Software Engineering with Java Agent Components

Martin L. Griss

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

Invited paper, Borcon, Nov 2003.

Abstract*. Component-based software engineering (CBSE) and product-line development have delivered significant improvements in software development, promising improved reuse, agility and quality. Components can be (largely) independently developed. To further increase the independence and flexibility of components, software agent components have great promise to improve application and system construction. Built on conventional distributed computing and application management platforms, on web service technology or within a P2P infrastructure, agent components are effective for independent development, for scalable and robust systems and dynamic evolution of features. There are many kinds of software agents, with differing characteristics such as mobility, autonomy, collaboration, persistence and intelligence, each offering greater flexibility than traditional components. We will discuss agent technology and those elements that enable more robust, scalable and evolutionary systems, and the application of agent components to personal assistants and software engineering environments.*

1. Introduction

For some time now, component-based software engineering (CBSE) has promised, and indeed delivered, significant improvements in software development [Heineman and Council, 2001]. 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 [Griss and Wentzel, 1995].

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 [Griss and Pour, 2001]. 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 [Griss, et. al. 2002; Fonseca et. al., 1999], and now at UC Santa Cruz in collaboration with the University of Utah [Griss and Kessler, 2003], 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, flexible 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 [Maes et. al., 1999]. 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 [Glushko et al., 1999; Meltzer and Glushko, 1998].

An overview of agent capabilities from a large-scale AOSE/CBSE perspective can be found in books ([Huhns and Singh, 1998; Jennings and Wooldridge 1998], papers ([Genesereth and Ketchpel, 1994; Odell, 2002; Wooldridge et. al., 2000; Shoham 1993]), conferences 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 [O'Brien and Nicol 1998]

There are several different kinds of agent system [Griss, 2000]; our work at Hewlett-Packard Laboratories and UC Santa Cruz [Griss et al, 2002] focuses 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 shopping, meeting scheduling agents, mail agents, and software development.
- **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.

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 based most of our work on two Java-based toolkits: ZEUS [Nwana et al, 1999] and JADE [Bellifemine, 1999].

In the rest of the paper, we discuss agents as next-generation components and selected features appropriate to the integration of reuse and agent technologies. We also describe application of agents to personal and team assistants for software development environments.

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 [Griss 2000a; Griss 2001; Griss et. al. 2002; Griss and Kessler 2003] where we provide a graphical model and discuss several of the characteristics of a typical agent system. 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 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. Each agent can be responsible for autonomic self-management, load balancing, etc.
- **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.

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 [Jennings and Woolridge 1998; Odell 2002; Woolridge et al., 2000; Kendall 1999].

Agent infrastructures provide services and mechanisms so that agents have fewer yet richer interfaces, increasing opportunities for dynamic composition. Agent-oriented programming (AOP) [Shoham 1993], and methods such as GAIA [Woolridge et. al., 2000] 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

This vision is fairly ambitious, but significant progress has been made. Agent technology has matured considerably via standards by Foundation for Intelligent Physical Agents (www.fipa.org), open source agent toolkits, such as JADE [Bellifemine 1999], large-scale agent interoperability test (www.agentcities.org) and numerous applications of agents in research, advanced development and products [Griss, 2000]. New supporting technologies based on HTTP, P2P, J2EE and .NET technologies are available, and new methods, models and tools to develop agents and choreograph agent-based collaborations are becoming available. The research agenda has three 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.
- Apply this more robust and flexible agent technology to flexible, adaptive applications.

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 [Cowan and Griss, 2002; Cowan et al, 2002]. Also, agent management has been improved.
- Developing UML-based tools, and some UML extensions to model groups of agents and the protocols between agents [Griss et. al, 2002]. AUML [Odel, 2002] 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 [Griss 2000; Griss and Kessler, 2003]. The Iconic Modeling Tool [Falchouck and Karmouch, 1998] uses visual techniques and UML to assemble and control mobile agents, using a variant of interaction diagrams to show connections.
- Improving agent “choreography,” using state machines, Petri nets or workflow to coordinate the interactions between agents. 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. 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 [Griss 2000], 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. Workflow is an important choreography technology. A light-weight, dynamic agent infrastructure in Java with agents has been prototyped [Chen et al, 2000]. Agents can represent the participants and resources, the society of agents collaborate to enact the workflow [Sutton and Osterweil, 1997;Kaiser et al, 1999]. Our work combines rules and hierarchical state machines [Griss et al, 2002;] Griss and Kessler, 2003], to provide both flexibility and precision.
- Analyzing the multi-agent architectures and behavior engines of several agent platforms, notably ZEUS, JADE and FIPA-OS [Fonseca et al., 2001, 2001a], leading to a proposed 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/)*

- Developing a more complete reuse-based model-driven methodology and toolset, 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 [Woolridge et al, 2002; Griss and Kessler 2003]. This leads to a highly incremental development model, which deals with both agent societies and individual agents. Gschwind's Agent Development Kit (ADK) provides an AgentBean model to allow some agents to be assembled from smaller Java-bean components [Gschwind et al, 1999]. ZEUS includes a visual generator of agent systems from role models [Nwana et al, 1999]. Kendall uses agent role models and patterns to feed an aspect-based implementation [Kendall, 1999].

5. Personal and Team Assistants for Software Engineering

At UCSC, we are developing several Personal and Team Assistant agents. The concept and experience with personal agents that reduce work and overload by providing task and domain-aware assistance was most clearly articulated by Maes [Maes et. al., 1994]. At HP, we developed several context-aware personal assistants and a robust J2EE multi-agent infrastructure for individual and team applications such as meeting scheduling, shopping and email management [Griss et al., 2002; Bergman et al., 2002; Cowan and Griss 2002; Griss and Letsinger, 2000]. The personal assistant agent uses a set of personal service agents, including Notification, Phone, Email, Calendar, Location and Web information agents. These agents collaborate to manage appointment requests, automatically filter, file and selectively respond to selected emails based on user profile, preferences, current tasks and calendar-derived information, such as "on travel", "urgent notify to mobile device", "meeting reminder", and "urgent message forwarder". The location agent will try to keep track of where the (mobile) user is, based on various cues. The notification agent will use personal preferences, urgency, and information from the user's calendar and other context to decide how to forward the notification: via email, instant message or phone. The phone agent selects a wired or wireless phone, and sends an appropriate (summary) notification using voicemail or text message.

Individual personal agents can register by role and interest with a team assistant. The team (or group) assistant interacts with other team members' personal agents and group-oriented service agents in appropriate contexts, redistributing notifications and coordinating group decisions . Agents collaborate to discover and merge semantically related/relevant information and services, integrating status of team members and tasks, and manage joint tasks, goals and calendars.

These personal assistants and team have been applied to prototype solutions in several problem areas, related to intelligent traffic information systems, mobile professional support [Griss et. al 2002] and software engineering [Rodriguez 2002]. We briefly describe the use of agents to create an intelligent team software development environment. In general, software development agents support tool integration, and process management and work-product design tasks of software developers, managers and librarians. Agents assist in the coordination between tools and team members, transform and filter tool, email and software events, monitor upcoming and overdue milestones, and notify of software lifecycle events for team. Agents use rules and explicit process models of routine software development tasks such as configuration, testing, and reviews to assist in non-semantic tasks, triggered by role, context, tasks, and events. Agents use explicit UML and feature models of domain, architecture, components and applications, to assist in finding and modifying selected work-products as they develop, offering modeling and design critique and reuse assistance. Agents will offer heuristic advice and monitor the most risky activities from a list of risk factors developed by a detailed study.

Agents help integrate a variety of tools, using wrappers to intercept and forward important events between the tools. Current software tools and tool interfaces are maturing, enableing this kind of work. For example, the rule-based modeling UML assistants developed by Jaczone (www.jaczone.com) to augment the use of Rational's XDE UML tool shows how simple agents can

produce better models. While interesting, it does not support teams, and focuses only on UML modeling heuristics. Many tools now include fairly sophisticated “wizards” and expose rich APIs for tool “plug-ins” and scripts. These tool capabilities will allow independently written agents, each with its own rules and heuristics, to watch for risky situations, and negotiate with other agents and the user, to prioritize advice, resolve some process or work-product conflicts and manage related routine tasks and workflow. Related work includes intelligent design assistants for software engineering or other branches of engineering include such the Programmers Apprentice [Rich and Waters, 1990; Fischer and Ye, 2001] design critics [Fischer et al., 1991], and information tool integration [Cranefield and Purvis; Genesereth et al., 1997].

In our first prototype, described in more detail by Marko Rodriguez [Rodriguez, 2003] in this conference, we are developing an agent based toolset for a small group of developers, architects and a manager using JBuilder and other tools on their personal machines (our SCATE group). They use ANT on their machines to manage local building and testing. They also use CVS and ANT on a central server to manage version control and nightly builds. We run an AntAgent on the server, listening for ant-events from runs of ANT on any machine. Individuals then have their personal assistants subscribe to the ant agent, indicating which projects and which kind of events they are interested in. When appropriate, the ant agent sends a customized message to all relevant subscribers, based on role, project, and interests.

Future work will add other kinds of listeners (to JBUilder, CVS, etc.), more and powerful rule-based and machine-learning based filters to decide which events to send to the user PA, and how and whom to notify. We are also developing several GroupAssistants that can manage team subscriptions, voting, coordination, scheduling and role-oriented notification [Rodriguez, 2003].

6. Conclusions

Multi-agent technologies will combine with web-service technologies to produce a robust environment for constructing complex, adaptive systems. We see important application for flexible, adaptive applications for mobile users, collaborative environments, education, financial and shopping applications. To help make this adoption and mainstreaming of agent technologies happen soon, we as a community 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 need libraries of agent parts, complete agents, and pre-connected agent societies. 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

[Bellifemine 1999] 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

[Bergman et al. 2002] Bergman, R., Griss, M., and Staelin, C. A Personal Email Assistant, HP Laboratories report HPL-2002-236, August 2002 (*Submitted to Percom 2003.)*

[Chen et al., 2000] 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

[Cowan and Griss, 2002] Cowan, D. and 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

[Cowan et. al., 2002] 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)

[Cranefield and Purvis] Cranefield, S. and Purvis, M. Agent-based Integration of General-Purpose Tools, Proceedings of the CIKM '95 Workshop on Intelligent Information Agents, 1995.

[Falchouck and Karmouch, 1998] Falchuk, B., Karmouch, A.: Visual Modeling for Agent-Based Applications. IEEE Computer, Vol. 31, No. 12, December (1998), 31 – 37.

[Fischer and Ye, 2001] Fischer, G. and Ye, Y., Personalizing Delivered Information in a Software Reuse Environment, 8th International Conference on User Modeling, July 2001

[Fischer et al., 1991] Fischer, G., Lemke, A. C., Mastaglio, T, and Morch, A. I., The role of critiquing in cooperative problem solving, ACM Transactions on Information Systems (TOIS) 9(2), April 1991.

[Fonseca et al., 1999] Fonseca, S., Griss, M., Letsinger, R.: An Agent-Mediated E-Commerce Environment for the Mobile Shopper, HPL-2001-157, June (2001)

[Fonseca et. al, 2001a] Fonseca, S., Griss, M., Letsinger, R.: Evaluation of the ZEUS MAS Framework, HPL-2001-154, June (2001)

[Fonseca, et. al., 2001]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)

[Genersereth and Ketchpel, 1994] Genesereth, M., Ketchpel, S.: Software Agents, Communications of the Association for Computing Machinery, July (1994), 48-53

[Genesereth et al., 1997] Genesereth, MR., Keller, AM., and Duschka, OM. , Infomaster: an information integration system, Proceedings of the 1997 ACM SIGMOD international conference on Management of data, 1997 , Tucson, Arizona, United States.

[Glushko et al., 1998] Glushko, R., Tenenbaum, J., Meltzer, B.: An XML framework for agent-based E-commerce. Communications of the ACM, Vol.42, March (1999)

[Griss and Kessler, 2002] Griss, M., and Kessler, RR. "Achieving the Promise of Reuse with Agent Components." First International Workshop on Software Engineering for Large-Scale Multi- Agent Systems, ICSE, May 2002, Springer, p. 1-9. (2002)

[Griss and Letsinger, 2000] Griss, M. and Letsinger, R., Games at Work - Agent-Mediated E-Commerce Simulation, Autonomous Agents 2000, Barcelona, Spain, June 2000. Also The IT-Journal, 3Q, 2000, HP.

[Griss 2000] Griss, M., My Agent Will Call Your Agent - But Will It Respond?, Software Development Magazine, Feb 2000.

[Griss 2001] Griss, M., Software Agents as Next Generation Software Components, In "Component-Based Software Engineering," Edited by Heineman, G., and Councill, W., Addison-Wesley, May 2001.

[Griss and Pour, 2001] Griss, M., Pour, G.: Accelerating Development with Agent Components, IEEE Computer, 34(5): 37-43, May (2001)

[Griss and Wentzel, 1995] Griss, M., Wentzel, K.: Hybrid Domain-specific Kits, Journal of Systems and Software, Sep (1995)

[Griss et al., 2002] Griss, M., Letsinger, R., Cowan, D., Sayers, C., VanHilst, M., and Kessler, R., CoolAgent: Intelligent Digital Assistants for Mobile Professionals - Phase 1 Retrospective, HP Laboratories report HPL-2002-55(R), July 2002

[Griss et. al, 2002a] 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)

[Griss, 2000] Griss, M.: Implementing Product-Line Features By Composing Component Aspects, Proceedings of 1st International Software Product Line Conference, Denver, Colorado, August (2000)

[Griss, 2001] Griss, M.: Software Agents as Next Generation Software Components, In Component-Based Software Engineering, George T. Heineman and William Councill (eds), Addison-Wesley, May (2001)

[Gschwind et. al., 1999] 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/

[Heineman and Council, 2001] Heineman, G., Councill, W.(eds): Component-Based Software Engineering, Addison-Wesley (2001)

[Huhns and Singh, 1998] Huhns, M., Singh, M.: Readings in Agents, Morgan-Kaufman, (1998)

[Jennings and Wooldridge, 1998] Jennings, N., Wooldridge, M.: Agent Technology, Springer (1998)

[Kaiser et al., 1999] Kaiser, G., Stone, A., Dossick, S.: A Mobile Agent Approach to Light-Weight Process Workflow, In Proc. International Process Technology Workshop, (1999)

[Kendall, 1999] Kendall, E.: Role Model Designs and Implementations with Aspect-oriented Programming, in Proc. of OOPSLA 99, Denver, Co., ACM SIGPLAN, Oct, (1999) 353- 369

[Maes et al, 1999] Maes, P., Guttman, R., Moukas, A.: Agents that buy and sell, Communications of the ACM, Vol.42, No.3, March (1999) 81-91

[Maes et al., 1994] Maes, P., Guttman, R., and Moukas, A., Agents that reduce work and overload, Communications of the ACM, Vol. 42, No. 3, March (1994) 81-91

[Meltzer and Glusko, 1998] Meltzer, B., Glushko, R.: XML and Electronic Commerce, ACM SIGMOD. 27.4 December (1998)

[Nwana et al., 1999] 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

[O'Brien and Nicol] O'Brien, P., Nicol, R.: FIPA: Towards a standard for intelligent agents. BT Technical Journal, 16(3), (1998); also http://www.fipa.org

[Odell, 2002] O'Dell, J.: Objects and Agents Compared, Journal of Object Technology, Vol 1, Number 1, May, (2002); also http://www.auml.org/

[Pour and Hong 2003] Pour, G. and Hong, A., Internet-Based Multi-Agent Framework for Software Service Retrieval and Delivery, in Computational Intelligence for Modeling and Control Series, 2003.

[Pour 2002] Pour, G., Web-Based Architecture for Component-Based Application Generators, Internet Computing Series, 2002.

[Pour 2003] Pour, G., Web-Based Multi-Agent Architecture for Software Development Formal Peer Inspection, in Computational Intelligence for Modeling and Control Series. 2003.

[Rich and Waters, 1990] Rich, C. and Waters, RC. The Programmer's Apprentice. ACM Press (Frontier Series), Addison-Wesley 1990.

[Rodriguez, 2003] M. Rodriguez, Using Java Agents to Make JBuilder A More Group-Centric IDE, Borcon, San Jose, Nov 2003.

[Shoham, 1993] Shoham, Y.: Agent-Oriented Programming, Artificial Intelligence, Vol. 60, No. 1, (1993), 139-159.

[Sutton and Osterweil] 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

[Wooldridge et. al., 2000] Wooldridge, M., Jennings, N., Kinny, D.: The Gaia Methodology For Agent-Oriented Analysis And Design, AAMAS (2000)