Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research
Report
TM-99-03

A Process Model for the Design of Multi-Agent Systems

Jürgen Lind

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

April 1999

Deutsches Forschungszentrum für Künstliche Intelligenz

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210
E-Mail: info@dfki.uni-kl.de

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341
E-Mail: info@dfki.de

WWW: http://www.dfki.de

Deutsches Forschungszentrum für Künstliche Intelligenz
DFKI GmbH
German Research Center for Artificial Intelligence

Founded in 1988, DFKI today is one of the largest nonprofit contract research institutes in the field of innovative software technology based on Artificial Intelligence (AI) methods. DFKI is focusing on the complete cycle of innovation — from world-class basic research and technology development through leading-edge demonstrators and prototypes to product functions and commercialization.

Based in Kaiserslautern and Saarbrücken, the German Research Center for Artificial Intelligence ranks among the important “Centers of Excellence” worldwide.

An important element of DFKI's mission is to move innovations as quickly as possible from the lab into the marketplace. Only by maintaining research projects at the forefront of science can DFKI have the strength to meet its technology transfer goals.

DFKI has about 115 full-time employees, including 95 research scientists with advanced degrees. There are also around 120 part-time research assistants.

Revenues for DFKI were about 24 million DM in 1997, half from government contract work and half from commercial clients. The annual increase in contracts from commercial clients was greater than 37% during the last three years.

At DFKI, all work is organized in the form of clearly focused research or development projects with planned deliverables, various milestones, and a duration from several months up to three years.

DFKI benefits from interaction with the faculty of the Universities of Saarbrücken and Kaiserslautern and in turn provides opportunities for research and Ph.D. thesis supervision to students from these universities, which have an outstanding reputation in Computer Science.

The key directors of DFKI are Prof. Wolfgang Wahlster (CEO) and Dr. Walter Olthoff (CFO).

DFKI's six research departments are directed by internationally recognized research scientists:

- ❏ Information Management and Document Analysis (Director: Prof. A. Dengel)
- ❏ Intelligent Visualization and Simulation Systems (Director: Prof. H. Hagen)
- ❏ Deduction and Multiagent Systems (Director: Prof. J. Siekmann)
- ❏ Programming Systems (Director: Prof. G. Smolka)
- ❏ Language Technology (Director: Prof. H. Uszkoreit)
- ❏ Intelligent User Interfaces (Director: Prof. W. Wahlster)

In this series, DFKI publishes research reports, technical memos, documents (eg. workshop proceedings), and final project reports. The aim is to make new results, ideas, and software available as quickly as possible.

Prof. Wolfgang Wahlster
Director

A Process Model for the Design of Multi-Agent Systems

Jürgen Lind

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

DFKI-TM-99-03

This work has been supported by a grant from The Federal Ministry of Education, Science, Research, and Technology (FKZ ITW-).

ISSN 0946-008X

A Process Model for the Design of Multi-Agent Systems

Jürgen Lind

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

April 9, 1999

Abstract

In this paper, we propose a pragmatic process model for the development of multi-agent system based on the combination of standard software engineering techniques with a special focus on multi-agent systems. The resulting process model is the attempt to make our experience in the design of multi-agent systems available to other system designers.

The approach presented in this paper has evolved over several years and it has been successfully applied and refined in different types of multi-agent systems. A short case study of our latest project is included in the paper.

1 Introduction

It is a widely agreed fact in the Software Engineering community that any software life cycle or process model must be tailored towards the characteristic needs of the application domain of the target system [3] [4]. There is no "silver bullet" [7], ie. none of the numerous existing techniques can be used for all types of problems. Therefore, we resisted the temptation to announce yet another "general purpose" method, but instead we suggest a design methodology specialized for multi-agent systems.

Several characterizations for multi-agent systems (MAS) have been proposed eg. [18], [15], [14], [40], summarizing to the following: multi-agent systems are systems with a variable number of interacting, autonomous entities that communicate with each other using flexible, complex protocols. The agents within a multi-agent system usually have complex individual components and run concurrently in a distributed environment. Multi-agent systems are usually continuous systems (as opposed to functional systems according to [38]) with a loose coupling between the individual components of the system.

Not every multi-agent system has all features mentioned above, but we can use this characterization to derive a number of requirements that are specific for this kind of systems. Some of the requirements are direct results of the features mentioned while others represent non-functional aspects of the target systems.

The concurrent, distributed approach to problem solving requires a sophisticated system design that first of all guarantees that the resulting system is dead-lock free and

terminating (wrt. to a given task, ie. this is no contradiction to the concept of a continuous system!). Furthermore, we would also like to have a deterministic system that allows the reproduction of system runs. Though this requirement cannot always be be realized (eg. in the case of randomized algorithms) it is nonetheless an important factor to increase the users trust into the system. Closely connected to this requirement is the need to produce traceable systems [28]: the designer (and later the user) must be enabled to follow the systems activities in order to decide whether the system shows a reasonable behaviour or not. This is also of vital interest for the development phase of the system with respect to debugging. For monolithical systems, debugging features are supported by the programming language or a development environment; in the case of multi-agent systems, however, only little or no support is given by existing languages and environments. Thus, either standard agent development frameworks must be used or it is the task of the system designer to develop and integrate the facilities to support tracing and debugging. Finally, the target system must be designed to scale [22]. Designing a system that works well for 10 agents is often a straightforward task; however, scaling the system up to, say, 5000 agents is often a completely different matter and may require a re-design of major parts if the operational size of the system was neglected during the design and test phase.
In this paper, we propose a pragmatic process model for the development of multi-agent systems. We have combined several standard SE techniques such as round-trip engineering or multi-view modelling and focused them on the special context of multi-agent systems. The resulting process model is not the attempt to invent the "silver bullet" but the attempt to make our experience in the design of multi-agent systems available to other system designers. We strongly recommend to use standard software components or design patterns whenever they are available because the re-invention of the wheel usually leads to sub-optimal solutions! The approach presented in this paper has evolved over several years and it has been successfully applied and refined in different types of multi-agent systems:

- The multi-agent solution for the Train Coupling- and Sharing (TCS) approach [36], [25], [26] is a system for scheduling and cost optimization of a large number of railroad transportation tasks using novel railroad technologies.
- In the MoTiV/PTA [6], [33] project, a multi-agent system for distributed intermodal route planning was developed.
- The Teamwork Library [24], [10], [11] is a framework for distributed search that was originally developed for equational theorem proving but it was also used in other application fields [20], [23].

This paper is organized as follows: firstly, we define our view on the software development process and on the products that are generated during that process. We then introduce the basic elements of our process model before we present the model itself. A short case study of a project that was executed using the process model and some of the lessons learned during the project are presented next; a conclusion summarizes the paper.

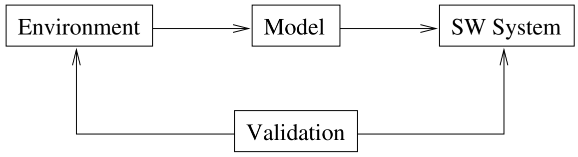

Figure 1: SW Development Process

2 Models and Systems: What and How?

Before we can present our design approach, it is necessary to clarify our view on the software design process and the product(s) involved in this process. The ideal software development process is shown in figure 1:

1. The designer constructs an abstract model of the aspects of the real world that should be modelled by the SW System.

2. The system is implemented according to the model.

3. The operational system is evaluated according to the real world.

The result of this development process is a software system that consists of a *model* and the *code* that implements the model on a computer platform. The model represents the designers view on the system and its environment as well as the designers intention of what the system is supposed to to.
The model and the code are linked together in a way that the features of the model are mapped to features of the code. This mapping is by no means a bijection: as shown in Figure 2, a single feature of the model can be linked to several features of the code and vice versa. Furthermore, the links shown in Figure 2 are to be interpreted as "rubber strings" in the sense that pulling on a link has effects on all (directly or indirectly) connected model or code features — the effects decrease with increasing distance or increasing number of intermediate nodes.
In the sequential development process shown in Figure 1, the designer will in a first step construct the model and implement the corresponding code in a second step. Unfortunately, this ideal process usually cannot be established eg. because the model is incomplete or inconsistent, or the mapping of the model to a given computer platform is faulty or impossible with a given hard- or software.
Therefore, we suggest an *incremental* system development approach: the designer constructs a minimal model of the system which is instantly implemented. Then the model and the code are consistently updated using one of the two possible operations: the **enhance** operation adds new features to the model or the code an specifies their respective links while the **adjust** operation "pulls" on some of the existing links (eg. by adjusting the code according to features of the underlying operating system) and then updating the linked features (eg. the parts of the model affected by a specific requirement).

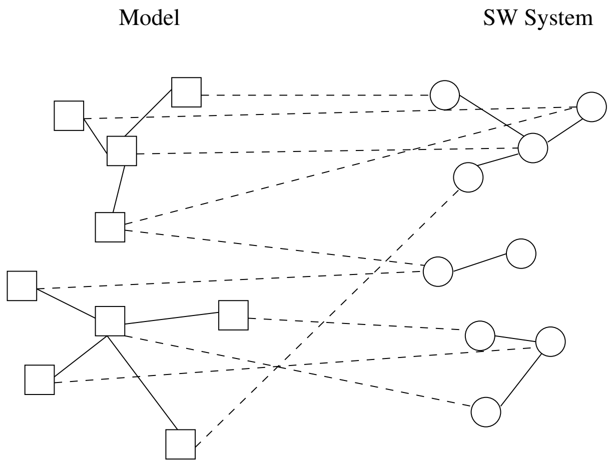

Figure 2: Connections between Model and SW System

An example for this iterative enhancement process is shown in Figure 3: in a first step, the initial model is enhanced by a GUI component and a Input/Output component which are also added to the initial code fragment. In the following steps, these components would be extended until the final system is constructed.

3 Process Model

In this section, we present the building blocks and a semi-algorithmic process model for the idea sketched in the previous section.

3.1 Views

The fundamental abstractions used in our process model are so-called *views*. Views represent orthogonal aspects of the target system and thus allow for a rather natural decomposition of the design task.
In our model, we differentiate between three views: The *architectural (or static) view* is concerned with the entities within the system and their static interconnections, the *interaction (or dynamic) view* handles more task oriented questions but still on a rather abstract level and the *functional view*, finally, focuses on the local decision and planning algorithms of the agents.
Although these views are quite independent of each other – though not completely independent, eg. the features of the functional view (eg. what the agents are supposed to do) surely depend on the architectural question what objects will be the agents –, they can be ordered according to their application dependency as shown in Figure 4: a lot of the questions handled in the architectural and interaction view are rather application independent, this allows to build abstract multi-agent systems that share the same architecture and interaction scheme but that can be instantiated for different problem

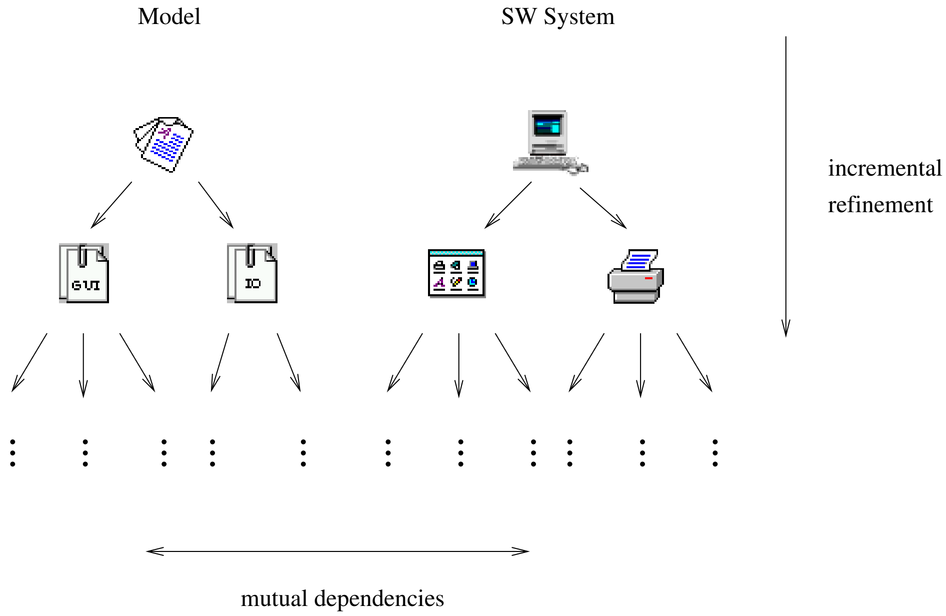

Figure 3: Example for Model Refinement

areas by adjusting the functional view. For example, the basic architecture and interaction scheme that was used in the TELETRUCK system [8] was also successfully applied for the T^{CS}_{MAS} system [26].
In the next section, we describe how the different views can be developed and how they are integrated in order to build an operational system.

3.2 Process Model

A process model in software engineering sense is a formalization of the software design and implementation activities and of the products that are connected with these activities [19]. Sequential models such as the Waterfall [31], [19] or the V-Model [2] are organized in several steps where each step has a number of actions together with associated pre- and postconditions. The activities in each step be can executed when the preconditions are met and they are terminated when the postconditions become true. Associated with each activity are one or more products. Typical products that occur within the waterfall model are the system design document, the component design documents or the code.
The model proposed in this paper has a slightly different view on the products and activities; in our model, only a single product (consisting of the model and the code) exists. This single product is refined and extended during the software development process until is has reached an acceptable state. The various activities of the software development process are thus executed several times; models of this type are usually called iterative enhancement [5] or round-trip engineering models [2].
In our model, we have three major activities that implement the model enhancement

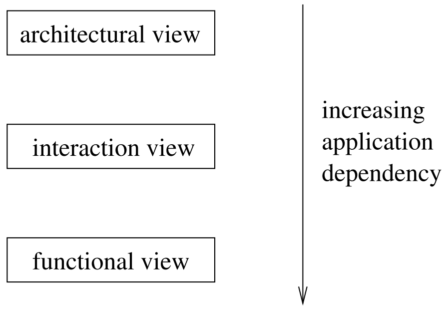

Figure 4: View Hierarchie

or model adjustment discussed earlier. These activities are *integrate*, *validate* and *feedback*: the different views presented in the previous section are assembled in so-called integration steps that join together fragments of the model or the code in order to generate larger units. These integration steps are then followed by validation steps that determines whether the integration process led to satisfactory results or not. In the latter case, a feedback loop is activated that distributes the results of the validation process (and thus the reasons for failures of this process) via the *feedback bus* to the model specifications. These specifications must are changed accordingly and new integration and validation attempt is taken. This process iterates until the validation succeeds.
The complete model shown in Figure 5 is executed as follows:

1. A feasibility analysis [40] is executed in order to decide if the problem domain in question is suitable for a multi-agent solution. Then a requirements specification of the functional aspects of the target system is generated. This document is later used for the validation steps.

2. During the initial design phase, the initial versions of then architectural, interaction and functional view are developed. These components are the basis for all further activities.

3. (a) i. The first integration phase leads to the first version of the *abstract* multi-agent system. This fragment focuses on architectural and interaction aspects of the target system and neglects functional aspects as far as possible.
      ii. The abstract system is validated against architectural and interaction view.
   (b) In parallel, the components developed out of the functional specification can be validated.

4. The next integration step assembles the abstract system and the components and leads to the so-called *operational* multi-agent system.

5. The operational system is validated against system requirements document. If the state is accepted, this step results in the *final* multi-agent system, otherwise the feedback loop is activated.

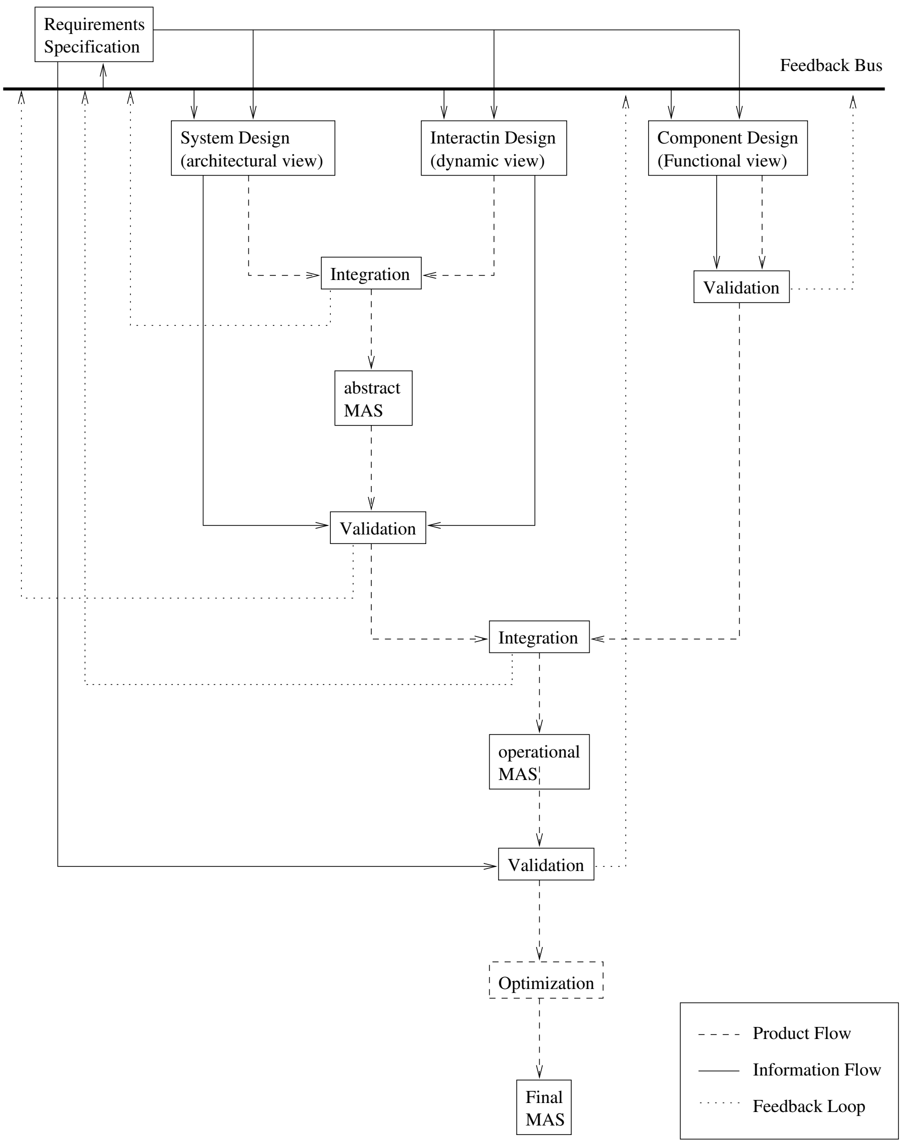

Figure 5: Process Model

6. In an optional step, the operational system can be optimized without(!) changing the model. This step is reserved for non-functional optimization only!

3.3 Design Guidelines

In the following paragraphs, we provide some guidelines about which aspects of the target system should be handled in which of the views discussed earlier. Note that these guidelines represent our personal experiences in the design and implementation of multi-agent applications — each application has unique characteristics that may require an adaption or an extension of the process model. The aspects discussed below are by no means exhaustive; they are just excerpts from our experience base and we recommend that each organization develops their own local guidelines that match their individual needs [9].
The **architectural view** is concerned with the fundamental structure of the application, typical issues to be handled in this view include:

- the *agent architecture* [39] wrt. functional aspects of the application. For applications with fuzzy task specifications, for example, a BDI agent architecture [30], [13] might be more suitable whereas for applications with rather strict task specifications simpler architectures may suffice.

- the *society model* of a multi-agent application. An agent society can either be *homogenous*, ie. it is comprised of agents with the same architecture, or it can be *heterogenous*. In the latter case, the data exchange between agents with different architectures is usually the most difficult part and it has large impact on the entire application — for example, in this case it is usually not possible to use native data formats. Another aspect of the society model is the structure of the agent society. We can have a flat structure where all agents interact on the same level or we can have a hierarchical structure where some agents play a more important role than others.

- the *programming model* to be used in the implementation. Programming models can be classified according to their degree of concurrency: a *sequential* programming model admits for no concurrency at all; functions are called strictly one after the other. This kind of programming model does not require sophisticated synchronization mechanisms and it is sufficient in the case where the multi-agent paradigm is solely used for design purposes, ie. to structure the target system. A *pseudo-parallel* model, on the other hand, typically uses light weight processes (threads) within a single operating system process to achieve concurrent execution of program fragments. This kind of programming model is well suited for multi-agent applications because it allows for a reasonable degree of parallelism while still being easy to handle wrt. debugging and traceability (if the tools nedded for these task are available — we will return to this point in the case study in Section 4). A fully *distributed* programming model, finally, distributes the computation space over several computers connected via a network. This model is quite hard to handle because it often requires complex synchronization and fault tolerance mechanisms. However, for real world applications, this might be the only possible choice.

- the *communication model* that handles all aspects in conjuction with agent interaction. The *agent identification scheme*, for example, determines how the agents find each other within the agent society; possible choices are a fully distributed scheme where each agent knows any other agent or a centralized scheme that uses an *agent directory service* to provide the necessary information to any client. Another important aspect is the message model to be used within the system. *Synchronous* message passing requires a more restricted design of agent interaction but it is usually easier to use than *asynchronous* message passing. However, the asynchronous message exchange usually allows for a more effective use of concurrency.

The **interaction view** handles all matters in conjuction with agent interaction, eg.

- the *negotiation protocols*. For example, we can differentiate between *competitive* or *market based* negotiation protocols where each agent tries to optimize its local performance, and *decompositional* protocols that assume that the interacting agents are willing to cooperate to achieve the system goal. Thus, these protocols only seek to find the best decomposition of the original system task.
- *Communication bottleneck analysis*. In multi-agent systems, communication plays the major role and therefore, traffic analysis is one of the most important questions wrt. to system performance and scalability. A thorough analysis of which agents communicate using which messages can often help to eliminate traffic overheads and thus reduce the communication ratio of the system in order to increase the average system performance.

The **functional view**, finally, models

- The *local (autonomous) decisions* of the agents. The decisions that do not consider the other agents within the agent society are local decisions — a prominent example for this kind of functions are the plan execution facilities of an agent.
- The *decisions during interaction processes*. The decisions taken by an agent when it is involved in a multi-agent negotiation protocol are the interaction decisons. An example for this kind of functions are cost functions that are used by the agents to evaluate the utility of particular (joint) actions.

In this section, we have presented a process model that features a multi-view approach with multi-stage feedback. The model has explicit feedback loops that direct the information gained in the validation steps to all relevant components and it supports incremental system development using an iterative refinement loop. In the next section, we present a brief overview over a software project that was executed using this process model.

4 Case Study: $\mathrm{T}^{\mathrm{CS}}_{\mathrm{MAS}}$

4.1 Project Overview

The goal of the $\mathrm{T}^{\mathrm{CS}}_{\mathrm{MAS}}$ project [25], [26] is to evaluate an alternative approach [21], [12], [36] to classical freight transport process that uses small railroad *transportation mod-*

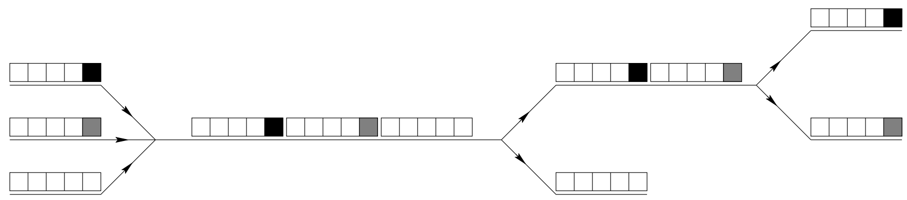

Figure 6: Train Coupling and Sharing

ules [37] instead of conventional trains. Whereas a normal train is made up of one locomotive and several freight waggons, a transportation module consists of two power units on either side of the module and three permanently coupled intermediate vehicles with a fixed number of loading spaces. Thus, a transportation module is a single unit of limited size. When a company wants to deliver some freight to a customer, it orders a transportation module at a local freight center and loads its goods on this module. The module itself is then responsible to find its way through the railroad network. The problem is now, that a location route in a railroad network cannot be used by two **independent** modules at the same time. Either a route is blocked by a single module or two (or more) modules share a route by hooking together at the beginning of a location route and splitting up afterwards. This process is shown in Figure 6; modules that share location routes are referred to as *unions*. In order to use the underlying railroad infrastructure most efficiently, the railroad modules should share as many location routes as possible while taking care of their local constraints.
A schedule for the modules is generated in the following way: an initial solution is obtained by running the *contract-net* [34] protocol whenever a new module enters the system. New modules are incrementally integrated in the existing schedule which guarantees, that always a solution for the problem (as far as it is known to the system) exists. However, this solution may be – and usually is – not optimal. In order to improve the quality of the existing solution, the *simulated trading* [1] protocol is run on the set of unions currently known to the system.
The multi-agent system to solve this optmization task is fully implemented in Oz [35], a high-level programming language that combines constraint inference with concurrency. Oz is a dynamically typed language and supports object-oriented system development, finite domain and feature constraints. The implementation of the $\mathrm{T}^{\mathrm{CS}}_{\mathrm{MAS}}$ system consists of approximately 74 classes with 18000 lines of code. The decomposition of the system according to our view scheme is the following:

Architectural view In the $\mathrm{T}^{\mathrm{CS}}_{\mathrm{MAS}}$ system, we use the INTERRAP agent architecture [27] as architecture for the agents that represent the transportation modules in the scenario. The original model is extended to feature a holonic approach [16] that makes it easier to model the union formation process during the problem solving. Furthermore, we have a homogenous, flat agent society, ie. all agents have the same architecture and each agent can interact with any other agent. The programming model is a pseudo-parallel model that is supported by the implementation language, agent identification is achieved by using a central agent directory service and the message model is synchronous.

Interaction view We use two cooperative, market-based negotiation protocols — the *contract net* protocol to generate an initial solution whenever a new task is announced to the system and the *simulated trading* protocol that is used to optimize the current solution. The optimization process is not executed in every round, instead it is only activated every k rounds or on an explicit user request.

Functional view The functional aspects include the local planning of optimal routes for the individual modules and the planning of optimal locations for join operations with other modules. The system features a dynamic plan execution monitoring component that is also modelled in this view.

Due to the limited space we can only give a short summary of the T^{CS}_{MAS} system, the interested reader should refer to [25] and [26] for details. In the next section, we will discuss some of the experiences that we made during the course of the project.

4.2 Lessons learned

Every software project adds some new knowledge to the experience database of the designer and to the organization knowledge. The vast majority of these new experiences are only small pieces that "tune" the process model such that it matches the new, special requirements of the project. Sometimes, however, very important gains in experience occur — "experience milestones" might be an adequate term for this kind of experiences. In this paper, briefly discuss two experience milestones that we discovered within the T^{CS}_{MAS} project.

The first lesson we learned is concerned with monitoring and debugging a multi-agent system with a pseudo-parallel computation model. Earlier projects (the Teamwork Library and the MoTiV/PTA project) featured distributed models that made use of real parallelism; the T^{CS}_{MAS} project on the other hand uses Oz threads to implement the agents within a single user process. Although the Oz programming system has a debugger, it is hard to follow the activities within a multi-agent system because the debugger does not respect the "bounds" of the agents, ie. it does not support navigation within the code being executed wrt. the question which agent is being inspected at the moment etc. Thus, additional monitoring and debugging features are necessary and must be provided by the system engineer. In the T^{CS}_{MAS} project, we developed several tools that enabled us to inspect the messages being sent by the agents, their respective plans or ongoing computation activities of the agents. The relevant point now is not, that these tools had to be designed and implemented. The important point is, that none of these activities was foreseen and thus included in the original project schedule! We simply did not see the necessity when the project plan was set up and so the plan had to be modified in order to include the additional work packages that resulted from the need for monitoring and debugging support. Thus, whenever possible, standard agent programming frameworks such as presented in [32] or [29] should be used in order to relief the system designer from this task. If the use of agent libraries or agent programming frameworks is not possible, ie. because of a special programming language to be used in the project, the effort to set up the necessary tools must be carefully evaluated and included in the project plan.

The second important aspect during the project was the vulnerability of a multi-agent systems wrt. to scalability. Much has been written about this subject [40], [22], [17], but in a real world project the difficulty remains to adapt the academic results to the possibilities and limitations of the system under consideration. It is usually not possible to build the application in a way that contemplates with the prerequisites of a research model and then to apply the derived concepts. Instead, the designer must find a pragmatic way to preserve the system structure and to make use of the research results in order to get the system running for larger problems. In the $\mathrm{T}^{\mathrm{CS}}_{\mathrm{MAS}}$ project for example, the operational validation of the system was made with only small or medium size problem descriptions. This was partly due to the fact that real problem descriptions were not available at project start and thus the designers had to use their imagination to generate test cases — as it turned out, these test cases were by far smaller than the problem sizes to expect and consequently, the system did not scale. To overcome this difficulty, we had to spent additional effort to reduce the agent interaction in a way that the resulting system still yields good solutions but in a reasonable time. This goal was eventually achieved by the use of clustering techniques. The problem demonstrates, however, how important it is for the designer to keep the operational size of target system in mind when the test cases are specified.

5 Conclusion

We have presented a pragmatic software engineering process model that is specialized for the development of multi-agent systems. The process model is an iterative model that features a multi-view approach towards the system design. 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. A brief case study has demonstrated how the method was used in an actual project.

References

[1] BACHEM, A., HOCHSTÄTTLER, W., AND MALICH, M. Simulated trading: A new approach for solving vehicle routing problems. Tech. Rep. 92.125, Mathematisches Institut der Universität zu Köln, Dezember 1992.

[2] BALZERT, H. *Lehrbuch der Software-Technik*, vol. II. Spekrum Akademischer Verlag, 1998.

[3] BASILI, V. R. The Experience Factory: packaging software experience. In *Proceedings of the 14th International Conference on Software Engineering* (1989), NASA Goddard Space Flight Center.

[4] BASILI, V. R., CALDIERA, G., AND ROMBACH, H. D. Experience Factory. In *Encyclopedia of Software Engineering*, J. J. Marciniak, Ed., vol. 1. John Wiley & Sons, 1994, pp. 469–476.

[5] BASILI, V. R., AND TURNER, A. J. Iterative enhacement: A practical technique for software development. In *Proceedings of the First National Conference on Software Engineering* (Sept. 1975), IEEE Computer Society Press, pp. 56–62.

[6] BAYRISCHE LANDESREGIERUNG. Bayerninfo, 1996. http://www.bayerninfo.de.

[7] BROOKS, JR., F. P. No silver bullet. In *Proceedings of the IFIP Tenth World Computing Conference* (Elsevier Science, 1986), H.-J. Kugler, Ed., pp. 1069–76.

[8] BÜRCKERT, H.-J., FISCHER, K., AND VIERKE, G. Transportation scheduling with Holonic MAS, the TeleTruck approach. In *Proc. PAAM98* (1998).

[9] CANTONE, G. Advanced software factory: Models and experiences for the improvement. In *Proceeedings of the CQS International Conference on Quality Software* (Milan, Italy, October 1991).

[10] DENZINGER, J. The teamwork approach to distributed search. Tech. rep., Universität Kaiserslautern, 1994.

[11] DENZINGER, J., AND LIND, J. Twlib - a library for distributed search applications. In *Proceddings of the ICS96-AI* (Kaohsiung, 1996), pp. 101–108.

[12] FABEL, P. Increasing the flexibility of freight traffic - using modular train units as an example. In *Proceedings of the World Congress of Railway Research (WCRR)* (Colorado Springs; USA, 1996).

[13] FISCHER, K., MÜLLER, J. P., AND PISCHEL, M. A pragmatic BDI architecture. In *Intelligent Agents — Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95)* (1996), M. Wooldridge, J. P. Müller, and M. Tambe, Eds., Lecture Notes in Artificial Intelligence, Springer-Verlag.

[14] FRANKLIN, S., AND GRAESSER, A. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages* (1997).

[15] FULBRIGHT, R. D., AND STEPHENS, L. M. Classification of multiagent systems. Tech. Rep. ECE-LMS-94-06, University of South Carolina, Columbia, SC 29208, June 1994.

[16] GERBER, C., SIEKMANN, J., AND VIERKE, G. Holonic multi-agent systems. Tech. Rep. TR-99-01, DFKI, 1999.

[17] GERBER, C., STEINER, D., AND BAUER, B. Resource adaptation for a scalable agent society in the mecca domain. In *Intelligent Software Agents for Communication Networks*. Springer, 199.

[18] GOODWIN, R. Formalizing properties of agents. Tech. Rep. CMU–CS–93–159, School of Computer Science, Carnegie-Mellon Universit, Pittsburgh, PA, May 1993.

[19] JALOTE, P. *An Integrated Approach to Software Engineering*, 2nd ed. Spinger, 1997.

[20] KÖGL, C. Verteilte Berechnung von Gröbnerbasen unter Verwendung des Teamwork-Paradigmas. Master's thesis, Universität Kaiserslautern, 1995.

[21] KRACKE, R., SIEGMANN, J., VOGES, W., BOECKER, J., AND ZIRKLER, B. Systemgestaltung des Schienengüterverkehrs unter Einsatz der Strategie des Train-Coupling and -Sharing. Tech. rep., Universität Hannover, 1995. Studie im Auftrag der DB AG.

[22] LEE, L., NWANA, H., NDUMU, D., AND DE WILDE, P. The stability, scalability and performance of multi-agent systems. *BT Technology Journal 16*, 3 (1998).

[23] LEOPOLD, T. Verteilte Lösung des Travelling-Salesman-Problems durch TEAMWORK. Master's thesis, Universität Kaiserslautern, 1995.

[24] LIND, J. TWLib – a generic library for TEAMWORK applications. Master's thesis, Universität Kaiserslautern, 1996.

[25] LIND, J., AND BÖCKER, J. Optimising the Train Coupling and -Sharing system with a multi-agent approach. In *Proceedings of the 11th Mini-EURO Conference on AI in Transportation Systems and Science* (Helsinki, 1999). To appear.

[26] LIND, J., FISCHER, K., BÖCKER, J., AND ZIRKLER, B. Transportation scheduling and simulation in a railroad scenario: A multi-agent approach. In *Proceedings of the Fourth International Conference on the Practical Application of Intelligent Agents and Multi-Agents* (London, 1999). To appear.

[27] MÜLLER, J. P. *The Design of Intelligent Agents: A Layered Approach*, vol. 1177 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag, Dec 1996.

[28] NDUMU, D. T., NWANA, H. S., LEE, L. C., AND COLLINS, J. C. Visualising and debugging distributed multi-agent systems. In *Proceedings of the 3rd International Conference on Autonomous Agents* (1999). To appear.

[29] NWANA, H. S., NDUMU, D. T., LEE, L. C., AND COLLINS, J. C. ZEUS: A tool-kit for building distributed multi-agent systems. *Applied Artifical Intelligence Journal 13*, 1 (1999), 129–186.

[30] RAO, A. S., AND GEORGEFF, M. BDI Agents: from theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)* (San Francisco, CA, June 1995), pp. 312–319.

[31] ROYCE, W. W. Managing the development of large software systems: Concepts and techniques. In *WESCON Technical Papers, v. 14* (Los Angeles, Aug. 1970), WESCON. Reprinted in Proceedings of the Ninth International Conference on Software Engineering, 1987, pp. 328–338.

[32] SCHILLO, M., LIND, J., FUNK, P., GERBER, C., AND JUNG, C. SIF - the social interaction framework system description and user's guide to a multi-agent system testbed. Tech. Rep. TR-99-02, DFKI GmbH, 1999.

[33] SIEMENS AG. Verfahren und Anordnung zur Ermittlung einer Route von einem Startpunkt zu einem Zielpunkt. Deutsche Patentanmeldung 197 46 417.3, 1997. Donald Steiner, Jürgen Lind, Alastair Burt and Hartmut Dieterich.

[34] SMITH, R. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers* (1980).

[35] SMOLKA, G. The Oz programming model. In *Computer Science Today*, J. van Leeuwen, Ed., Lecture Notes in Computer Science, vol. 1000. Springer-Verlag, Berlin, 1995, pp. 324–343.

[36] VOGES, W., AND MIERAU, U. Train Coupling & -Sharing. In *Proceedings of the World Congress of Railway Research (WCRR)* (Florence; Italy, 1997).

[37] WINDHOFF AG. CargoSprinter, 1996. http://www.windhoff.de.

[38] WOOLDRIDGE, M. Agents and software engineering. *AI*IA Notizie XI(3)* (1998), 31–37.

[39] WOOLDRIDGE, M., AND JENNINGS, N. R. Intelligent agents: Theory and practice. *The Knowledge Engineering Review 10*, 2 (1995), 115–152.

[40] WOOLDRIDGE, M. J., AND JENNINGS, N. R. Pitfalls of agent-oriented development. In *Proceedings 2nd International Conference on Autonomous Agents (Agents-98)* (Minneapolis, 1998), pp. 385–391.

A Process Model for the Design of Multi-Agent Systems

TM-99-03
Research Report

Jürgen Lind

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