



UNIVERSITY OF
CALGARY

**SENG609.22: Agent-Based Software
Engineering
Assignment**

Agent-Oriented Engineering Survey

By: Allen Chi
Date: 20th December 2002

Course Instructor: Dr. Behrouz H. Far

0. Abstract

Agent-Oriented Software Engineering is considered to be the next step in the evolution of traditional software engineering paradigm. Agent-based system is still at an early stage of the development, and yet the widespread adoption of the Internet has made it a feasible goal.

In Agent-Oriented Software Engineering, system is composed of agents, as opposed to object in Object-Oriented paradigm. An agent is a software entity that functions autonomously and can communicate and interact with other agents and/or human. With the use of agents as a high-level abstraction in a system, many researchers believed that Agent-Oriented Software Engineering is suitable in designing and implementing complex, heterogeneous, distributed, and networked software system.

Being a relatively new field, Agent-Oriented Software design and development is currently based on inspiration and informal guidelines. This paper has sought to give an overview of Agent-Oriented Software Engineering.

1. Introduction

With the wide adoption of the Internet as an open environment, it makes the widespread adoption of multi-agent technology a feasible goal. One aspect of agent that is broadly mentioned in many literatures is that an agent is a conceptual entity that can perceive and act in a proactive or reactive manner within a dynamic and uncertain environment where other agents exist and interact with each other based on shared knowledge of communication and representation. Simply put, agents that are not able to work together with other agents, including human, are virtually useless.

Agent-Oriented Software Engineering (Agent-SE) is one of the most recent contributions to the field of Software Engineering. It has been described as the new paradigm and a logical evolution of traditional approaches to software engineering. Agent-SE offers natural and powerful means of conceptualizing, designing, building and managing complex, scalable, heterogeneous, open, distributed and networked systems. It has several benefits compared to the existing software development approaches, in particular the ability to use agents to represent high-level abstractions of active entities in a software system.

In this report, we present an overview of the Agent-Oriented Software Engineering. The structure of this report is as follows. Section 2 introduced some terminology used in this paper. Section 3 gives an overview of what agent is. Section 4 gives an overview of some methodology used in Agent-SE. Section 5 gives an overview of some common Agent-Architecture Model. Section 6 talks about the usage of design pattern in Agent-SE. Section 7 simply concludes the paper.

2. Terminology

Like other software engineering paradigms, Agent-SE has a set of closely related terms used in the literature. These terms are described as follows:

Agent-Oriented Programming (AOP) is introduced by Shoham in 1993. It is an improvement and extension over Object-Oriented Programming (OOP). AOP is a programming paradigm that supports societal view of computation. In AOP, objects, also known as agents, interact with each other to achieve individual goals.

Agent-Oriented Development (AOD) is an extension of Object-Oriented Development (OOD). It includes the full development process that covers the requirement specification and design as well as programming.

Agent-Oriented Software Engineering (Agent-SE), Multi-agent systems Engineering (MaSE), and Agent-based Software Engineering are semantically equivalent terms. However, MaSE refers to a particular methodology, and Agent-SE seems to be the most widely used-term. The difference between Agent-SE and AOD is that Agent-SE also covers issues such as reusability and maintenance of the system as well as the development process.

3. Agent Overview

Agents are often described as entities with attributes that are considered useful in a particular domain. For example, personal assistant agents (PAA) are entities that help users perform a task (e.g. Microsoft animated paperclip agent); Information agents are entities that filter and coherently organize unrelated and scattered data (e.g. Google Web Spider); Mobile agents are entities that are able to roam networking environment to fulfill their goals.

Although, it is difficult to give a precise definition to the term agent, many researchers have attempted to provide a meaningful

classification of agent based on their attributes. A list of the most common accepted agent attributes is as follows [Bradshaw, 1997][Far, 2002]:

- **Autonomy:** goal-directness, proactive and self-starting behavior
- **Mobility:** the ability to migrate in a self-directed way from one host platform to another
- **Collaborative behavior (Social ability):** the ability to work with other agents to achieve a common goal
- **Inferential capability:** the ability to act on abstract task specifications, using models of self, situation and/or other agents
- **Knowledge-level communication ability:** the ability to communicate with other agents with language more resembling human-like "speech acts" than typical symbol-level program-to-program protocols
- **Reactivity:** the ability to selectively sense and act in a dynamic and uncertain environment
- **Temporal continuity:** persistence of identity and state over long periods of time
- **Adaptively:** the ability to learn and improve with experience in a dynamic and uncertain environment
- **Personality:** the ability to manifest attributes of a believable human character

Based on the agent attributes, we can further classify them using the notion of agency [Jennings, 1995]. There are two types of agency: weak and strong. In the weak notion of agency (from Distributed computing and Distributed Artificial intelligent), agents have their own will, they take initiative, they are able to interact with each other, and they respond and act on stimulus. In the strong notion of agency (from Artificial Intelligent), agents not only preserve the weak notions of agency, but also are able to move around, they are truthful and do what they are told to do, and they will perform in an optimal manner to achieve their goals.

3.1 What Is An Agent?

There is no once-and-for-all definition to the term agent. However, one notion of agent that is broadly mentioned in many literatures is that an agent is a software entity that can perceive and act in a proactive or reactive manner within a dynamic and uncertain environment where other agents exist and interact with each other based on shared knowledge of communication and representation [Jennings, 1995], [Shoham, 1997], [Far, 2002].

To consider a software entity as an agent, it should at least have the ability to interact (e.g. cooperate, coordinate and compete) with other agents and/or human as well as the ability to communicate and shared knowledge.

3.2 Is Agent = Object?

It is sometimes difficult to differentiate between an agent and an object since they both share many similar characteristics. For example, OOP views system consisting of object communicating with one another to perform internal computations based on ad-hoc messages, whereas AOP views system consisting of agents rather than objects. The internal computations of agents are based on beliefs, capabilities, and choices, and agents communicate with other agents based on messages adopted from speech-act theory.

Although, this view allows us to see the similarities between agent and object, the differences are less obvious. The following points out the differences between agent and object [Jennings, 1998], [Far, 2002].

- Agents do not invoke method upon one another, but rather requesting actions to be performed. In OOP, the decision lies within the object that invokes the method, whereas in AOP, the decision lies within the agent that receives the request. In other words, agents have the right to say "no" to a request. Simply put, objects are totally obedient to one another and do not have autonomy over their choice of actions
- Agents are each considered to have its own thread of control, whereas in the standard object model, there is a single thread of control in the system
- In OOP, objects are used as abstractions for passive entity (e.g. a car), whereas in AOP, agents are regarded as a possible successor of objects since they can improve the abstractions of active entity (e.g. broker)
- Agents are flexible, autonomous software entities, whereas objects are not.

3.3 Is Agent A Universal Solution?

Being a relatively new concept, there is a danger that software designers become overly optimistic regarding the abilities of agent. Jennings and Wooldridge discussed the potential pitfalls and classified

them into five groups [Jennings, Wooldrigde, 1999]. They are summarized below:

- Political Pitfalls: this may occur if the concepts of agent are applied as a universal solution to all problems
- Conceptual Pitfalls: this may occur if the designer forgets that agents are multi-threaded software entity which may not be appropriate for systems or domains with global constraints and requiring real-time response
- Analysis and Design Pitfalls: this may occur if the designer ignores other possible software design paradigms and methodologies, and/or other related technologies
- Agent-Level Pitfall: this may occur if the designer tries to use too much or too little artificial intelligence in the agent-system
- Society-Level Pitfall: this may occur if the designer sees agents everywhere or applies too few agents in the agent-system

Although agents are common object that can potentially be used by many different application fields, software designers should be cautious when applying them. Designers should ask themselves questions such as whether an agent is needed? Is there a valid reason to justify the used of agent? Are there too many or too few agents in the system?

3.4 Agent's Interaction

Interaction is the most important feature of an agent. Agent must interact to share information and knowledge in order to achieve its goals. Researchers have identified three key elements needed to achieve agent interaction. They are:

- A common agent communication language and protocol between agents
- A common format for the content of communication between agents
- A shared Ontology between agents

3.5 Agent Communication Language

There are two main approaches to design an Agent Communication Language (ACL). The first approach is procedural where communication is based on executable content. The second approach is declarative where communication is based on declarative statements. Due to the fact that it is difficult and complex to control, coordinate and merge the executable content using the procedural

approach, the declarative approach is preferred over the other. One of the most popular declarative agent languages is the Knowledge Query and Manipulation Language (KQML).

3.6 Ontology

Ontologies are defined as specification schemes for describing concepts and their relationships in a domain [Far, 2002]. It is important that agents not only have ontologies to conceptualize a domain, but also have ontologies with similar constructions. Such ontologies are called common ontologies. Once interacting agents have committed to a common ontology, they will use it to interpret communication interactions, thereby leading to mutual understanding among agents.

4. Agent-Based Methodology

Neither of the traditional software engineering paradigms provides well-defined engineering techniques or formal guidelines to incorporate agent-oriented viewpoint in designing and developing agent-based software systems [Far, 2002]. In particular, they fail to adequately represent the autonomous problem-solving behavior of an agent; they also fail to capture the complexity of an agent system's organizational structures; and fail to model agent's way of performing interactions.

The following sub-sections give a brief overview of some methodologies that are tailored to the analysis and design of agent-based systems.

4.1 The Gaia Methodology

Gaia is a general methodology presented by Wooldridge, Jennings and Kinny that support both the micro-level (agent structure) and macro-level (agent society and organization structure) of agent development. Its intend is to help software designer systematically develop an implementation-ready design based on system requirements.

The Gaia methodology consists of two phases: analysis and design. In Gaia analysis phase, the first step is to analyze and define roles in the system, and the second step is to model interactions between the roles. In Gaia, roles consist of the following four attributes:

- Responsibilities: sub-divided into two types: liveness properties meaning the role has to add something good to the system, and safety properties meaning the roles will prevent and disallow something bad to happen
- Permissions: define what the role allows, in particular, what information it allows to access
- Activities: define the type of tasks the role can perform without interacting with other roles
- Protocols: define the specific patterns of interaction

In Gaia design phase, the first step is to map roles into agent types, and then decide the right number of agent instances of each type. The second step is to determine the services model needed to fulfill a role by identifying the main services that will be associated with each agent type, and the last step is to create the acquaintance model for the representation of communication for each agent type.

In conclusion, the key concepts in Gaia are roles, which have associated with them responsibilities, permissions, and protocols. Roles can interact with one another in certain institutionalized ways, which are defined in the protocols of the respective roles. However, Gaia also has its limitation. It requires that inter-agent relationships (agent organization) and agent abilities are static at run-time, and it is not suitable to use for developing open and unpredictable domain of internet applications, but has proven as a good approach for developing closed domain agent-systems.

4.2 The MASSIVE Development Method

The Multi Agent Systems Iterative View Engineering (MASSIVE) development method introduced by Jürgen [1999] is a practical software engineering development model that is specialized for the development of multi-agent system.

MASSIVE provides a product model that is used to describe the design and the implementation of the target system, an iterative process model that guides the designer in constructing the product models according to the user requirements. The process model is sub-divided into a macro process model that covers the entire life cycle of the system development, and several micro process models that are used to describe particular aspects of the target system. In addition, it introduces an organizational framework that supports learning and reuse over project boundaries and allows the project management to configure the process and product models of a particular project.

MASSIVE uses the notion of views that form the conceptual basis for a wide range of product models that are developed and refined throughout various software projects. A view represents a set of conceptually linked features of the target system. In other words, a view is a projection of the complete model onto a particular subject. MASSIVE consists of seven views:

- Environment View: in this view the environment of the target system is analyzed from the designer's perspective to capture the global knowledge as well as from the system's perspective to capture the local knowledge
- Task View: in this view, the functional aspects of the target system are analyzed, and a task hierarchy is generated to determine the basic problem on solving capabilities of the entities in the target system, while the non-functional requirements of the target system are defined and quantified as far as possible
- Role View: this view determines the functional aggregation of the basic problem on solving capabilities according to the physical constraints of the target system
- Interaction View: In this view, interaction within the target system is seen as a generalized form of conflict resolution that is not limited to a particular form such as communication. Instead, several generic forms of interaction exist that can be instantiated in a wide variety of contexts. Thus, the target problem should be analyzed with respect to the applicability of these generic forms before designing new forms
- Society View: this view is used to classify the society that either pre-exists within the organizational context of the system or that is desirable from the point of view of the system designer. A society model is then developed that is consistent with the roles within the society and that achieves the defined goals
- Architectural View: this view is a projection of the target system onto the fundamental structural attributes with respect to the system design. The major aspects that are dealt with in this view are the system architecture as a whole and the agent architecture
- System View: this view deals with aspects that affect several other views at the same time or even the system as a whole

In conclusion, MASSIVE features a view-oriented product model, an iterative process model and an institutional framework that supports organizational learning and reuse over project boundaries. Like Gaia,

it also has the notion of role. A problem that designers may encounter when using MASSIVE is that, there is sometimes no clear boundary between views. That is, several views maybe applied to the same context, thus result in high coupling between views, which is not desirable. General speaking, we like to have high cohesion among the elements grouped together in the same view, and achieve a low coupling between other views.

5. Agent-Based Architecture Model

Several industrial and research groups started to pursue the standardization of agent-based architecture. The most notable ones are the Object Manager Group (OMG), the Foundation for Physical Agents (FIPA), and the Knowledge-able Agent-oriented System (KAoS) group. The following sub-section gives a brief overview of each model.

5.1 Object Manager Group Model

The OMG model outlines the characteristics of an agent environment composed of agents and agencies as entities that collaborate using general pattern and policies of interaction [Virdhagriswaran, Osisek O'Connor, 1995]. Agents in this model are characterized by their capabilities, their interactions, and mobility, whereas an agency is defined to support concurrent agent execution, security and agent mobility.

5.2 Foundation For Intelligent Physical Agents Model

The FIPA [FIPA, 2002] approach is based on a minimal framework for the management of agents in an open environment. This framework consists of a reference model and an agent platform. The reference model specifies the normative environment within which agents exist and operate, whereas the agent platform specifies an infrastructure for the deployment and interaction of agents. Prominent efforts are their Agent Management and Agent Communication Language (ACL) specifications. As of this writing, FIPA has standardized their specifications.

5.3 Knowledge-able Agent-Oriented System Model

KAoS is an open distributed architecture that describes agent implementation by starting from the notion of a simple generic agent to more specific role-oriented agents such as mediator agent, and

elaborates on the interactive dynamics of agent-to-agent messaging communication by using conversation policies.

6. Design Pattern in Agent-Oriented SE

Design patterns are a general solution to a specific recurring design problem. Several templates for software design pattern have been developed. The most notable design pattern templates are by the Gang of Four (GoF) [Gamma, Helm, Johnson, Vlissides, 1994]. Their template is used for patterns that describe constellations of classes that solve common software design problems of relatively small scope.

In the agent world itself, several attempts have been made by researchers to introduce patterns and pattern language in order to promote advances in agent-based technology, increase reuse and quality of code and reduce the effort of development of agent-based systems.

Aridor and Lange [1998] suggest a classification scheme for design pattern in a mobile agent context. Their classification scheme has three classes.

- Traveling class: this class specifies features for agents that move between various environment
- Task class: this class specifies how agents can perform task
- Interaction class: this class specifies how agents can communicate and cooperate

Kendall, Mahkon and Jiang [1997], suggests a seven layers model. They are mobility, translation, collaboration, actions, reasoning, beliefs and sensory. The three lowest layers have patterns that select the mental model of the agent. Compared to the model suggested by Aridor and Lange, the layered model has similar logical groupings of pattern. The mobility layer together with the translation layer corresponds to the traveling class, the collaboration layer corresponds to the interaction class, and the action layer corresponds to the task class. The difference is that the seven layered model aims to cover all main types of agent whereas the three class model covers only the mobile agent.

7. Summary

Agent-SE is considered to be the next step in the evolution of traditional software engineering paradigm. Agent-based system is still

at an early stage of the development, and yet the widespread adoption of the Internet has made it a feasible goal.

Being a relatively new software engineering paradigm, Jennings, a prominent researcher of the agent field points out that the failure of keeping promises and becoming an offer of media hype and then slaughter, could perfectly well happen to the field of agent research. If the technology is to be a success, we need to develop a better understanding of the situations in which agent solutions are appropriate as well as develop formal guidelines, heuristics, and formal principle to move forward agent-based system development.

This paper has sought to give an overview of Agent-Oriented Software Engineering. Further work should include a more thorough analysis of each methodology and architecture, look at how agents are current used in the industry, and include material on uncertainty management. For those who are interested in Agent-SE, it might pay off to invest some time to read and understand the key concepts and the recent advances in this area.

8. Bibliography

[Bradshaw, 1997]

Bradshaw, J.M., An Introduction to Software Agent, In: Software Agent, J.M. Bradshaw (Ed.), Menlo Park, Calif., AAAI Press 1997, pages 3-46

[Far, 2002]

609.22 Agent-Based Software Engineering, Lecture Notes:

An Overview of Software Engineering:

<http://www.enel.ucalgary.ca/People/far/Lectures/SENG609-22/01/index.html>

Methodologies in Agent-based System Analysis and Design:

<http://www.enel.ucalgary.ca/People/far/Lectures/SENG609-22/PDF/SENG609-22-02.pdf>

Agent Communication and Knowledge Sharing:

<http://www.enel.ucalgary.ca/People/far/Lectures/SENG609-22/PDF/SENG609-22-03.pdf>

Agent-Based System and Organization:

<http://www.enel.ucalgary.ca/People/far/Lectures/SENG609-22/PDF/SENG609-22-04.pdf>

FIPA: Foundation of Intelligent Physical Agent:

<http://www.enel.ucalgary.ca/People/far/Lectures/SENG609-22/PDF/SENG609-22-05.pdf>

[Jenning, 1995]

Jennings, N.R. and Wooldridge, M. Intelligent Agents: Theory and Practice, page 4-5:

<http://citeseer.nj.nec.com/rd/53078083%2C97055%2C1%2C0.25%2CDownload/http://citeseer.nj.nec.com/cache/papers/cs/318/http:zSzzSzwww.shiratori.riec.tohoku.ac.jpzSz%7EkinozSzAg-Theor-Pract.pdf/wooldridge95intelligent.pdf>

[Shoham, 1997]

Shoham, Y. An Overview on Agent-Oriented programming, In: Software Agent, J.M. Bradshaw (Ed.), Menlo Park, Calif., AAAI Press 1997, pages 271-290

[Jenning, 1998]

Jennings N. R., Sycara K. and Wooldridge M. J. A Roadmap of Agent Research and Development, Journal Of Autonomous Agents and Multi-Agent Systems.

<http://citeseer.nj.nec.com/rd/53078083%2C110980%2C1%2C0.25%2CDownload/http://citeseer.nj.nec.com/cache/papers/cs/5550/ftp:zSzzSzftp.elec.qmw.ac.ukzSzpubzSzisagzSzdistributed-aizSzpublicationszSzaa-mas.pdf/jennings98roadmap.pdf>

[Jenning, Wooldridge, 1999]

M. Wooldridge, N.R. Jennings, Software engineering with agents: Pitfall and pratfalls, IEEE Internet Computing 3 (3) (1999) 20-27.

[Jenning, Wooldridge, Kinny, 2000]

Michael Wooldridge, Nicholas R. Jennings, David Kinny, The Gaia Methodology For Agent-Oriented Analysis And Design, Autonomous Agents and Multi-Agent Systems, 2000

<http://citeseer.nj.nec.com/rd/53078083%2C513675%2C1%2C0.25%2CDownload/http://citeseer.nj.nec.com/cache/papers/cs/26098/http:zSzzSzwww.ecs.soton.ac.ukzSz%7EnrjzSzdownload-fileszSzjaamas2000.pdf/wooldridge00gaia.pdf>

[Jürgen, 1999]

Jürgen, L. [The MASSIVE Development Method for Multiagent Systems](#). In Proceedings of the Fifth International Conference on the Practical Application of Intelligent Agents and Multi-Agents (PAAM2000), Manchester, 2000

[Roberto, 1999]

Roberto A. Florez-Mendez, Towards A Standardization Of MAS: An Overview, 1999
www.cpsc.ucalgary.ca/~robertof/publications/crossroads99/crossroads99.pdf

[Kendall, Malkoun, Jiang, 1999]

Kendall E. A., Malkoun M. and Jiang C. Multiagent systems design based on object oriented patterns, 1999
<http://www.cs.toronto.edu/km/tropos/laypattern.pdf>

[Aridor, Lange, 1998]

Aridor, Y. and Lange D. B. Agent Design patterns: Elements of Agent Application Design, 1998
<http://www.moe-lange.com/danny/patterns.pdf>

[Virdhagriswaran, Osisek, Connor, 1995]

Virdhagriswaran, S., Osisek, D. and O'Connor, P. Standardizing Agent Technology. In: ACM Standard View, 1995, Volume 3, Number 3, pages 96-101.

[FIPA, 2002]

FIPA Abstract Architecture Specification Version XC00001K (Experimental), November 11, 2002.