Analysis of Agent-Oriented Software Engineering

Jitendra P. Dave
Assistant Professor
Department of Master of Computer Application
C.U. Shah College, Wadhwan City, Dist. Surendranagar, Gujarat-India

Amit K. Patel
Assistant Professor
Department of Master of Computer Application
C.U. Shah College, Wadhwan City, Dist. Surendranagar, Gujarat-India

Abstract

Agent-Oriented Software Engineering is the one of the most recent contributions to the field of Software Engineering. It has several benefits compared to existing development approaches, in particular the ability to let agents represent high-level abstractions of active entities in a software system. This paper gives an overview of recent research and industrial applications of both general high-level methodologies and on more specific design methodologies for industry-strength software engineering.
Keywords: Intelligent Agents, Software Engineering, UML, Design Patterns and Components

I. Introduction

Agent-Oriented Software Engineering is being described as a new paradigm for the research field of Software Engineering. But in order to become a new paradigm for the software industry, robust and easy-to-use methodologies and tools have to be developed. But first, let us explain what an agent is. An agent, also called a software agent or an intelligent agent, is a piece of autonomous software, the words intelligent and agent describe some of its characteristic features. Intelligent is used because the software can have certain types of behavior ("Intelligent behavior is the selection of actions based on knowledge"), and the term agent tells something about the purpose of the software. An agent is "one who is authorized to act for or in the place of another" (Merriam Webster's Dictionary)

A. *Examples of Software Agents*

1. The animated paperclip agent in Microsoft Office 2. Computer viruses (destructive agents) http://www.elcomag.com/amund/ 3. Artificial players or actors in computer games and simulations (e.g. Quake) 4. Trading and negotiation agents (e.g. the auction - agent at EBay) 5. Web spiders (collecting data to build indexes to use by a search engine, i.e. Google) A common classification scheme of agents is the weak and strong notion of agency. In the weak notion of agency, agents have their own will (autonomy), they are able to interact with each other (social ability), they respond to stimulus (reactivity), and they take initiative (pro-activity). In the strong notion of agency the weak notions of agency are preserved, in addition agents can move around (mobility), they are truthful veracity), they do what they're told to do (benevolence), and they will perform in an optimal manner to achieve goals rationality). Due to the fact that existing agents have more in common with software than with intelligence, they will be referred to as software agents or agents in this context.

1) *Terminology*

Being a relatively new research field, agent-based software engineering currently has a set of closely related terms used in research papers, I will thus try to clarify and explain the terms and their relations below. Agent-Oriented Programming (AOP) is seen as an improvement and extension of Object- Oriented Programming (OOP). Since the word "Programming" is attached, it means that both concepts are close to the programming language and implementation level. The term "Agent-Oriented Programming" was introduced by Shoham in 1993- Agent-Oriented Development (AOD) is an extension of Object-Oriented Development (OOD). The word "Development" is sometimes interpreted as "Programming", on the other hand it is frequently interpreted to include the full development process that covers the requirement specification and design, in addition to the programming itself.

2) *Scope and Limitations*

In this paper we will present a topical overview of recent advances of methodologies for development of agent-based systems. The focus is both on General high-level methodologies and on more Specific design methodologies related to software Engineering. This means that specialized agent Methodologies, e.g. to improve coordination, cooperation, communication and artificial intelligence in agents and agent systems, are outside the scope of this paper. Suggested readings that give good Overviews of other aspects of the agent research Field is presented in the work by Jennings et al. And by Nwana et al. This paper is organized as follows: section 2describes aspect of Agent-Oriented Software Engineering, section 3 gives a description of high-level methodologies, section 4 describes design methods inspired by well-known software engineering methods and standards (e.g. UML, components and design patterns), section 5 describes problems, methodologies and tools for agents in industrial context.

II. Agent-oriented software engineering

The main purposes of Agent-Oriented Software Engineering are to create methodologies and tools that enables inexpensive development and maintenance of agent-based software. In addition, the software should be flexible, easy-to-use, scalable and of high quality. In other words quite similar to the research issues of other branches of software engineering, e.g. object-oriented software engineering.

A. How are agents distinguished from objects?

Agent-oriented programming (AOP) can be seen as an extension of object-oriented programming- (OOP), OOP on the other hand can be seen as a successor of structured programming. In OOP the main entity is the object. An object is a logical combination of data structures and their corresponding methods (functions). Objects are successfully being used as abstractions for passive entities (e.g. a house) in the real-world, and agents are regarded as a possible successor of objects since they can improve the abstractions of active entities. Agents are similar to objects, but they also support structures for representing mental components, i.e. beliefs and commitments.

In addition, agents support high-level interaction (using agent-communication languages) between agents based on the "speech act" theory as opposed to ad-hoc messages frequently used between objects examples of such languages are FIPA ACL and KQML

B. Can agents solve all software problems?

Since this is a new and rapidly growing filed, there is a danger that researchers become overly optimistic regarding the abilities of agent-oriented software engineering. Wooldridge and Jennings discuss the potential pitfalls of agent-oriented software engineering .They have classified pitfalls in five groups: political, conceptual, analysis and design, agent-level, and society-level pitfalls. Political pitfalls can occur if the concept of agents is oversold or sought applied as a universal solution. Conceptual pitfalls may occur if the developer forgets that agents are software, in fact multithreaded software. Analysis and design pitfalls may occur if the developer ignores related technology, e.g. other software engineering methodologies. Agent-level pitfalls may occur if the developer tries to use too much or too little artificial intelligence in the agent-system. And finally, society-level pitfalls can occur if the developer sees agents everywhere or applies too few agents in the agent-system.

III. High-level Methodologies

This section describes methodologies that provide a top-down and iterative approach towards modeling and developing agent-based systems.

A. The Gaia Methodology

Wooldridge, Jennings and Kinney present the Gaia methodology for agent-oriented analysis and design. Gaia is a general methodology that supports both the micro-level (agent structure) and macro-level (agent society and organization structure) of agent development, it is however no "silver bullet" approach since it requires that inter-agent relationships (organization) and agent abilities are static at run-time. The motivation behind Gaia is that existing methodologies fail to represent the autonomous and problem-solving nature of agents; they also fail to model agents' ways of performing interactions and creating organizations. Using Gaia, software designers can systematically develop an implementation-ready design based on system requirements. The first step in the Gaia analysis process is to find the roles in the system, and the second is to model interactions between the roles found. Roles consist of four attributes: responsibilities, permissions, activities and protocols. Responsibilities are of two types: liveness properties - the role has to add something good to the system, and safety properties- prevent and disallow that something bad happens to the system. Permissions represents what the roles allowed to do, in particular, which information it is allowed to access. Activities are tasks that a role performs without interacting with other roles.

Protocols are the specific patterns of interaction, e.g. a seller role can support different auction protocols, e.g. "English auction". Gaia has formal operators and templates for representing roles and their belonging attributes, it also has schemas that can be used for the representation of interactions. In the Gaia design process, the first step is to map roles into agent types, and then to create the right number of agent instances of each type.

The second step is to determine the services model needed to fulfill a role in one or several agents, and the final step is to be create the acquaintance model for the representation of communication between the agents. Due to the mentioned restrictions of Gaia, it is of less value in the open and unpredictable domain of Internet applications, on the other hand it has been proven as a good approach for developing closed domain agent-systems. As a result of the domain restrictions of the Gaia method, Zambonelli, Jennings et al. proposes some extensions and- improvements of it with the purpose of supporting development of Internet applications. Other sources for the discussion of micro and macro aspects of agent modeling include work by Chaib-draa.

B. The Multiagent Systems Engineering Methodology

Wood and DeLoach suggest the Multiagent Systems Engineering Methodology (MaSE). MaSE is similar to Gaia with respect to generality and the application domain supported, but in addition MaSE- goes further regarding support for automatic code creation through the MaSE tool. The motivation behind MaSE is the current lack of proven methodology and industrial-strength toolkits for creating agent-based systems. The goal of MaSE is to lead the designer from the initial system specification to the implemented agent system. Domain restrictions of MaSE is similar to those of Gaia's, but in addition it requires that agent-interactions are one-to-one and not multicast.

C. *Modeling Database Information Systems*

Wagner suggests the Agent-Object Relationship (AOR) modeling approach in the design of information systems. AOR is inspired by the two widely applied models of databases, i.e. the Entity- Relationship (ER) meta-model and the Relational Database (RDB) model. The purpose of the ER meta-model is to ease the transformation of relations between different types of data (entities) into an implementation-ready (database) information system design. This transformation is well-supported for static entities or objects, but falls short in modeling active entities or agents in an information system; the purpose of the AOR-model is to extend the ER-model by providing the ability to model relations between agents in addition to static entities. In AOR, entities can be of six types: agents, events, actions, commitments, claims and objects.

Commitments and claims are dualistic, commitments of one agent are seen as claims against other agents. Organizations are modeled as a group of sub-agents. Each of the sub-agents has the right to perform certain actions, but they are also committed to duties such as monitoring claims and events relevant for the agent-organization. The interpretation of duties and permissions seems to correspond with services and permissions found in the Gaia methodology an example of an agent-based database information system can be found in Magnanelli et.al.

IV. DESIGN METHODS

This section describes methodologies that are mainly inspired by the methodologies and standards of the object--oriented software engineering field.

A. *UML(Universal Modeling Language)*

The Universal Modeling Language (UML) is a graphical representation language originally developed to standardize the design of object classes. It has later been greatly extended with support for designing sequences, components etc.,

In fact all parts of an object-oriented information system design. Yim et al. suggest an architecture-centric design method for multi-agent systems. The method is based on standard extensions of UML using on the Object Constraints Language (OCL), and it supports the transformation of agent-oriented modeling problems into object-oriented modeling problems. In the transformation process, relations between agents are transformed to design patterns, these patterns are then used as relations between object classes, in contrast to the more commonly applied relation types between object classes such as inheritance. The result of this method is that designers and developers are able to use existing.

UML-based tools in addition to knowledge and experience from developing object-oriented systems. Odell, Parunak and Bauer suggested a three layer representation of Agent-Interaction Protocols (AIP). AIP are defined as patterns representing both the message communication between agents, and to the corresponding constraints on the content of such messages. In contrast to Yim et al.'s UML-based architecture, Odell et al.'s approach requires changes of the UML visual language and not only the expressed semantics. The representation requires changes of the following UML representations: packages, templates, sequence diagrams, collaboration diagrams, activity diagrams and state charts. In the first layer, the communication protocol (i.e. type of interaction) is represented in a reusable manner applying UML packages and templates.

The second layer represents interactions (i.e. which type of agents can communicate with whom) between agents using sequence, collaboration and activity diagrams as well as state charts. In the third layer, the internal agent processing (i.e. why and how the agent acts) is represented using activity diagrams and state charts. Parunak and Odell combine existing organizational models for agents in a UML-based framework in order to model and represent social structures in UML. This work is an improvement oo the Agent UML extensions to UML.

B. *Design Patterns*

Design patterns are reoccurring patterns of programming code or components software architecture. Aridor and Lange suggest a classification scheme for design patterns in a mobile agent context. In addition they suggest patterns belonging to each the classes. The purpose is to increasere-use and quality of code and at the same time reduce the effort of development of mobile agent systems. The classification scheme has three classes: traveling, task and interaction. Patterns in the traveling class specify features for agents that move between various environments, e.g. the forwarding pattern that specifies how newly agents arrived can be forwarded to another host. Patterns of the task class specify how agents can perform tasks, e.g. the plan pattern specifies how multiple tasks can be performed on multiple hosts. Patterns of the interaction class, specify how agents can communicate and cooperate. An example of an interaction class pattern is the facilitator, it defines an agent that provides services for identifying and finding agents with specific capabilities. Other approaches for design patterns for mobile- agents include the approach of Rana and Biancheri applying Petri Nets to model the meeting pattern of mobile agents.

Compared to the previously mentioned pattern classification scheme in the work by Aridor and Lange, the layered architecture has a similar logica grouping of patterns. The mobility layer together with the translation layer corresponds to the class of traveling, the collaboration layer corresponds to the class of interaction, and the actions layer corresponds to the class of task. The main difference betweenthis and the previously mentioned approaches for mobile agents, is that this one aims to cover all main types of agent design patterns.

C. *Components*

Components are logical groups of related objects that can provide certain functionalities. This might sound quite similar to agents, but in fact components are not autonomous as opposed to agents. By grouping related objects, components allow more coarse-grained re-use than the combination of single classes from scratch, this has shown to an effective and popular development approach in the software industry. Erol, Lang and Levy suggest a three-tier architecture that enables composition of agents by applying reusable components. The first tier is interactions, it is built up by agent roles and utterances. The second tier is local information and expertise that enables the storage of information such as execution state, plan and constraints of the agent. Information content, the third tier, is passive and often domain specific, since it is often used to wrap legacy systems, e.g. a mainframe database application.

V. AGENTS IN THE REAL-WORLD

The agent-oriented approach is increasingly being applied in industrial applications, but it is far from as widespread as the object-oriented approach. This section describes where and how agents have been applied with success in the manufacturing industry. Parunak defines agenthood, a taxonomy and a maturity metric in an industrial context.

His purpose is to improve the understanding and utilization of agent-oriented software engineering in industry. Agenthood, i.e. agent-oriented programming, is explained as an iterative improvement of the industry-strength methodology of object-oriented programming. The taxonomy classifies agent systems as belonging to one of the following environments: digital (i.e. software and digital hardware), social (involving human users) or electromechanical (non-digital hardware, e.g. a motor). Thereafter the taxonomy classifies agents according to the interface they support. Interface types are similar to the environments digital (e.g. communication portals), social (e.g. user interfaces) and electromechanical (e.g. motor control interfaces).Few business users, as opposed to researchers, are early-adapters of new and immature technology, as a result of this a maturity metric of agent-based systems is developed to be able to measure the level of agent technology and systems. The maturity metric has six degrees ranging from modeled applications to products. Modeled applications, the least mature, are theoretical applications in the form of architectural descriptions or analyses.

The metric continues with emulated applications that are relatively immature due to the fact that they are simulations in a lab environment. Prototype applications represent the next maturity degree, they run in a noncommercial environment but on real hardware. Pilot applications are relatively mature applications, however they are not expected to be completely bugfree,and after a certain period they usually become more mature and become production applications. A production application is being applied in several businesses, but they require support for installationand maintenance. The most mature applications are products, they are usually shrink-wrappedand sold over desk, and they can usually be installedand maintained by the non-expert user.

A. *Agents in the industry - where and how?*

Parunak presents a review of industrial agent applications. Application areas considered are: manufacturing scheduling, control, collaboration and agent simulation. Thereafter tools, methodologies, insights and problems for development of agent systems are presented and discussed. Manufacturing scheduling is the ordering and timing of processes in a production facility. The purpose is to optimize the production by maximizing the number of units produced per time slot and keep good quality of the product, and minimize the resource requirements per unit and the risk of failures. Processes and machinery has to be controlled in order to operate as scheduled.

The control can range from simple regulation of the power level for a piece of machinery to advanced real-time cybernetic control of processes. For many industries, human collaboration is needed to solve complex problems, e.g. in a design process - engineers and designers have to collaborate in order to guarantee that products are pleasant to look in addition to being safe.

B. *Agent Methodologies in the Industry*

Methodologies for creating industrial agent systems presented are Rockwell’s Foundation Technology and DaimlerChrysler’s Agent Design for agent based control .In Rockwell’s Foundation Technology four issues are considered in the development of agent-based control architectures, the first is flexibility related to fault-tolerance in a multi-objective environment, the second is self-configuration for the support of new products and rapidly changing old ones, without much manual reconfiguration, the third is productivity - how to at least maintain and hopefully improve productivity by applying agents, and the final issue is equipment life span cost - how to keep the agent in sync with life-cycle costs of the operating equipment.

VI. CONCLUSION

This paper has sought to give a topical overview of recent progress of agent-oriented software engineering methodologies. Further work should include a more thorough analysis of the field in addition to practical testing of and experiments with the methods

REFERENCES

[1] Aridor Y. and Lange D. B. Agent Design Patterns: Elements of Agent Application Design. In Proc. Of the second international conference on Autonomous agents, pages 108–115, 1998.
[2] Chaib-draa B. Connection between micro and macro aspects of agent modeling. In Proc. of the first international conference on Autonomous agents, pages 262–267, 1997.
[3] DeLoach S. A. Multiagent Systems Engineering- A Methodology and Language for Designing Agent Systems. In Proc. of Agent Oriented Information Systems, pages 45–57, 1999.
[4] Labrou Y., Finin T. and Peng Y. Agent Communication Languages: The Current Landscape. IEEE Intelligent Systems, 14(2), March/April 1999 1999.
[5] Lind J. Issues in Agent-Oriented Software Engineering. The First International Workshop on Agent- Oriented Software Engineering (AOSE-2000), 2000.