Abstract

Agent-oriented process model, proven methods and appropriate tools for supporting the process and methods are major issues that software engineers and managers should decide about them in developing agent-based systems. Software engineers should select appropriate tools when faced with a set of various tools with different functionalities and characteristics. In this paper, based on the existing practices in software engineering tools assessment and our experiences in developing agent-based systems, we present and define major decision criteria for a tool selection framework in agent-based development. Developers can use this framework for evaluating and selecting appropriate tool(s) for doing their development tasks. These criteria can be used in developing a decision-making technique for tool selection in agent-based development.

1. Introduction

Software engineers should think about major issues such as quality, agent-oriented software development process, agent-oriented analysis and design methods, implementation methods, testing methods and agent tools in developing agent-based systems. Over the past few years a number of tools have been developed by research and commercial organization for developing agent-based systems [23]. These tools can be classified as agent-oriented analysis and design, agent frameworks, agent prototyping environments, agent languages, and agent communication infrastructures. Each of these tools has a specific role in agent-development and has its own characteristics and functionalities. Using these tools has advantages such as faster development, better quality, standardization of development process and easier documentation. Software engineers should decide about the appropriate tools that can help them in doing their development tasks.

This paper focuses on defining major criteria for a tool selection framework in agent-based development. Developers can use these criteria in their decision-making process for choosing the appropriate tool(s) for their development tasks. As an analysis of these decision criteria we will compare some agent frameworks by using this framework. The remainder of this paper discusses about agent-based software engineering, role of agent development tools, related works in tools assessment in software engineering and proposed major criteria for the proposed selection framework. Section 2 describes agent-based software engineering as a layered technology. Section 3 presents a classification of agent-based development tools. Section 4 presents the related works in CASE tools assessment for software engineering practices. Section 5 presents our decision criteria for a tool assessment framework for agent-based development. In this section by using the proposed criteria, some agent development frameworks are compared. Finally, in Section 6, we present the conclusion and further works.

2. Agent-based Software Engineering

Now an increasing number of problems in many complex, open and distributed application domains are being solved by agent-based solutions. The key abstraction in these solutions is the agent. An extended classification of these agent applications is presented in [2]. This classification shows the variety of agent applications and their characteristics in industrial, commercial, medical networking, entertainment and educational application domains. These software applications have common characteristics such as complexity, openness and distribution of data or control [25]. Also, many of them are real-time and safety-critical systems. So, they should be reliable and high quality systems. Thus, development, operation and support of agent-based systems need application of systematic,
disciplined and quantifiable approaches. It is necessary to develop software engineering techniques that are specially tailored to agents. So, we need agent-based software engineering. Agent-based software engineering is a powerful way of approaching large scale software engineering problems [42, 49].

Software engineering is a layered technology [37]. The layers of this technology are quality, process, methods and tools. Based on this view, we approach to the agent-based software engineering as a layered technology [2]. As shown in Figure 1, this technology encompasses software quality, agent-oriented software development process, agent-oriented methods and agent tools. So, for developing high quality agent-based systems, developers and managers should have software quality control and assurance techniques, agent-oriented software development process models [6,29,35], agent-oriented analysis and design methods [7,8,16,22,23,45,48], agent construction methods, agent languages [14,38,43,44], testing methods [8,13], agent-based system maintenance and tools for supporting agent-based processes and methods[23].

Also, in agent-based software engineering we need other activities for complementing the main activities and steps in the agent-based development. We propose these typical umbrella activities for applying throughout the software process [2]: agent-based software quality assurance, agent-based software configuration management, software measurement for agent-based systems, risk management for agent-based systems and agent-based software project management. In the next section, based on the layered approach to the agent-based software engineering view, we present a discussion about agent-based development tools.

### Figure 1. Layers of Agent-based Software Engineering

#### 3. Classification of Agent-based Software Engineering Tool

Agent-oriented software engineering tools provide support for the process and the methods. So, agent-based tools provide support for analysis, design, construction, test and maintenance of agent-based systems. Development tools are powerful ways to move a new technology into widespread use. The benefits and objectives in using agent tools are [3, 18, 37]: Increasing the design and development speed, reducing project costs, assisting developers in managing the complexities of agent-based systems, ease and improve the testing process, Improving integration of development activities via common methodologies such as Gaia [48], MaSE [46], improving the quality and completeness of agent-based system documentation, simplifying the system support, increasing the experience and thus popularize the agent technology.

In the past years a number of commercial and academic products have emerged to support agent-based development process and methods. We classify the existing agent tools in the following categories: Agent-oriented analysis and design, Agent Development frameworks, Testing and debugging and Agent prototyping tools. In the following sections, each tool category is described and examples from each category are mentioned.

#### 3.1. Agent-oriented Analysis and Design

These tools support agent-oriented analysis and design methods. By using these tools, analyst and designer can create models of the agent-based system to be built. Depending on the supported method, the models contain a representation of agent’s roles, responsibilities and goals, dependencies and relationships between the various roles, organizational structure and the knowledge and behaviors associated with a role or agent, agent classes and agents conversations. Also, in some cases such as “agentTool” [8], these tools provide code generation capability from the analysis and design models for other agent frameworks and communication infrastructures. There are some extensions to UML [34] for supporting the basic agent-oriented concepts in analysis and design of agent-based systems [33]. Also, researchers try to exploit an UML-based notation in these phases [5]. In these cases, we can use any off-the-shelf CASE tool supporting UML (e.g. Rational Rose) for managing analysis and design phase and there is a little need for any tool development.

#### 3.2. Agent Development Frameworks

An agent framework is a combination of the execution environment that supports the runtime agent code, and a set of classes and packages that allows the programmer to write agent code and agent-based systems [23]. Usually, a framework provides tools for agent creation, debugging and testing. The developers of such tools use names such as frameworks, platforms, toolkits and infrastructure for their products. Here, we use agent development framework for such tools. Usually, these tools provide classes that implement communication, reasoning and cooperation, providing a library of agent components, a
visual modeling environment for creating analysis and design models and modeling agent interactions, an agent building environment that includes an automatic agent code generator, some predefined agent models and tools to ease the development of systems, tools for testing and debugging created agents and facilities for interoperability with other agent systems through the use of a well-known agent communication language, such as KQML or following an agent standard such as FIPA. There are many academic and commercial agent frameworks. AgentBuilder, Bond, Decaf, dMars, JiVE, KAos, and ZEUS are some examples of such frameworks [15,23,31]. By using these tools, developers can devote more time to the agents and/or social aspects of the needed agent-based system.

3.3. Testing and Debugging

These tools support agent testing and debugging tasks and are mainly provided as a part of agent development frameworks. In agent-oriented programming, debugging and testing is mainly done at agent level. At this level there may be different kind of errors. Some of these errors relate to organizational relationships between agents, missing resources, bad scheduling and incorrect coordination and some of them relate to the agent’s internal task logic. As an example of debugging tools, in AgentBuilder the agent debugger allows the developer to examine the agent’s mental model. The developer can also step through the agent’s operation cycle-by-cycle and examine the mental model as the agent executes [38].

3.4. Agent Prototyping

These tools enable the software engineer to develop mock-ups of the agent-based system, allowing the designer and customer to gain insight into the function, operation and response prior to actual implementation. For example, Agent Factory is an agent prototyping environment [23].

Some of the above mentioned tools are point solutions and are used to assist in a particular agent-based activity (e.g. agent analysis modeling) and do not directly communicate with other tools. But, some of them produce output in a standard format that is compatible with other tools. In some cases, the tool builders have formed bridge between the tools. For example, the agent analysis and design tool such as agentTool [8] is coupled with Java code generation for an agent communication infrastructure named agentMom [9] and an agent framework named Carolina [39]. In the next section, the proposed criteria for our decision making framework for agent development tools are described.

4. Tool Assessment in Software Engineering

CASE tools offer a number of functions useful for the various activities of software engineering (e.g. analysis, design, implementation and testing). The evaluation and selection of CASE products is clearly a complex decision process which is fundamental to successful adoption of tools in organizations. Software practitioners in practice organizations have difficulty on assessing available CASE products and confronting them with the problems or goals in software development in their organization. Many organizations choose a CASE product without establishing a formal evaluation criteria and examining the candidate tools. Software engineers often do not know how to evaluate and select CASE tools.

Many different works have addressed the problem of CASE tools assessment through formal and informal frameworks. Proposals on how to perform systematic CASE tools assessment can be found in [19,24,26,27,28,30]. Each work includes a distinct set of criteria, which mainly is based on the personal view and experiences of the author. The criteria also vary in the level of objectivity and most works emphasize the assessment of technical properties of tools. Many of these works use categories of questions or rules as an assessment instrument. For example, Mosley [30] has designed six categories of questions to determine how well a tool does what it was intended to do. The assessment instrument is the generic set of 140 questions plus a tailored set of functional questions specific to the type of a tool being evaluated. Many questions addressed tools’ functionality and the quality of process and product support.

IEEE standard 1209-1992 represents an effort towards unification of CASE tools assessment criteria [19]. The suggested process is divided in two aspects, evaluation and selection. In this standard, the assessment criteria are identified in the evaluation process. These criteria are organized in an assessment hierarchy. At top level of this hierarchy there criteria such as Functionality and Reliability and at the next levels there are criteria such as Common Functions, Operating Environment, HW/SW Environment, Technology Environment, Methodology Support and Language Support. Based on this standard, Venturini propose a DSS for supporting the evaluation and selection of CASE tools [45].

5. Decision Criteria for Agent Tool Selection

As mentioned in the previous section, tools can be classified as agent-oriented analysis and design tools, frameworks for agent development, prototyping environments, agent languages (in which agents are modeled and written) and agent communication infrastructures. Each of these tools has a specific role and its own characteristics in agent-development. As is the case in software engineering field, the agent-development tool evaluation and selection is a complex process and the existing works in software engineering do not explicitly address agent tools. They do not consider issues such as
agent-oriented analysis and design methods, agent architectures and agent communication and coordination facilities in the tools. In this section, we present and describe major decision criteria for evaluating and selecting the existing agent development tools. These criteria have been selected according to the existing standards and practices for CASE tools assessment in software engineering [19,24,26,27,28,30] and also our experiences in evaluating and selecting the existing tools for developing different kind of agent applications in the AUT Intelligent Systems Lab [2]. These criteria can form a basis for developing a decision-making technique for evaluating and selecting the appropriate tool(s) for the agent-based system development tasks. We have developed a set of decision criteria for our framework. All these decision criteria in our selection framework are described in the following sections.

5.1. Agent Architecture

Agents are implemented by their agent architecture. Agent architectures establish the link between the agent and its role(s). Agent architecture provides the runtime environment for the role descriptions that make up the agent. Agent architectures are classified into reactive architectures, BDI architectures, and layered architectures [47]. Some development frameworks follow these architectures for implementing their agent architecture. Some agent frameworks such as Open Cybele [22] provide a basis so that the developer can implement the appropriate agent architecture for the agent application domain.

5.2. Reasoning Mechanism

The agents need some mechanisms for reasoning about their own and other agents’ problem solving capabilities and for coordinating their activities. The reasoning mechanisms in most agent-based systems are AI techniques such as rule-based inference, classical planning, various logic formalisms, and constraint satisfaction techniques. Some agent development frameworks such as JACK [1] and ZEUS [31] provide reasoning methods or mechanisms for modeling and developing the agent reasoning behavior. Others such as JADE [4] provide mechanisms for integration with a specific reasoning engine and use it as an agent-reasoning engine.

5.3. Agent Communication Language (ACL)

Development frameworks that support developing multi-agent systems should provide facilities for communication between agents. Most agent communication languages (ACLs) are based on “speech act theory” [40]. ACLs specify message types called performatives that are sent from one agent to another and cause some illocutionary actions in the receiving agent. There are standard agent communication languages. Among them, KQML [10] and FIPA ACL [13] are more important. Many frameworks support either KQML or FIPA ACL. Some frameworks combine ACL support with the ability to have user defined agent communication language.

5.4. Ontology Creation

When agents communicate with each other they should share a framework of knowledge in order to interpret the messages they exchange. This framework is an ontology that defines the application domain concepts being communicated between the agents. Agent development frameworks should have facilities such as ontology editors for creating the domain-specific ontologies for the agent-based system or importing existing general-purpose ontologies.

5.5. Coordination Constructs

In many multi-agent applications, agents need to a process to ensure that the agent society functions in a coherent manner. Agent uses the coordination techniques for reasoning about its actions and the actions of others in the society. There are a variety of coordination techniques including organizational structuring, contracting, multi-agent planning and negotiation. Some agent frameworks provide a coordination component that is responsible for coordinating the agent’s overall activities with other agents using its coordination techniques. These techniques are usually provided as a library of standard coordination constructs such as contract-net protocol. Also, it is possible to define framework specific coordination protocols for the agent-based system.

5.6. Supported Standards

In most application domains, agents that are developed by different organizations and developers should interact with each other. Thus, some form of standardization is needed to transform agent technologies into the products, applications or services that are widely deployed. Developing open, interoperable and scalable agent systems are main driving forces behind standardization for agent technology. Now, standardization is mainly done in consortiums and joint projects [32]. FIPA is a standards consortium that is currently working on specifications for agent [11]. These specifications are for Application, Architecture, Communication including an agent communication language, content languages, and interaction protocols, and Message transport. Some agent frameworks such as April, Comtec, FIPA-OS, Grasshopper, LEAP, JADE and ZEUS are examples of
FIPA compliant frameworks [12]. OMG is another consortium that has done some works in the areas of agent-related extensions to UML and ontology. Also, OMG has developed the MASIF standard for supporting interoperability between agents on different types of platforms. The standards that are developed by such consortiums are *de jure* standards [32]. Also, there are a number of *de facto* standards in this area. The most important example of these standards is KQML. It is an agent communication language that has been employed in many agent frameworks. These types of standards can be a result of joint projects on agent technology. Java Agent Services API, DAML and MESSAGE are representative examples of proposals from these projects for standardization in the agent area [32].

5.7. Language

By language, we mean the language used for implementing the tool and also the language used for producing agent programs. The former affects the platform on which the tool work and the later affects the choice of agent-based system computer platform and operating system. Java, C/C++, and Objective C are the most common languages that have been used in creating agent tools. Java, C/C++, Tool Command Language (Tcl), Scheme, Jess, Proprietary Agent Definition Languages (ADL) are agent programming languages that have been used in the existing agent frameworks for producing agent programs. Now, many tools have turned to Java to provide platform independence. Therefore, they are cross-platform tools and can work on any platform with a Java Virtual Machine (JVM).

5.8. Agent Definition Language

Agent development frameworks can provide a programming language with a higher level of abstraction for describing agent-based systems and creating agent programs. For example, JACK Agent Language is an agent language that is used in JACK [1]. It is a super-set of Java that encompasses the full Java syntax while extending it with constructs to represent agent-oriented features. Reticular Agent Definition Language (RADL) is another example of such languages [38]. It is an object oriented language that is provided by AgentBuilder for creating agent programs. In this language, the agents are defined as a set of rules. Usually, these frameworks have a compiler and run-time engine for compiling and executing the programs that are written in the supported agent definition language.

5.9. Implementation Support

Some development frameworks and agent analysis and design tools have code generation feature. These tools can generate code for the deployment of the agent-based system after basic design. Then, users can add their customs code to the agent programs for defining the desired behavior. Some agent frameworks directly deploy the agent-based system. In this manner, the user can not change the agent code.

5.10. Supported Process or Method

Tools provide support for the agent-oriented methods and/or an agent-development process. For example, agentTool supports the MaSE method for analysis and design of multi-agent systems [8,46] and JiVE supports multi-agent system design and development in JAFMAS [15]. Some frameworks such as ZEUS and AgentBuilder support their own agent-development process.

5.11 Visual Development Environment

Some tools provide visual environments for the agent-based system developers. By using these GUIs, developers can create different analysis and design models for their agent-based system, they can represent and visualize the agent systems and agent conversations, create and manage ontologies, define and describe the needed agents and their behavioral roles, select the coordination protocols, create agent programs and debugging and testing the produced agent system.

5.12. Integration with Legacy Software

Some agent tools have capabilities to link an external program or resource to an executing agent program. This feature can extend the functionality of the produced agent-based system.

5.13. Open source

The developers of an open source tool provide the source code of the software in addition to the executable version of it. The source code is free and the license does not restrict users from modifying and distributing it. Users can build their own executables by using standard development tools. This feature can minimize costs for educational institutes and small medium enterprises. Also, the tools can find a larger user community that can be engaged in the bug correction process and as a result tool quality is improved. But, the original tool developers should have strong control mechanisms for supporting their tools.

5.14. Documentation and Support

Like any professional software the agent tools need descriptive information that describes the use and operation of them. These documents can be manuals,
Some of the existing tools have good documentations in form of manuals and user-guides. But some of them are very weak in documentation and support and therefore learning and working with these tools is very difficult for their users.

5.15. Application Domains

Some frameworks produce agents that (because of their specific architecture) are suitable for a specific problem domain such as simulation, monitoring or wireless applications. Some of these tools do not provide agent mobility that is needed for some mobile agent applications. Because of technical issues, some agent frameworks are not suitable for producing large agent-based systems.

5.16. Cost

Some agent tools are for academic and research use and are free but some of them are commercial. The commercial products have an evaluation version that developers can try these versions before buying the tools.

We have compared four agent development frameworks according to the above mentioned criteria in Table 1. This table can be used as a basis for tool evaluation.

6. Conclusion

Now an increasing number of problems in industrial, commercial, medical networking, educational and entertainment application domains are solved by agent-based systems. These solutions are open, complex and distributed software systems and like any professional software should be engineered. In engineering these agent-based systems, software engineers should consider software quality, appropriate process, methods and tools. They need a set of tools that can help them to analyze and design the agent application, produce the agent-based system, test and evaluate the produced system and manage the changes in it.

In this paper, we defined a set of criteria for evaluation the agent-development tools. These criteria have been selected according to the existing works in tools assessment and our experiences in developing agent applications with the existing tools. Some of these criteria such as availability, open source, documentation and support are mainly business criteria. Other criteria such as supported agent architecture, reasoning mechanism, implemented agent communication language and coordination constructs are mainly functional criteria and are specific in developing agent-based systems. After determining process, analyzing the application nature and needs, choosing appropriate methods and considering other technical and business issues, software developers can use this framework for making decision about the appropriate tool(s) that can help them in doing their tasks in developing agent-based systems.

As a further work, we will develop a formal selection framework and a decision-making system for selecting tools for agent-based development. In this system, we weight the criteria according to developers’ requirements or according to the requirements concerning standard software development processes such the one proposed by ISO 9000. Then, according to the degrees of fulfillment and corresponding weights a total score can be calculated for ranking the tools. Also, based on these criteria we can develop a decision tree for guiding the developers in choosing their appropriate tool(s).

References

Table 1. Comparison of four agent-development frameworks according to our decision criteria

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Development Framework</td>
<td>Development</td>
<td>Development</td>
<td>Development</td>
<td>Development</td>
</tr>
<tr>
<td>Standard</td>
<td>None</td>
<td>None</td>
<td>FIPA</td>
<td>FIPA</td>
</tr>
<tr>
<td>Agent Architecture</td>
<td>Cognitive Mental</td>
<td>BDI</td>
<td>Mixed</td>
<td>Collaborative</td>
</tr>
<tr>
<td></td>
<td>Model-based</td>
<td>Architecture</td>
<td>reactive - deliberative</td>
<td>Agent</td>
</tr>
<tr>
<td>Reasoning Mechanism</td>
<td>Rule-based</td>
<td>Practical</td>
<td>Rule-based</td>
<td>Planning</td>
</tr>
<tr>
<td></td>
<td>reasoning, planning</td>
<td>reasoning, Rule-based</td>
<td>reasoning , planning</td>
<td>Planning</td>
</tr>
<tr>
<td>ACL Support</td>
<td>KQML</td>
<td>Send, reply</td>
<td>FIPA ACL</td>
<td>FIPA ACL</td>
</tr>
<tr>
<td>Ontology</td>
<td>None</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Implementation Support</td>
<td>Code generation</td>
<td>Code generation</td>
<td>Direct Deployment</td>
<td>Code generation</td>
</tr>
<tr>
<td>Agent Definition Language</td>
<td>Reticular Agent</td>
<td>JACK Agent</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Definition Language (RADL)</td>
<td>Agent Language (JAL)</td>
<td>Language</td>
<td></td>
</tr>
<tr>
<td>Visual Development Environment</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Coordination Constructs</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Testing and Debugging</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Legacy Software Integration</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Supported Method or Process</td>
<td>Reticular Agent</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Development Process</td>
<td>None</td>
<td>Language</td>
<td>Zeus Agent</td>
</tr>
<tr>
<td>Language</td>
<td>Java</td>
<td>Java</td>
<td>Java</td>
<td>Java</td>
</tr>
<tr>
<td>Application Domains</td>
<td>Information</td>
<td>Manufacturing,</td>
<td>E-commerce,</td>
<td>E-commerce,</td>
</tr>
<tr>
<td></td>
<td>Management, E-commerce,</td>
<td>Defense</td>
<td>Entertainment</td>
<td>resource/</td>
</tr>
<tr>
<td></td>
<td>E-commerce, Control and</td>
<td>Simulation,</td>
<td></td>
<td>process</td>
</tr>
<tr>
<td></td>
<td>Monitoring</td>
<td>Decision Support</td>
<td></td>
<td>management</td>
</tr>
<tr>
<td>Cost</td>
<td>95$ - 4995$</td>
<td>Commercial</td>
<td>Free</td>
<td>Free</td>
</tr>
<tr>
<td>Open Source</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Documentation</td>
<td>Excellent</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Excellent</td>
</tr>
<tr>
<td>Support</td>
<td>Web site</td>
<td>On-line TR, mailing list</td>
<td>Web site, active mailing list</td>
<td>Web site, mailing list</td>
</tr>
</tbody>
</table>