Agent-Oriented Software Engineering with MASSIVE

Jürgen Lind
Ackerstraße 1
81541 München, Germany
jli@agentlab.de

Betreuer: Prof. Dr. Jörg Siekmann
Art der Arbeit: Dissertation
Fachbereich GI: 1

1 Introduction

Although agents and agent-based computing have been around for some time it is only until now that these topics begin to gain attendance beyond the research community; it is being discovered by non-researches that agent technology is a powerful tool for the development of large and complex systems. These days, typical software architectures contain many dynamically interacting components, each with their own thread of control and engaging in complex coordination protocols [4]. Therefore, a new programming metaphor that captures these systems is needed. Although the basic structural elements of the agent-based approach as well as their connections are not yet fully understood, it nonetheless seems to be a promising means for dealing with these highly complex systems.
However, although the agent-oriented view is likely to become a major means to describe complex software systems, development methods, i.e. methods that provide guidelines how to build actual agent-based applications or multiagent systems, are still in their infancy and must be further advanced to establish the technology in an industrial context. As stated in [11]: "Relatively less attention has been paid to the important question of the process that designers go through. Industrial users will use agents more readily if basic principles and guidelines are available . . .". The strong need for agent and agent systems development methods in industry is exemplified by the major European telecommunication companies that have launched a joint research project to foster the definition of a development method for agent applications [5]. Furthermore, the engineering of agents and multiagent systems is not only a technical matter that has been picked up by industry, it is also an interesting research field that can provide new methods and techniques for a better understanding and modeling of highly complex systems [7].

2 Pragmatic Software Development

Most of the development methods for multiagent systems are *normative* models. They define – from the (subjective) point of view of the respective author – which activities make up the software engineering task, the temporal ordering of these activities, the involved products and often also the notation that must be used. However, the problem with this approach is clearly pointed out in the following quote from [14]: "One cannot standardize thinking and one should not even attempt to do so."
Furthermore, this fact usually limits the applicability of a particular method whenever there is a deviation of the prerequisites of an actual project and the prerequisites of the method to be used. In short, these methods focus on software engineering processes as they *should be*, not as they *are really like* [3]. Therefore, it seems to be a more promising idea to follow the suggestion from [6] to ". . . concentrate on what it seems that people really do, not on what they might do if they were perfect.". This approach can lead to more *pragmatic* software engineering models that are based on an explicit cognitive model that supports individualized strategies and notations.
The **M**ulti**A**gent **S**ystem**S** **I**terative **V**iew **E**ngineering (MASSIVE, [9], [1]) development method is a pragmatic framework for the development of multiagent systems that accounts for the special requirements mentioned above. It combines novel ideas in software development with standard software engineering techniques that have been adapted to the special context of multiagent systems. Furthermore, MASSIVE features an organizational framework that supports learning over project boundaries.

3 MASSIVE Views

The terminological framework that is used in the MASSIVE method is based on so-called *views*. A view represents a set of conceptually linked features, ie. a view is a projection of the complete design onto a particular subject. During the development of the system design, the designer will soon discover that some design aspects are more closely related to one another then to other aspects. These "natural" collections of aspects seem to be characteristic for a particular class of applications and it therefore suggests itself that these empirical decompositions capture the nature of an application class quite well. For these reasons, my personal idea of a view is therefore to see a view as an collection of features that cross-cut functional boundaries as well as different representations.

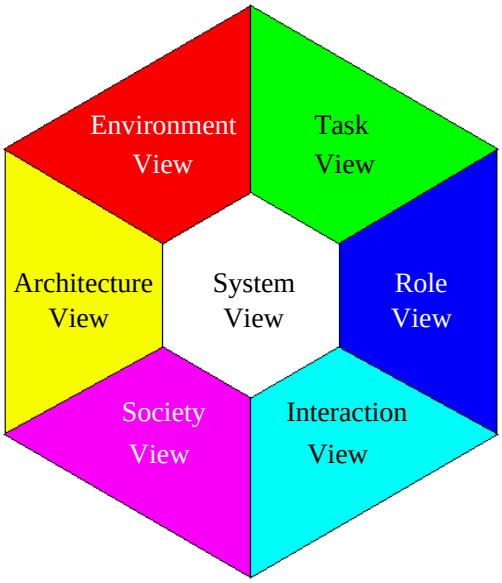

Figure 1: MASSIVE Views

Ideally, the target system can be decomposed in several independent views with well defined interfaces. In MASSIVE, each view represents a set of conceptually linked features, a view is thus a projection of the design onto a particular subject. This interpretation is supported by results from programming experiments in cognitive psychology were it was found that different abstractions (views) are used for different sub-tasks of the software development process [12]. A collection of views that achieves a logical decomposition of the target system is called a *view system*, the interfaces between the views are modeled as explicit connections between views by sharing features between the connected views. These shared features belong to two or more views at the same time and represent the interface between their parent views.

Views are general concept to arrange features with respect to a logical decomposition of the application class eg. multiagent systems. The view system that I shall present now has matured over several years during the work on the design and implementation of multiagent systems. The system as shown in Figure 1 consists of seven views that I will describe briefly in the following paragraphs.

Environment view In this view, the environment of the target system is analyzed form the developers perspective and from the systems perspective. These two points-of-view usually differ greatly as the developer usually has global knowledge whereas the system has only local knowledge. In the Robocup domain, for example, the developer has access to the complete state of system and its environment and this state is completely deterministic from this point-of-view. For the perspective of the individual agent, on the other hand, only parts of the environment are accessible and the state transitions appear to be nondeterministic because of the ongoing activities that cannot be perceived by the agent.

Task view In the Task view, the functional aspects of the target system are analyzed and a task hierarchy is generated that is used to determine the basic problem solving capabilities of the entities in the final system. Furthermore, the nonfunctional requirements are defined and quantified as far as possible. Note that this view does not assume that a multiagent approach is used for the final system and therefore provides a rather high-level analysis of the problem in question.

In the case of a compiler application, for example, the basic functional requirement is that the system translates a program specified in a high-level language to a particular assembly language. The quality of the resulting implementation or the maximal tolerable time for the compilation are nonfunctional requirements and the basic problem solving capabilities are for example lexical analysis or code generation.

Role view This view determines the the functional aggregation of the basic problem solving capabilities according to the physical constraints of the target system. A role is an abstraction that links the domain dependent part of the application with the agent technology that is used to solve the problem under consideration. In my view, an agent consists of one or more role descriptions and an architecture that is capable of executing these role models which makes it important to aggregate the basic capabilities according to physical constraints.

In a robotics application for a storage area, for example, one may find robots that are capable of carrying containers from one area to another and others that are capable of stacking containers onto each other. Therefore, the roles of “carrier” and “stacker” cannot be assigned to a single agent because of the physical constraints of the robots unless a third sort of robots exists that can execute both basic problem solving capabilities.

Interaction view Interaction is the fundamental concept within a system that is comprised of multiple independent entities that must coordinate themselves in order to achieve their individual as well as their joint goals. 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 that can be instantiated in a wide variety of

contexts are discussed and the developer is encouraged to analyze the target problem with respect to the applicability of these generic forms before designing new forms.
The most popular examples for interaction are of course communication protocols simply because they have been studied for quite some time. However, multiagent systems that simulated a physical environment or even real physical multiagent systems contain several other necessities and possibilities of interaction then communication and these forms of interaction must be considered in a general purpose method as well.
Society view A society is a structured collection of entities. The goal of this view is 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 developer. According to this classification and well defined quality measures for the performance of the target society that depend on application specific aspects, a society model is developed that is consistent with the roles within the society and that achieves the defined goals.
To illustrate how the applied quality measure affects the desirable society structure, consider the example of Internet trading. To achieve the best trade, the number of participants in the trading process should be rather high in order to increase the chance of finding a profitable trade. On the other hand, a high number of participants also increases the computational and communicational overhead and thus a clustering of trading agents would increase the communicational and computational efficiency of the system. The final structure of the agent society (flat or clustered) thus depends on the precedence of the quality measures (quality of the solution vs. efficiency) that is taken by the system designer.
Architectural view The architectural 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 – due to the size and complexity of this particular aspect – the agent architecture. The system architecture is described according to various aspects and includes things such as the agent management or database integration. The required agent architecture is characterized according to the requirements of the problem to be solved; it is strongly recommended that the system developer should at first try to select one of the numerous existing architectures before trying to develop a new architecture from scratch.
An important aspect that has to be dealt with in this view is to find the appropriate segregation between agents and objects. Just because agents provide a means for structuring a problem does not mean that they are necessarily the best means to do so. Sometimes, it is a better idea to implement particular abstractions as ordinary objects and therewith increase the system performance by avoiding the inevitable overhead that comes with turning an object into an agent.
System view This view, finally, deals with aspects that affect several of the other views at the same time or even the system as a whole. The System view, for example, handles the user interface that controls the interaction between the system and the user(s) on a task specific as well as the task independent level. Generally speaking, the task specific aspects are usually the input specification and the output presentation whereas the task independent aspects deal with the visualization of the system activities in order to enable the user to follow the ongoing computations and interactions. Other aspects that are treated in this view are the system-wide error-handling strategy, performance engineering and the system deployment once it has been developed.
In this section, I have presented the product model of the MASSIVE method which is the core of the entire approach. In the next section, I will discuss a generic process model that can be used to construct the product model described above.

4 Iterative View Engineering

Iterative view engineering is a product centered software engineering process model that combines Round-trip Engineering and Iterative Enhancement. Round-trip engineering is itself a combined method that consists of a generative step where the goal is to construct a software system from its specification and an analytical step where the starting point is existing code and the goal is the specification. The first step is called *Forward Engineering* and the second step is called *Reverse Engineering*.
Iterative Enhancement is a process model that operates in several cycles on a partitioned and incomplete system model. The incomplete design is iteratively enhanced (hence the name of the method) in each cycle. The major advantage of this approach is that it takes into account that any specification is initially incomplete and usually undergoes frequent changes because of upcoming new information, external factors or simply because of human errors. Although the idea of Iterative Enhancement is pretty old in terms of computer science research, it is still developed further and used in software development methods such as described in [2]. MASSIVE proposes an extension of basic idea that uses Iterative Enhancement as the *macro* process that encapsulates several *micro* processes that are used to construct individual parts of the model.
Combining these ideas in a joint process model results in the Iterative View Engineering model shown in Figure 2. The Round-trip steps are oriented along the the horizontal axis of the figure and the Iterative Enhancement steps are depicted on either side of the figure.
The Iterative view engineering process model is the following: Initially, the design model and the implementation are empty. In a first cycle of iterative enhancement (①), the designer specifies the first version of the design probably by using different micro models for each view, note that the enhancement step operates on the complete set of views. In the following step (②), parts of the the initial design are implemented and if an error occurs during the construction phase, the design has to be refined (③) until it can be implemented. Next, the initial implementation is tested. During

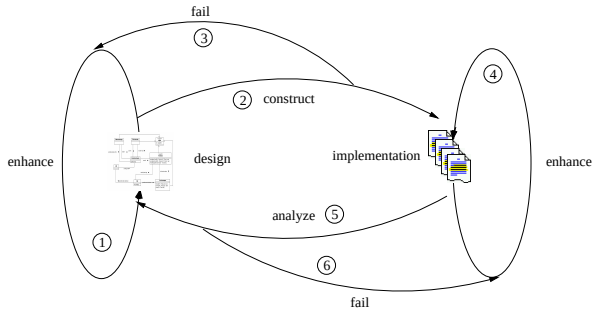

Figure 2: Iterative view engineering

the test phase, it often turns out that the design specification was incomplete (even for the parts that were generated) or wrong in which case the implementation is changed or enhanced (④) in order to meet the intentions of the initial model. After the test and enhancement phase of the implementation is complete, the results must be integrated into the design model in an analysis step (⑤). If the implementation cannot be re-engineered (eg. because the expressive power of the modeling language is too limited for a particular feature of the implementation) the implementation must be changed accordingly (⑥). After this step, the next cycle is executed until the entire system is fully implemented.

5 Conclusion

The approach presented in this paper has evolved over several years and it has been successfully applied and refined in different types of multiagent systems:

- The multiagent solution [10] for the Train Coupling- and Sharing (TCS) approach is a system for scheduling and cost optimization of a large number of railroad transportation tasks using novel railroad technologies.
- The MoTiV/PTA is a project concerned with the design of a device for individual travelers according to the FIPA standards. One of the services required in the PTA domain is intermodal route planning for which a multiagent solution has been implemented [13].
- The TEAMWORK LIBRARY [8] is a framework for distributed search that was originally developed for equational theorem proving but that was also used in other application fields such as the computation of Groebner Bases or the Traveling Salesman Problem.

The above list illustrates that the MASSIVE method is applicable in a wide variety of problem domains and that it has been successfully used in different domains.

References

[1] agentlab. `http://www.agentlab.de`.
[2] G. Booch. *Object Solutions: Managing the Object-Oriented Project*. Addison-Wesley, 1996.
[3] J. M. Carroll and M. B. Rosson. Usability specifications as a tool in iterative development. In *Advances in Human-Computer Interaction*. Norwwod, 1985.
[4] P. Ciancarini and M. Wooldridge, editors. *Agent-Oriented Software Engineering*, number 1957 in LNAI, 2001.
[5] EURESCOM. MESSAGE: Methodology for Engineering Systems of Software AGEnts (P-907), 1999. `http://www.eurescom.de`.
[6] T. R. G. Green. Programming languages as information structures. In *Psychology of Programming*. Academic Press Ltd., London, 1990.
[7] N. R. Jennings, K. P. Sycara, and M. Wooldridge. A roadmap of agent research and development. *JAAMAS*, 1(1), 1998.
[8] J. Lind. TWLib – A Generic Library for TEAMWORK Applications. Master's thesis, University of Kaiserslautern, 1996.
[9] J. Lind. *Iterative Software Engineering for Multiagent Systems - The* MASSIVE *Method*, volume 1994 of *Lecture Notes in Computer Science*. Springer, May 2001.
[10] J. Lind, K. Fischer, J. Böcker, and B. Zirkler. Transportation Scheduling and Simulation in a Railroad Scenario: A Multi-Agent Approach. In *PAAM99*, 1999.
[11] H. V. Parunak. *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*, chapter 9. MIT Press, 1999.
[12] N. Pennington and B. Grabowski. The tasks of programming. In *Psychology of Programming*. Academic Press Ltd., London, 1990.
[13] Siemens AG. Method and device for determining a route from the originating point to the termination point. International Patent WO 99/20981, 1999.
[14] H. Zemanek. Formal definition the hard way. In *Formal Models in Programming*. Elsevier, 1985.