A Development Method for Multiagent Systems

Jürgen Lind
German Research Center for AI (DFKI)
Im Stadtwald, B36
66123 Saarbrücken, Germany
lind@dfki.de

Abstract

In this paper, we propose a pragmatic process model for the development of multiagent systems based on the combination of standard software engineering techniques. The development method that is presented in this paper features a product model to describe the target system, a process model to construct the product model and an institutional framework that supports learning and reuse over project boundaries. The approach has evolved over several years and it has been successfully applied and refined in various industrial research projects.

1 Introduction

It is a widely agreed fact in the Software Engineering Community that any software life cycle, process or product model must be tailored towards the characteristic needs of the application domain of the target system [Basili *et al.*, 1994] as there is no "silver bullet" [Brooks, 1986], ie. a method that can be used for all types of problems.

The strong need for agents and agent systems development methods in an industrial context 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 [EURESCOM, 1999]. Furthermore, the engineering topic of agents and multiagent systems is not only a technical matter that has been picked up by industry, it is also a interesting research field that can provide new methods and techniques for a better understanding and modeling of these highly complex systems [Jennings *et al.*, 1998].

Recently, several development methods for multiagent system have been proposed [Iglesias *et al.*, 1998], the methods that are closest related to the work presented in this paper are described in [Burmeister, 1996], [Brazier *et al.*, 1997] [Iglesias *et al.*, 1997], and [Wooldridge *et al.*, 1999]. However, most of these methods feature a technology-driven, model-oriented and sequential approach towards the development of multiagent applications that is not always adequate for this class of systems. Therefore, we will now outline some alternatives that are implemented in the method proposed in this paper.

Requirements-driven vs. technology-driven Most of the related development methods assume that the technology, ie. a multiagent approach, for the target system has already been set and that the system must be designed and implemented according to this technology. In a requirements-driven development approach, the requirements of the target system determine the technology that is to be used for the system design and implementation and not vice-versa. Thus, a requirements-driven approach starts earlier in the life cycle of the target system when the fundamental characteristics of the problem under consideration are analyzed. The only approach that partly adopts this idea is the DESIRE method [Brazier *et al.*, 1997] that starts with the construction of a task hierarchy of the problem domain. However, even in this approach, the development of the task hierarchy assumes a multiagent solution for the final system. In a more rigorous requirements-driven approach, the task hierarchy will be built without a particular technology in mind and after the task hierarchy is constructed, objective decision functions are then applied to it in order to find the most appropriate technologies. However, the basic difficulty of this idea is that until now, no such decision functions exist and thus a subject to further research in the multiagent community.

View-oriented vs. model-oriented The related approaches all develop several distinct models of the target system that must be subsequently integrated in order to construct the final system. This integration process is not always trivial as the links between the models are usually neither explicitly defined nor obvious. Thus, we suggest to adopt a view-oriented approach instead. A view-oriented method deals with the system as a whole and uses different perspectives on the entire system as fundamental abstraction. The system is not decomposed but rather viewed from different angles with different foci on particular aspects. The advantage of this approach is that it avoids the integration process because the model is always consistent from any view as changes in one view are always directly propagated to other views.

Iterative vs. sequential A sequential process

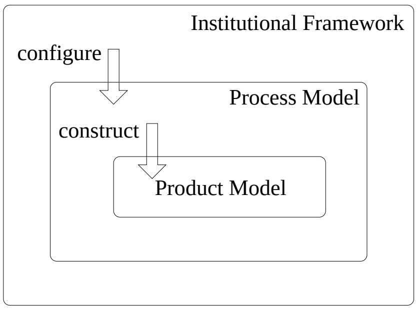

Figure 1: The Goal Framework

model has the optimistic assumption, that everything will work out as desired and that all documents are complete, correct and consistent once they are created. However, this assumption may hold in well understood problem domains for which a lot of systems have been constructed, but it is unlikely that it holds in the case of multiagent systems because this class of systems is not well understood until now. Except for [Wooldridge *et al.*, 1999], none of the related methods has an explicit iterative process model and even there, the iterative cycle is only applied to the design phase of the development process.

Still, a process model that is to be used in the context of multiagent applications should respect the fact that the system requirements and design documents are normally incomplete, incorrect and inconsistent due to the limited experience that exists with this class of applications. To overcome these difficulties, it is a good idea to iterate the processes that were applied to construct the documents and to evaluate the results of each iteration according the explicit quality measures that were defined early in the development process. Although this approach does not guarantee that the documents capture all relevant aspects in the correct way, it nonetheless reduces the risk of errors because of the repeated quality evaluations.

The basic structure of a software development method as shown in Figure 1 should provide one or more *Product model(s)* that are used to describe the design and/or the implementation of a particular system, one or more *Process model(s)* that guide the designer in constructing the product models according to the user requirements and it should also provide an *Institutional framework* that allows for the configuration and constant improvements of the underlying processes as well as for reuse of existing experience.

In this paper, we propose the **M**ultiagent systems **i**terative **v**iew **e**engineering (MASSIVE) development method as a pragmatic framework for the development of multiagent systems. MASSIVE accounts for the special requirements mentioned above and is oriented at the basic idea outlined in Figure 1. 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 institutional framework that supports learning over project boundaries.

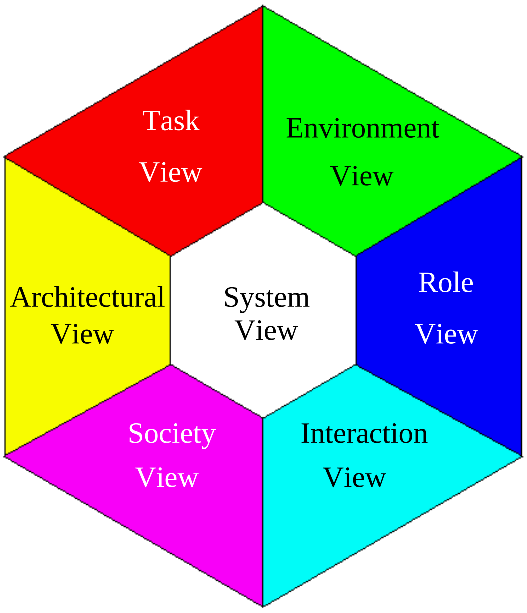

Figure 2: MASSIVE Views

2 MASSIVE Views

The terminological framework that we propose 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 model onto a particular subject. Thus a view is an abstraction of the model with respect to a specific point-of-view and usually, several abstractions from the same reality exist. A collection of views is a *view system* that achieves a *logical decomposition* of the target system.

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

Task view In the Task view, the functional aspects of the target system are analyzed and a task hierarchy is generated that is then used to determine the basic problem solving capabilities of the entities in the final system. Furthermore, the nonfunctional requirements of the target system 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 code 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.

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 [Noda, 1995], 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.

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 our 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, we 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. In order 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 [Collins and Ndumu, 1998]. 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, we have presented the product model of the MASSIVE method which is the core of the entire approach. In the next section, we will discuss a generic process model that can be used to construct

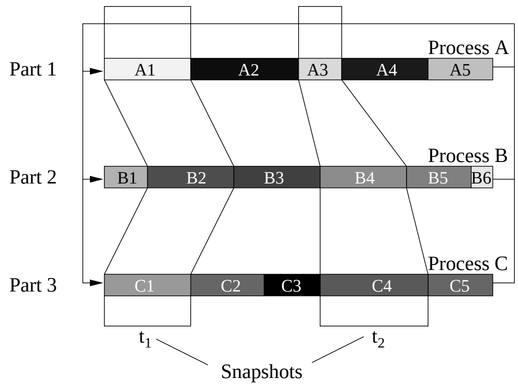

Figure 3: Micro Processes

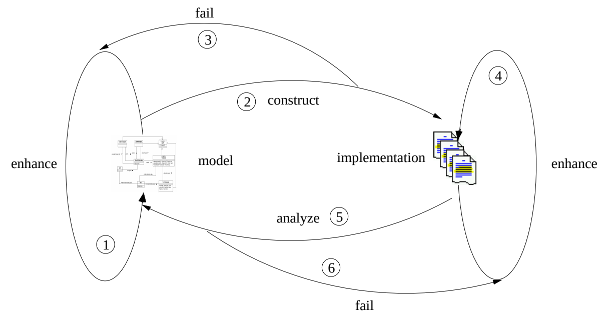

Figure 4: Iterative view engineering

the product model described above.

3 Iterative View Engineering

Iterative view engineering is a product centered software engineering process model that combines Round-trip Engineering [Balzert, 1998] and Iterative Enhancement [Basili and Turner, 1975]. 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 is similar to the Waterfall Model [Royce, 1970] except for the fact, that the Waterfall Model needs a complete and stable requirements definition while the Iterative Enhancement model operates in several cycles on a partitioned and incomplete system model. The incomplete model 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 [Parnas and Clements, 1986]. 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 [Booch, 1996].

We propose an extension of basic idea that sees the Iterative Enhancement as the *macro* process that encapsulates several *micro* processes that are used to construct individual parts of the model. This idea is illustrated in Figure 3 that shows a product model that consists of three parts where each part is iteratively constructed with another process model. The vertical boxes represent snapshots of the entire model at particular points in time t_1 and t_2 where one can see that the parts of the model are in different process stages.

Combining these ideas in a joint model results in the Iterative View Engineering model shown in Figure 4. 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 model and the implementation are empty. In a first cycle of iterative enhancement (①), the designer specifies the first version of the model 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 model are implemented and if an error occurs during the construction phase, the model has to be refined (③) until it can be implemented. Next, the initial implementation is tested. During the test phase, it often turns out that the model 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 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 (⑥) in order to comply to the modeling language. After this step, the next cycle is executed until the entire system is fully implemented.

The generic process model presented in this section can be used to construct the view-oriented product model that was discussed earlier. However, the focus of the Iterative view engineering model is still a single project and thus we need an additional institutional framework the supports learning and reuse over project boundaries.

The *Experience Factory* [Basili *et al.*, 1994] is such a conceptual framework that supports systematic learning within an organization. The basic idea underlying the Experience Factory is closely related to case-based reasoning and operates on similarities of software development processes.

The Experience Factory method differentiates between two organizational substructures: the *Project Organization* is responsible for the project work on the basis of software development process according to the Quality Improvement Paradigm (QIP) [Basili

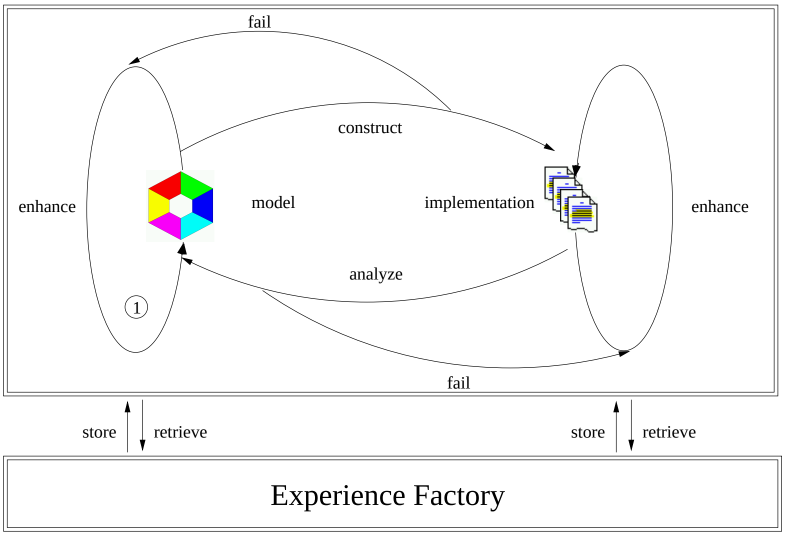

Figure 5: The MASSIVE Development Model

et al., 1994] and on the basis of the process models presented in the previous sections. The *Experience Factory*, on the other hand is completely independent of the project organization and is responsible for analyzing and synthesizing all kinds of experiences and to provide the organization-wide repository of experiences.

The major advantage of the Experience Factory is that it provides a single interface to the knowledge that exists within an organization and a central authority that collects and updates this knowledge. Furthermore, an Experience Factory can be introduced in parallel to the ongoing business of an organization and does not require complicated synchronization processes nor does it disturb the normal operational business. Finally, the idea of an Experience Factory is highly scalable. It can be used by an individual programmer who will then play several roles in order to store his or her personal experiences for later reuse, or it can be introduced into a large company as strategic department that increases the resource efficiency of the company by avoiding double work.

The institutional framework of the Experience factory completes the building blocks for the MASSIVE method according to the requirements given in the introduction. In the next section, we will demonstrate how the pieces are assembled in order to yield a coherent development method for multiagent systems.

4 The MASSIVE method

The core of the MASSIVE method is of course the view system that was outlined in Section 2. The view system is the conceptual basis for a wide range of product models that are developed and refined throughout all software projects that are carried out according to the suggested process model. The main idea of the process model is an iterative view engineering approach that is itself based on iterative enhancement and round-trip engineering. However, as we have said before, this macro process contains itself several small micro processes that are used for individual activities within the design process for a particular view. These micro models as well as the overall process model are in no way fixed for the entire lifetime of the project model, they are also subject to changes and refinements during the course of time. In order to preserve these adaptations and to make them accessible to others, the process model and the product model are both embedded into a larger organizational structure called the Experience Factory. The Experience Factory provides the formal framework for a permanent learning process that takes place over project boundaries and that eventually models the multiagent experience of organization in terms of specific product and process models for various domains.

Figure 5 shows the resulting structure and shows how the product model is integrated into the process model that is itself embedded into the framework of the Experience Factory.

5 Case Studies

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 [Lind *et al.*, 1999] 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 **P**ersonal **T**rip **A**ssitant (PTA) for individual travelers according to the FIPA [FIPA, 1997] standards. One of the services required in the PTA domain is intermodal route planning for which we have implemented a multiagent solution.

- The TEAMWORK LIBRARY [Lind, 1996] 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 these domains.

6 Summary and Future Work

We have presented a pragmatic software engineering development model that is specialized for the development of multiagent systems. The model features a view-oriented product model, an iterative process model and an institutional framework that supports organizational learning and reuse over project boundaries. The model presented in this paper has evolved during several projects and it should be adapted by potential users to fit their individual needs and experiences.

One of the most interesting topics in the design research community in the last years is multiagent technology to support the design process [Lander, 1997] and in our future work, we will thus try to use the MASSIVE development method to built a multiagent system that supports the method itself. The resulting system will be comprised of several repository agents that are responsible for managing the resources in the Experience base and a number of wizards that assist the user in the micro processes for each view and that are responsible for maintaining the consistency of the entire model.

Acknowledgments

I would like to thank my colleagues at the MAS group of the DFKI, my project partners and the people at the Intelligent Business Systems Research Group of BT for their feedback on various aspects of MASSIVE. Furthermore, I am also thankful to the anonymous reviewers for their comments.

References

[Balzert, 1998] Helmut Balzert. *Lehrbuch der Software-Technik*, volume II. Spekrum Akademischer Verlag, 1998.

[Basili and Turner, 1975] V. R. Basili and A. J. Turner. Iterative Enhacement: A Practical Technique for Software Development. In *Proc. of the 1st National Conference on SE*, 1975.

[Basili *et al.*, 1994] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Experience Factory. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1. John Wiley & Sons, 1994.

[Booch, 1996] Grady Booch. *Object Solutions: Managing the Object-Oriented Project*. Addison-Wesley, 1996.

[Brazier *et al.*, 1997] F.M.T. Brazier, B. Dunin-Keplicz, N. Jennings, and J. Treur. Desire: Modelling multi-agent systems in a compositional formal framework. *Int. Journal of Cooperative Information Systems*, 6, 1997.

[Brooks, 1986] Frederick P. Brooks, Jr. No silver bullet. In H.-J. Kugler, editor, *Proceedings of the IFIP Tenth World Computing Conference*, pages 1069–76, Elsevier Science, 1986.

[Burmeister, 1996] Birgit Burmeister. Models and methodology for agent-oriented analysis and design. In *KI'96 Workshop*, number D-96-06 in DFKI Documents. DFKI, 1996.

[Collins and Ndumu, 1998] J. Collins and D. Ndumu. The ZEUS Role Modelling Guide. Technical report, BT, Adastral Park, Martlesham Heath, 1998.

[EURESCOM, 1999] EURESCOM. MESSAGE: Methodology for Engineering Systems of Software AGEnts. Technical report, EURESCOM, 1999.

[FIPA, 1997] FIPA. Fipa '97 specification parts 1–7, version 1.0, 1997. The Foundation for Intelligent Physical Agents.

[Iglesias *et al.*, 1997] Carlos Iglesias, Mercedes Garrijo, José Gonzalez, and Juan R. Velasco. Analysis and design of multiagent systems using MAS-CommonKADS. In *ATAL-97*, 1997.

[Iglesias *et al.*, 1998] Carlos Iglesias, Mercedes Garrijo, and José Gonzalez. A Survey of Agent-Oriented Methodologies. In *ATAL-98*, 1998.

[Jennings *et al.*, 1998] N. R. Jennings, K. P. Sycara, and M. Wooldridge. A roadmap of agent research and development. *JAAMAS*, 1(1), 1998.

[Lander, 1997] Susan E. Lander. Issues in Multiagent Design Systems. *IEEE Expert*, April 1997.

[Lind *et al.*, 1999] Jürgen Lind, Klaus Fischer, Jörg Böcker, and Bernd Zirkler. Transportation Scheduling and Simulation in a Railroad Scenario: A Multi-Agent Approach. In *PAAM99*, London, 1999.

[Lind, 1996] Jürgen Lind. TWLib – A Generic Library for TEAMWORK Applications. Master's thesis, University of Kaiserslautern, 1996.

[Noda, 1995] I. Noda. Soccer Server: A Simulator of Robocup. In *Proc. of AI symposium 1995*. Japanese Society for Artificial Intelligence, 1995.

[Parnas and Clements, 1986] D. Parnas and P. Clements. A Rational Design Process: How and Why to Fake It. *IEEE Transactions on Software Engineering*, SE-12(2), 1986.

[Royce, 1970] Winston W. Royce. Managing the development of large software systems: Concepts and techniques. In *WESCON Technical Papers, v. 14*, Los Angeles, 1970. WESCON.

[Wooldridge *et al.*, 1999] M. Wooldridge, N. R. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. In *Agents '99*, 1999.