents. So it acts as a management server. DF can create spider agents and also can stop them or kill them. URL agent is a management center for URL tasks. It takes charge of fetching, storing, distributing and coordinating URL tasks. On the one hand, URL agent fetches URLs from URL database into the task queue and distributes them to spider agents. On the other hand, URL agent receives new URLs from spider agents, then coordinates and saves them into URL database. Thus the URL database will be updated.

Spider agent starts its work after receiving URL tasks from URL agent. It fetches web pages according to URL and parses their hyperlinks. Then it sends these new URLs to URL agent and requests for new tasks. Many parallel and distributed spider agents can be employed in this system. The main job of index agent is classifying, indexing and saving web pages, which won’t be detailed in this paper.

In Figure. 7, the broken lines indicate the controlling streams between DF and other agents. The double-arrow lines indicate URL streams, including URL tasks and new URLs. The single-arrow lines indicate data streams of web pages. It will save these web pages into database after classifying and indexing.

6. Conclusions and future work

This paper presents MAGE, an agent-oriented programming environment, with complete tools to support agent-based requirement analysis, design, development and deployment. This paper also gives some comparison between MAGE and other platforms and gives some applications on MAGE in the end.

MAGE is a powerful development environment for autonomous computing. In terms of MAGE platform we are going to research on autonomic computing and agent-based grid in the future.

REFERENCES

[6] Steven P. Fonseca, Martin L. Griss and ReedLetsinger: Agent behavior architectures a MAS framework comparison. AAMAS 2002: 86-87


