Agent-Oriented Software Engineering: A Comparison of Agent and Non-Agent Version of a Cluster Server

Andraž Bežek, Matjaž Gams
Department of Intelligent Systems, Jozef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia
email: {andraz.bezek, matjaz.gams}@ijs.si

ABSTRACT

The agent-oriented software engineering research community is advocating the agent-based approach for the design of distributed systems. But currently, there is a substantial gap between the massive number of publications reporting advantages of agents, and the relatively small number of successful large applications. We present an agent-based cluster server application together with the analysis of advantages and disadvantages of the agent-based version compared to the non agent-based version of functionally similar cluster server application. Obtained experience is analyzed from the viewpoints of designers, developers and users. Observations confirm several previously published advantages of the agent-based approach and implementation; yet reveal some less familiar disadvantages as well. Overall, we argue that agent-based approach provide reasonable benefits already at this stage, while we lack further agent-based research.

Categories and Subject Descriptors
I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence – multiagent systems.

General Terms: Design.

Keywords: Agent-oriented software engineering, multi-agent systems, load-balancing.

1. INTRODUCTION

It is claimed that agent-based software engineering can overcome two important problems of previous approaches: a) too rigidly defined interaction between entities, and b) insufficient mechanisms for representing organizational structure of the system. But, there is a substantial gap between the massive number of publications advocating advantages of agents and the relatively small number of successful large applications. Practical experiences are not overwhelmingly consistent with theoretical publications. In this paper we present our experiences with the agent-based approach for a particular application of a cluster server. The major contributions of the paper are analyses of beneficial and undesired properties of the agent software engineering as proposed in [2] [4].

2. AGENT-BASED CLUSTER SERVER

A cluster server (CS) is an Internet based load-balancing application. It balances incoming IP traffic among servers in a cluster. The fault-tolerant operation is achieved by distributing load-balancing functionality among many computers. The nature of IP routing requires only one computer to perform load-balancing; therefore others can monitor its execution. In case of a failure, software on remaining computers will vote for the new active load-balancing computer, which will take over all functionality of the failed load-balancing computer. The active computer must periodically check servers and services on servers for proper operation. If a malfunction is detected, the computer or service is removed from the cluster. According to configuration, a computer in a cluster can be server, load-balancer or both.

Previous versions of CS, many of them awarded [1][3], were designed as a typical distributed system. The distributed and asynchronous nature of software introduced many execution threads. The synchronization of multiple activities among modules proved to be a daunting task. The design required a developer to consider all synchronization problems and therefore all module dependencies. On the other hand, the size of the software obscured this approach. Only a couple of developers comprehended the whole structure. The direct consequence was the inability to divide development among many developers, resulting in inefficient task decomposition. Head developers were over-occupied while other developers could not adequately help.

We investigated ways to overcome obstacles observed during the development of CS 6. The big and complex structure was inappropriate for extensions so an obvious solution was to partition the big program into smaller parts and apply agents. Although the decision to completely rewrite the system into a multi-agent version was not an easy one, it was chosen as the best option.

The design of CS 7 was in several ways a straightforward process: big conceptual parts were divided into conceptually smaller components. The difference emerged as we reversed this process and combined functionally connected small components into agents. The amount of interaction among components guided the choice what is an agent and what not. According to [2], interactions within agents are by an order of magnitude more common than interactions among agents. As a result, we defined 11 types of agents, and 21 different agent instances. During the design process, we have identified three distinct agent types. They are based on the type of interaction and are summarized bellow:
- **Agent-reactive**: Such agents reply only on demand by other agents. Their reactive structure makes the implementation to be fairly straightforward.

- **System-reactive**: These agents use information gained from other agents, but they do not provide any. Their main task is to monitor agent-system and to react accordingly to their goals. Possible concurrent actions must be serialized or synchronized.

- **Collaborative**: The most advanced type of all agents. They must synchronize their actions according to requests and replies from the agent system. Consequently, their actions can be cancelled or started depending on actions of other agents.

The important property of a multi-agent system is decentralized decision logic. This enabled developers to be evenly distributed among different implementation tasks, which showed to be smaller and easier to implement compared to those in CS 6. Compared to non-agent approach, the distribution of work among developers showed to be major benefit; resulting in faster development process, which took overall approximately 300 man-months of work.

2. **COMPARISON BETWEEN THE AGENT AND NON-AGENT VERSION**

The comparison between the two systems, i.e. the two approaches, is analyzed through the eyes of a designer, a developer and a user. Opinions were constructed based on discussion between 5 designers during 1.5 years, 20 developers during 1.5 years and 10 users – beta-testers during 6 months.

2.1 **Designer’s View**

Development of CS 7, as opposed to CS 6, incorporated the process of combining similar components into agents. This process added an additional layer of abstraction and introduced more hierarchical and advanced structural design, resulting in improved comprehensibility and ease of cooperation. One of the biggest advantages was observed in terms of dividing design issues among different agent designers. Each design task (e.g.: design of a single agent) was smaller and less interconnected than comparable task in non-agent version. The lower amount of dependencies between agents, compared to a non-agent approach, is a consequence of agent design process, which was based on the amount of interaction between components as described in Section 2. To summarize, during the design of CS 7 we have identified the following advantages: design of single agents can be divided among many designers, agents introduce new design abstraction layers, it is possible to introduce agents that were not predicted in the design, the system design is easier to understand, and decentralization reduces system complexity. The identified disadvantages are also important. Due to its distributed nature, it is not possible to exactly predict system interaction. Consequently, there is no guarantee about system correctness, although this is a common property of many distributed systems.

2.2 **Developer’s View**

Developer’s work is often seen as a straightforward process, but it is not. The ingenuity of a developer can significantly speed up implementation cycle thus saving time and money. The right tools help developer achieve ever-tight deadlines. Although the agents are smaller in terms of design specification it showed that the implementation was not so obvious. The possibility of response failure added additional synchronization mechanisms. We also noticed that developers cooperate on the basis of agent-to-agent interaction, thus breaking rigid organizational ties. Final analysis of advantages showed: developer must satisfy smaller goals based on local knowledge, there may be different solutions for the same problems, developers cooperate on an agent-to-agent interaction basis. The reported disadvantages were: developers must predict failed or multiple responses, debugging of distributed systems is a daunting task, and currently there is a lack of agent tools and languages that can support the agent-oriented programming.

2.3 **User’s View**

A typical user for CS 7 is a system administrator. Although system administrators are technically oriented and above average in terms of accepting technical improvements, it does not automatically mean that they gladly accept any new approach. Novelties are often seen as dangerous in general, especially in distributed environments, which are often considered as not very reliable. On the other hand, successful new approaches are generally well accepted in time. Analysis showed the following advantages: agents enhance robustness, and agents enable minimal system interruption between upgrading, but introduced disadvantages as: agents consume more computer resources, and high number of programs - agents - is confusing.

3. **CONCLUSION**

The intent of this paper was to present our experience with a large software project designed with the agent-oriented software engineering, resulting in a multi-agent system. In addition, we wanted to verify two claims: a) agent-based software design and implementation has the potential to significantly improve the way humans model, design and build complex, distributed software systems; and b) the agent-based approach is a potential new mainstream software engineering paradigm.

Through the viewpoints of a designer, developer and user we have identified some advantages and disadvantages. Despite a relatively big list of disadvantages and an early stage of agent-oriented software engineering, our experience with a 25 man-year project clearly indicates that the agent-oriented approach indeed enables important improvements compared to more classical software engineering approaches.

4. **REFERENCES**

[1] Corporate IT Best Product winner for enterprise-class customers, IT 2000 Sydney Conference

