Achieving the Promise of Reuse with Agent Components

Martin L. Griss[1] and Robert R. Kessler[2]

[1] Computer Science Department 349BE, University of California, Santa Cruz
1156 High Street, Santa Cruz, CA 95064
griss@soe.ucsc.edu
http://www.soe.ucsc.edu/~griss
[2] University of Utah, School of Computing, 50 S. Central Campus Dr. #3190,
Salt Lake City, UT USA 84112
kessler@cs.utah.edu

Abstract. Using software agents as next generation flexible components and applying reuse technologies to rapidly construct agents and agent systems have great promise to improve application and system construction. Whether built on conventional distributed computing and application management platforms, on a specialized agent platform, on web service technology or within a P2P infrastructure, agents are a good match for independent development, for scalable and robust systems and dynamic evolution of features, and for autonomic self-managing systems. In this paper we describe the vision and progress we have made towards developing a robust infrastructure, methods, and tools for this goal.

1 Introduction

For some time now, component-based software engineering (CBSE) has promised, and indeed delivered, significant improvements in software development [1]. Greater reuse, improved agility and quality are accessible benefits. CBSE produces a set of reusable assets (usually components) that can be combined to obtain a high-level of reuse while developing members of a product-line or application family. Typically, one first performs domain analysis to understand and model commonality and variability in the domains underlying the product-line, and then a layered modular architecture is defined, specifying layers and core components, key subsystems and mechanisms. Finally, high-level specifications and interfaces are defined for pluggable or generated components. Implementation begins with the development or selection of a framework that implements one or more layers of the architecture. Delivering the reuse potential as a well-designed domain-specific kit carefully allocates variability to a combination of components, frameworks, problem-oriented languages, generators and custom tools [2].

Once this is done, components can be (largely) independently developed, or in closely related sets, doing detailed design and careful implementation of components and generator templates. Sometimes, when defects are to be repaired or new features added, it is a simple matter of enhancing a component or developing a new conforming component. However, at other times, the architecture has to be changed, new interfaces must be defined, and change ripples to many components.

Software agents offer great promise to build loosely-coupled, dynamically adaptive systems on increasingly pervasive message-based middleware, P2P and component technology, Java, XML, SOAP and HTTP[3]. Agents are specialized kinds of distributed components, offering greater flexibility than traditional components. There are many kinds of software agents, with differing characteristics such as mobility, autonomy, collaboration, persistence, and intelligence. Research in our group, previously at Hewlett-Packard Laboratories [4],[5], and now at UC Santa Cruz in collaboration with the University of Utah, is directed at the use of multi-agent systems in the engineering of complex, adaptive software systems. An important step is to simplify and improve the engineering and application of industrial-strength multi-agent systems and intelligent web-services to this problem. The research integrates several different areas, combining multi-agent systems, component-based software engineering, model-driven software reuse, web-services, and intelligent software.

In this paper, we will highlight some of the issues and our progress involved in making this step toward more robust, scalable and evolutionary systems using agent components and reuse techniques.

2 Multi-agent systems

Multi-agent based systems have several characteristics that support the development of flexible, evolving applications, such as those behind E-commerce and web-service applications. Agents can dynamically discover and compose services and mediate interactions. Agents can serve as delegates to handle routine affairs, monitor activities, set up contracts, execute business processes, and find the best services [6]. Agents can manage context- and location-aware notifications and pursue tasks. Agents can use the latest web-based technologies, such as Java, XML and HTTP, UDDI, SOAP and WSDL. These technologies are simple to use, ubiquitous, heterogeneous and platform neutral. XML will become the standard language for agent-oriented interaction, to encode exchanged messages, documents, invoices, orders, service descriptions and other information [7], [8]. HTTP, the dominant WWW protocol, provides many services, such as robust and scalable web servers, firewall access and levels of security.

An overview of agent capabilities from a large-scale AOSE/CBSE perspective can be found in books ([9],[10],[11]), papers ([6],[12],[13],[14],[15]) and web sites (http://agents.umbc.edu/ , http://www.hpl.hp.com/reuse/agents).

While there are many definitions of agents, many people agree that: "*an autonomous software agent is a component that interacts with its environment and with other agents on a user's behalf."* Some definitions emphasize one or another aspects such as mobility, collaboration, intelligence or flexible user interaction. Organizations such as FIPA (Foundation for Intelligent Physical Agents) are defining reference models mechanisms and agent communication language standards [16].

There are several different kinds of agent system [17]; our work at Hewlett-Packard Laboratories [4] focused on two types of agents:

- **Personal agents** interact directly with the user, presenting some "personality" or "social skills," perhaps as an anthropomorphic character, monitoring and adapting to the user's activities, learning the user's style and preferences, and automating or simplifying certain rote tasks. Examples include meeting scheduling agents, mail agents, software development, etc.
- **Collaborative agents** communicate and interact in groups, representing users, organizations and services. Multiple agents exchange messages to negotiate, share information, etc. Examples include online auctions, planning, negotiation, logistics and supply chain and telecom service provisioning.

In particular, in our prototyping we use personal agents as 24/7 user representatives to find, organize and interact with a set of collaborating team and service agents for meeting arrangers, e-commerce systems and email management [4],[5].

Many varieties of agent system and toolkits have been developed and described in the literature or on the web. Recently, even numerous Java-based, FIPA compliant systems are seeing use in the wider community. We have done most of our work using two Java-based toolkits: ZEUS [18] and JADE [19].

In the rest of the paper, we discuss agents as next-generation components and discuss some features and variants appropriate to the integration of reuse and agent technologies. We then discuss our research program and summarize the progress we have made to date, with an emphasis on our latest results in model-driven agent behavior choreography.

3 Agents as Next Generation Components

Multi-agent systems have a number of features that make them attractive for highly dynamic, evolving applications, in which a multiplicity of external systems, services, users, appliances and developers interact and change. These features are related to those of components, web services, workflow, and rule-based systems, but in combination provide a distinct software engineering capability. Agent systems are described with many different sets of features; the ones we find most compelling, and plan to exploit and extend in our research are discussed in detail in [17],[20],[21] where we provide a graphical model and discuss several of the characteristics of a typical agent system. As indicated, the more of each of these attributes, the more agent-like the component system becomes, and the more appropriate the use of an Agent-Oriented Software Engineering (AOSE) approach. These characteristics are supported by mechanisms and interfaces in the underlying infrastructure, as well as by models and policies configured in each agent or group of agents.

The key aspects that make agents suitable as next generation components [3], [17], include:

- **Loosely coupled, message-oriented.** The components conform to a message-oriented rather than method-oriented framework. As components, they have a dynamic component lifecycle; as agents, they can introspect on their own state, leading to some degree of self-management, and negotiate with other agent-components for services. The coupling is loose, similar to that obtained with a software bus; typically agents are not built assuming the existence of other specific components. Services are invoked by sending messages, and multiple agents can respond to those messages, or an appropriate agent found by searching for them dynamically, and have alternative strategies and exception handling for coping with the absence of a needed agent service. Dynamic registration and discovery of components by name and features, and the loose coupling are a much greater degree of independent development. A new agent can be loaded and activated while the system is running, and can be found on next lookup.

- **Reactive and proactive autonomy** – Agents pursue their own agendas of activities, and respond to and initiate asynchronous events. Typically they have explicit representations of goals, tasks, priorities, plans, and so can make quite significant adjustments in behaviors when exceptional conditions are discovered, if other agents disappear, or if they refuse to respond. The key is to treat an agent as a collection of reactive behaviors, not just a collection of methods. Agent communication languages and protocols provide a clear framework and expectation of errors, timeouts and service refusals that are to be explicitly handled.
- **Collaboration and coordination** – Techniques and models to choreograph a planned or *ad hoc* society of agents. These include workflow, and state machines, sub-goal delegation and management. Most interesting is emergent behavior as new agents are added, discover each other, and a growing capability.
- **Adaptability and intelligence** – Agents can learn from experience, and adjust themselves to changing situations. We have explored a tasteful integration of machine learning, rule-based and information retrieval techniques, with a blackboard/event-bus style coordination of independent elements.
- **Other salient attributes –** Each agent can be responsible for autonomic self-management, load balancing, etc. We can exploit component and reuse technologies such as frameworks, patterns, generators, and aspects to build individual agents and compatible societies of agents. In particular, the decomposition of behaviors and corresponding protocols across members of a multi-agent society are amenable to a natural aspect-oriented realization.

Agent-oriented software development extends conventional component development, promising more flexible componentized systems, less design and implementation time, and decreased coupling between the agents. In the same way that models, collaborations, interaction diagrams, patterns and aspect-oriented programming help build more robust and flexible component and component systems, the same techniques can be applied to agent components and agent systems [10],[13],[14],[20],[22],[23]

Agent infrastructures provide services and mechanisms so that agents have fewer yet richer interfaces, increasing opportunities for dynamic composition. Agent-oriented programming (AOP) [15], and methods such as GAIA [14] decomposes large complex distributed systems into relatively autonomous agents, each driven by a set of beliefs, desires and intentions (BDI). An agent-oriented developer creates a set of agents (with different beliefs and goals) that collaborate among themselves by exchanging carefully structured messages.

4 Progress Towards Reuse Engineering with Agent Components

Our vision is fairly ambitious, and thus we report on a work in progress so far. We do not yet have a complete, coherent solution. Our research agenda towards the systematic integration of reuse and agent technology has two primary goals:

- Treating software agents as loosely-coupled next-generation components. This yields components and frameworks that are more flexible, adaptable, robust and self-managing, combining agents, workflow and services. We build on existing agent systems, standards and infrastructure such as JADE, ZEUS, FIPA, JAS and J2EE.
- Applying software reuse and model-driven development techniques to the rapid and problem-specific construction of multi-agent systems. This exploits combinations of technologies such as customizable components, patterns, micro-frameworks, aspect-oriented composition, domain-specific kits, generators, visual-builders and use of UML to generate multi-agent protocols and behaviors, and complete agent systems.

Work so far has comprised several threads, and produced several results:

- Developing a robust, industrial-strength infrastructure for agent operations; this has been done by delivering our agents as compatible services in a J2EE environment, producing a system called BlueJADE [24],[25].
- Developing UML-based tools, and some UML extensions to model groups of agents and the protocols between agents[22]. We will discuss this recent work in more detail in the next section.
- We have analyzed the multi-agent architectures and behavior engines of several agent platforms, notably ZEUS, JADE and FIPA-OS [26],[27] and are now ready to embark on a refactoring and reengineering of these systems to make the basic parts more reusable and composable. Some guidance is provided by the Java Agent Specification (*http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/jas/)*
- We are developing a more complete reuse-based model-driven methodology, integrating feature-oriented domain-analysis and use-case driven development to model families of systems, followed by a combination of patterns for collaborations, component-based and aspect-oriented generation of components [20]. This leads to a highly incremental development model, which deals with both agent societies and individual agents.

5 Model-driven Agent Behavior Choreography

A key mechanism that makes multi-agent systems highly flexible, but initially more complex, is the interaction between agents using structured messages. Instead of the interaction between agent components being described by multiple, distinct interfaces defined using an IDL, a standard *agent communication language* (ACL) is used through a single interface.

A component-interface contains a syntactic description of each method, its parameter names and types, return values, and possible exceptions; typically the semantics is implicit, and must be understood from documentation. Instead of defining many interfaces and methods, the agent approach is to use a simple interface with more complex, structured messages. These messages can be extended dynamically as the system evolves, avoiding a costly re-design and re-implementation.

Of extreme interest in this message-based multi-agent setting is how to coordinate the interactions in the form of message-exchange between multiple agents. While some agents are used individually, groups of agents can collaborate to perform more complex tasks. For example, to purchase books, a group of agents will exchange messages in a conversation to find the best deal, bid in an auction, arrange financing, select a shipper, and track the order. Other B2B interactions include service provisioning, supply chain, negotiation, and fulfillment. The grouping can be static or dynamic. The conversation can be between people and agents, or between agent and agent, or a mix. Groups can be statically or dynamically determined.

We need to coordinate the interactions between the agents to achieve a higher-level task, such as requesting, offering and accepting a contract for some services. We call this "choreography" or "conversation management." An agent group will have a series of stylized message exchanges to accomplish this task, perhaps by advertising and using a brokered service, bidding during an auction, responding to a call for proposals, or negotiating a price.

There are several possible levels of choreography of the expected message sequences, depending on the need for a relatively "loose" or more "tight" control of allowed interactions [17], ranging from built-in handling of certain conditions such as time-outs and exceptions, to rule-based systems, and state-machines and workflow for protocols. While earlier work combined workflow and agents [21],[28],[29], we have most recently focused on combining rules and hierarchical state machines

- *Rules* - A set of rules can be defined for any agent, or community of agents. For example, to determine how long to wait for a response, the number of other agents an agent can talk to at one time, and what sort of responses to make to a specific message from various types of agent. Not all request/response patterns are constrained by the rules; rules can be used where appropriate to select which messages to respond to, and how to respond. A standard rule language can be used, such as the forward chaining rule system provided by ZEUS [18], or our use of Jess^{TM} with JADE.
- *Conversation protocols* - Often a group of agents must "lock-step" through a standard protocol of messages; if A says "x" then B says "y". For example, bidding during an auction, or responding to a call for proposals. These protocols can be expressed as FIPA protocol diagrams, UML interaction diagrams, finite state machines, UML state charts or Petri-nets. Each participant can have the same or a compatible model, stepped through explicitly to determine the action and response any incoming message. Our work has used UML state charts [22].

Our current approach uses hierarchical state machines to define the detailed flow of messages, rules to act as message filters, and a rule-based flexible action language for the semantics. Rather than extending interaction diagrams, as done in AUML, we have directly used hierarchical UML State Machines, embedding an event-driven state machine engine within the JADE behavior mechanisms. We developed a visual tool (using Visio) and a set of Java templates and libraries that allow us to completely generate agent behavior code from a UML model [22]. One of the primary benefits of full hierarchical state machines is that we can neatly factor typical exceptional and common timeout behaviors into surrounding composite states, allowing the nested state to focus on the main part of the protocol. We are now working on a way of generating a set of agents and a set of compatible protocol parts by hierarchically decomposing a single state machine describing the agent society into a set of independent state machines. A reengineered version of this event-driven, state machine behavior engine and generator will be released to the JADE community early in 2003.

6 Related Work

AUML [13] extends UML with enhanced interaction diagrams to make more explicit some of the message and protocol handling. UML models of vocabularies, workflow, role diagrams, patterns, and feature trees will drive aspect-oriented generators to create highly customized agent systems [20],[23], The Iconic Modeling Tool [30] uses visual techniques and UML to assemble and control mobile agent programs and itinerary, using a variant of interaction diagrams to show connections.

Agents and workflow can perform a range of simple or complex workflow- like tasks, such as automatic notification via email of the availability of a report, sending a reminder or re-scheduling a meeting [1], or negotiating on a users behalf [6]. Several authors have explored workflow as an important part of the choreography of multiple agents: a light-weight, dynamic agent infrastructure in Java ([29],[29]), supports "on demand," dynamic loading of new classes to extend agents with domain-specific XML interpreters, new vocabulary parsers or workflow. Agents can collaborate to perform a workflow, *e.g., telecom provisioning (BT), or service provisioning (HP).* Agents can represent the participants and resources, the society of agents collaborate to enact the workflow[32]. Agent systems have been used to implement or augment workflow systems ([10], [33], [34]).

Gschwind's Agent Development Kit (ADK) provides an AgentBean model to allow some agents to be assembled from smaller Java-bean components [31]. ZEUS includes a visual generator of agent systems from role models [18]. Kendall uses agent role models and patterns to feed an aspect-based implementation [23].

7 Conclusions

Multi-agent technologies will combine with web-service technologies to produce a robust environment for constructing complex, adaptive systems. To help make this adoption and mainstreaming of agent technologies happen soon, we need to produce robust agent platforms, integrated with J2EE and web-service platforms, and create powerful agent construction toolkits, model-driven generators, and visual builders to quickly define and generate (large parts of) of individual agents and agent systems.

We expect UML, AUML, and workflow techniques to play a large role in the definition, generation and execution of choreographed multi-agent interactions.

Further research and experimentation is needed to make it easier to define and implement different agent systems directly in terms of their features and capabilities. An agent, or set of compatible agents, will be constructed by combining aspects and components representing key capabilities.

References

1. Heineman, G., Councill, W.(eds): Component-Based Software Engineering, Addison-Wesley (2001)
2. Griss, M., Wentzel, K.: Hybrid Domain-specific Kits**,** Journal of Systems and Software, Sep (1995)
3. Griss, M., Pour, G.: Accelerating Development with Agent Components, IEEE Computer, 34(5): 37-43, May (2001)
4. Griss, M., Letsinger, R., Cowan, D., Sayers, C., VanHilst, M., Kessler, R.: CoolAgent: Intelligent Digital Assistants for Mobile Professionals - Phase 1 Retrospective, HP Laboratories report HPL-2002-55(R1) , July (2002)
5. Fonseca, S., Griss, M., Letsinger, R.: An Agent-Mediated E-Commerce Environment for the Mobile Shopper, HPL-2001-157, June (2001)
6. Maes, P., Guttman, R., Moukas, A.: Agents that buy and sell, Communications of the ACM, Vol.42, No.3, March (1999) 81-91
7. Glushko, R., Tenenbaum, J., Meltzer, B.: An XML framework for agent-based E-commerce. Communications of the ACM, Vol.42, March (1999)
8. Meltzer, B., Glushko, R.: XML and Electronic Commerce, ACM SIGMOD. 27.4 December (1998)
9. Huhns, M., Singh, M.: Readings in Agents, Morgan-Kaufman, (1998)
10. Jennings, N., Wooldridge, M.: Agent Technology, Springer (1998)
11. Bradshaw, J.: Software Agents, MIT Press, (1997)
12. Genesereth, M., Ketchpel, S.: Software Agents, Communications of the Association for Computing Machinery, July (1994), 48-53
13. O'Dell, J.: Objects and Agents Compared, Journal of Object Technology, Vol 1, Number 1, May, (2002); also http://www.auml.org/
14. Wooldridge, M., Jennings, N., Kinny, D.: The Gaia Methodology For Agent-Oriented Analysis And Design, AAMAS (2000)
15. Shoham, Y.: Agent-Oriented Programming, Artificial Intelligence, Vol. 60, No. 1, (1993), 139-159.
16. O'Brien, P., Nicol, R.: FIPA: Towards a standard for intelligent agents. BT Technical Journal, 16(3), (1998); also http://www.fipa.org
17. Griss, M.: My Agent Will Call Your Agent, Software Development Magazine, Feb (2000)
18. Nwana, H., Nduma, D., Lee, L., Collis, J.: ZEUS: a toolkit for building distributed multi-agent systems, in Artificial Intelligence Journal, Vol. 13, No. 1, (1999) 129-186; also http://more.btexact.com/projects/agents/ZEUS
19. Bellifemine, F., Poggi, A., Rimassi, G.: JADE: A FIPA-Compliant agent framework, Proc. Practical Applications of Intelligent Agents and Multi-Agents, April (1999), 97-108; also http://sharon.cselt.it/projects/jade
20. Griss, M.: Implementing Product-Line Features By Composing Component Aspects, Proceedings of 1st International Software Product Line Conference, Denver, Colorado, August (2000)

21. Griss, M.: Software Agents as Next Generation Software Components, In Component-Based Software Engineering, George T. Heineman & William Councill (eds), Addison-Wesley, May (2001)
22. Griss, M., Fonseca, S., Cowan, D., Kessler, R.: Using UML State Machines Models for More Precise and Flexible JADE Agent Behaviors, HPL 2002-298(R) and AAMAS AOSE workshop, Bologna, Italy, July (2002)
23. Kendall, E.: Role Model Designs and Implementations with Aspect-oriented Programming, in Proc. of OOPSLA 99, Denver, Co., ACM SIGPLAN, Oct, (1999) 353- 369
24. Cowan, D., Griss, M.: Making Software Agent Technology Available to Enterprise Applications, 1st International Workshop on Challenges in Open Agent Systems, AAMAS'02, Bologna, Italy, July (2002)
25. Cowan, D., Griss, M., Kessler, R., Remick, B., Burg, B.: A Robust Environment for Agent Deployment , AAMAS 2002 - Workshop on Challenges in Open Agent Environments, Bologna, Italy, July (2002)
26. Fonseca, S., Griss, M., Letsinger, R.: Agent Behavior Architectures - A MAS Framework Comparison, AAMAS 2002 - 1st International Conference on Multi-Agent Systems and Applications; also, HPL-2001-332, Dec (2001)
27. Fonseca, S., Griss, M., Letsinger, R.: Evaluation of the ZEUS MAS Framework, HPL-2001-154, June (2001)
28. Chen, Q., Chundi. P., Dayal, U., Hsu, M.: Dynamic Agents for Dynamic Service Provisioning, Intl. Conf. on Cooperative Information Systems, August (1998)
29. Chen, Q., Hsu, M., Dayal, U., Griss, M.: Multi-Agent Cooperation, Dynamic Workflow and XML for E-Commerce Automation, Autonomous Agents 2000, June (2000), Barcelona
30. Falchuk, B., Karmouch, A.: Visual Modeling for Agent-Based Applications. IEEE Computer, Vol. 31, No. 12, December (1998), 31 - 37
31. Gschwind, T., Feridun, M., Pleisch, S.: ADK - Building Mobile Agents for Network and Systems Management from Reusable Components, in Proc. of ASA/MA 99, Oct, Palm Springs, CA, IEEE-CS, pp 13-21; also http://www.infosys.tuwien.ac.at/ADK/
32. Sutton Jr., S., Osterweil, L.: The design of a next generation process programming language, Proceedings of ESAC-6 and FSE-5, Springer Verlag, (1997) 142-158
33. Kaiser, G., Stone, A., Dossick, S.: A Mobile Agent Approach to Light-Weight Process Workflow, In Proc. International Process Technology Workshop, (1999)
34. Shepherdson, J., Thompson S., Odgers, B.: Cross organizational Workflow Coordinated by Software Agents, WACC '99- Work Activity Coordination and Collaboration Workshop Paper, February (1999); also http://www.labs.bt.com/projects/agents/index.htm