Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research Report
RR-92-41

A Multi-Agent Approach towards Group Scheduling

Andreas Lux

August 1992

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
D-6750 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Philips, SEMA Group Systems, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct *systems with technical knowledge and common sense* which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Intelligent Communication Networks
- Intelligent Cooperative Systems.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Prof. Dr. Gerhard Barth
Director

A Multi-Agent Approach towards Group Scheduling

Andreas Lux

DFKI-RR-92-41

Parts of this report have also appeared in:

A. Lux, F. Bomarius, D. Steiner: A Model for Supporting Human Computer Cooperation. in: Proceedings of the AAAI 92 Workshop on Cooperation among Heterogeneous Intelligent Systems, San Jose, CA. July 1992.

This work has been partially supported by the European Community as part of ESPRIT II project 5362 IMAGINE.

A Multi-Agent Approach towards Group Scheduling

Andreas Lux

German Research Center for Artificial Intelligence Inc.
Project KIK-Teamware

P.O. Box 2080, D-6750 Kaiserslautern, West Germany

e-mail: lux@dfki.uni-kl.de

August 1992

Abstract

Thanks to rapid improvements in computer and communication technology the network of national and international business relationships is becoming more and more dense. Intelligent cooperation mechanisms are a necessary prerequisite for efficient cooperation. This report examines an everyday cooperative scenario, scheduling and management of appointments, from the point of intelligent computer support. The example is chosen to clarify our approach towards a formal model to describe cooperative processes. It shows the suitability of the approach to quickly design and implement typical cooperative scenarios. Especially, the integration of different existing calendar tools within the general cooperation model provides a clear advantage over existing approaches.

Contents

Chapter 1

Motivation

Many studies [BR84, KDK85, Eh87a, Eh87b] have been performed how office workers keep or should keep their personal calendars. Although there are many individual differences, the following two common characteristics are evident:

- Most often, a calendar is assigned to one person; it is small and portable and thus easily accessible, even en route.
- For reasons of extra space, some people use a desk calendar as well - despite the maintenance problems involved with the use of two calendars.

Both types of paper calendars are used for a variety of purposes, namely as

- a schedule, i.e a reminder of future events
- a diary for past activities
- a notebook for addresses, phone numbers, birthdays or other repeated events and other important dates

In the last decade, electronic calendars with features such as reminder facilities or automatic alarms have been developed. However, as research in this field has shown [KDK85], these calendars do not offer the power and flexibility of traditional pocket and desk calendars.
The design and implementation of a useful calendar and appointment system, therefore, represents an exciting challenge. Especially the additional capability of automatically scheduling appointments is a clear advantage of electronic calendars over paper calendars. Scheduling group meetings with several participants is a complex and difficult task which requires intelligent cooperation support to become easier and more flexible. Ideally, an intelligent appointment system should behave as a secretary who fixes dates and arranges meetings independently from the superior as far as possible.
Automatic on-line scheduling systems are desirable, because making appointments without computer support requires much cooperation effort between the participants and is often inefficient. . At the moment, several rounds of phone calls, letters or electronic messages are necessary to arrange a meeting between a number of people with busy schedules. So, organisational advantages arise if the amount of cooperation is minimized. With the rapid progress of today's information processing technology, it should be possible to support users in calendar and appointment management.
Additional functionalities can also be easily provided using such technology, e.g. automatic rescheduling of meetings, reservation of meeting rooms or the availability of equipment like overhead projectors.
Such functionalities are useful for the acceptance of the system and support Ehrlich's argumentation [Eh87b], that more individuals will maintain their calendars on-line if the perceived collective benefit is higher. Naturally, the appointment system can only be successful to the extent that users maintain their schedules on-line and up-to-date.
In the rest of this report, we concentrate on the aspects of scheduling appointments and maintaining schedules, although the use of electronic calendars as a diary or a notebook is equally important.

Chapter 2

Existing Studies and Systems

Calendar management and appointment scheduling as research topics have been examined from different perspectives in the last decade; efforts range from studies about the necessary prerequisites and usefulness of such tools up to implementations of prototype systems.
A study by Kincaid et al. [KDK85] has been performed to determine the demands for cooperation support through automatic appointment systems. They interviewed a group of office workers who had access to electronic calendars as integral part of their working environment. They observed that electronic calendars do not yet offer the power and flexibility of traditional paper calendars. Furthermore, they do not contain a facility for automatically scheduling appointments involving multiple participants, although this would be a clear advantage over traditional paper calendars. As a result, they found out that such a component is jugded highly desirable, and they identified functional requirements for it.
A detailed study of the working situation of managers has been done by Beckurts and Reichwald [BR84]. The area of management has a high degree of communicative activities, often having to do with fixed dates and appointment scheduling. Managers agree to an automatic appointment system if they can handle it flexibly with respect to their individual needs; they, however, defend their personal free times.
Studies of Ehrlich [Eh87a, Eh87b] show that electronic calendars primarily fulfill communicative functions for managers or their secretaries. Automatic appointment systems yield organisational advantages, because they minimize negotiation effort and offer additional functionality like reservation of rooms.
Prototypical implementation of appointment systems has been done for different objectives. The spectrum comprises systems where coordination of appointments is an extension of managing dates to systems which concentrate on the communication and cooperation structures of decentralized units.
A typical example of personal related calendar management is the system Alis [Ap86]. There is only very limited support for the coordination of appointments. Only one date can be negotiated at a time. The initiator has to inform himself about the state of the negotiation. He has to evaluate the received answers on his own and has to undertake the appropriate steps; the system's support functionality is restricted to gather and to distribute information and to represent information in the context of appointment scheduling.
The Eden Shared Calendar System [HA85] is implemented on top of the distributed operating system EDEN. The scheduling algorithm is based on the comparison of distributed managed calendars, which are freely accessible. Main emphasis is put on a global consistent user view. Therefore, a group calendar as a special object is introduced to handle the participants' answers and to manage centrally the actual state of the negotiation process.
Other distributed systems which are based on the comparison of calendars are MPCAL and RTCAL [Ci83, SG85, GS87]. They are extensions of the personal calendar system PCAL [Gr84] and were developed at MIT from 1982 through 1985. They provide controlled sharing and delegation of authority for calendar management on the basis of roles. So, different calendar types and different access rights to calendars exist. MPCAL is restricted because it does not undertake coordination activities. It only updates the participants' calendars with respect to their given answers to a meeting proposal. In RTCAL, users share information from their personal calendars in order to schedule a meeting. Participants can speak to each other over a telephone connection and use the computer display as a shared blackboard. Both systems mainly concentrated on data sharing and data consistency aspects in group systems.

In the appointment scheduling domain, the notion of agents was first introduced by [MS88, MS89]. The main purpose of the developed prototype was to examine various methods for distributed problem solving, to experiment with different application independent distributed control algorithms and to answer questions related to decentralization of data. A calendar agent represents a user, manages the user's electronic calendar and takes part in the scheduling process. Dates are fixed by the agents without consultation with their users; the general agreement of all users to all proposed dates is assumed. The scheduling process is supported by user profiles and date profiles. The daily user profile states the general willingness of a user to participate in a meeting for that day; the date profile is computed by exclusion of already fixed dates and additional time attributes like earliest/latest time point, dependency from another date, degree of movability.

In the system TVS [Wo91], appointment scheduling is modelled as a coordinated network of autonomous agents. Coordination is done by a special type of agent, the mediator agent. Appointment scheduling is introduced as a typical scenario to validate a proposed coordination model based on structured conversation [WF86].

VS [BPH+90] is a prototype of a priority-based, graphical system. It was tested in a field study where it showed generally useful; in particular, users found priority-based time slots and access to scheduling decision reasoning advantageous.

Another prototype based on the paradigm of multi-agent systems is MADMAN [EE92]. Each user owns a personal diary agent which she or he can access via a graphical display. In order to schedule group meetings, the corresponding diary agents cooperate in fixing a date that best fits their owners' needs. MADMAN uses concepts proposed in [MS89], especially time profiles for days and activities.

Sen and Durfee [SD92a, SD92b] argue that meeting scheduling is an inherently distributed process because of the natural distribution of calendar data. By viewing negotiation over meetings as a distributed search process they propose a formal model to determine performance and efficiency of different scheduling strategies. Their work is directed toward developing intelligent agents that can negotiate over scheduling options on behalf of their associated humans.

Although our developed prototype was inspired by ideas coming from an application independent approach to modelling human computer cooperative systems [SMH90], there are similarities to ideas and concepts of some of the above mentioned systems.

Like VS, we stress a user friendly interface to set up a meeting and to watch the ongoing cooperation processes. Looking at [MS88], the concept of the calendar agent and our agent model seems comparable at a first glance. However, Mattern et al. concentrated on concepts of distributed programming and used the application domain as a means to show the benefits of their newly developed, event-oriented concurrent language CSSA. We, in contrast, give emphasis to modelling the cooperation process in a multi-agent scenario; we rely on the 'low-level' concepts like parallel programming as already given and concentrate on the 'higher' level of cooperation. With respect to the point of cooperation, our work is mostly related to that of Woitass. However, it is more general, because we do not use a mediator agent to direct cooperation.

Beside the main point of modelling cooperation, we also work out a novel feature, namely the integration of different, already existing calendar tools like EMACS Calendar and Sun's Calentool in the overall appointment system.

Chapter 3

Appointment Management as a Typical Cooperative Scenario

As already pointed out in the first chapter, electronic appointment scheduling can be seen as a cooperative problem between humans and computers as intelligent assistants. It's a real distributed problem, because it is practically impossible to centralize the calendars beyond a certain size of a group and across organisational units. Furthermore, the privacy of the users' personal data and the naturalness of the problem are reasons against a centralized version and promote a distributed approach.
Whereas latest progress in communications and network technology provides the physical basis to develop a real distributed appointment scheduling system, recent research in computer science has come up with models supporting both formal and informal cooperation between geographically distributed entities; different approaches have been taken.
Research in Computer Supported Cooperative Work (CSCW) has led to systems where people can cooperate with each other via computers even when separated by great distances. However, the computer plays only a supporting role in such systems, cf. [GMN+91]; it does not participate actively in the problem solving process.
Distributed Artificial Intelligence (DAI, cf. [BG88]) has enabled computers to cooperate with each other. Drawing on the domain of real-life human cooperation, methods have been developed and formalized which support cooperation among computers; typical examples of such methods are negotiation, contract net and master-slave. DAI uses the term *agent* to denote any participant, human or machine, in a cooperative process.
But there is still a need for a link between these two research directions to support cooperation between humans and machines. The development of systems supporting the cooperation processes between humans and actively participating intelligent computers is the purpose of *Human Computer Cooperative Work* (HCCW, cf. [SMH90]).
Appointment scheduling can be seen as a suitable scenario within the HCCW framework. Before we look at the scenario in more detail, we will first elaborate some requirements for an intelligent appointment scheduling system.

3.1 Requirements for Intelligent Appointment Scheduling

To be well designed and widely usable, an appointment system has to fulfill the following requirements:

- The system should be readily accessible from within the office worker's desktop environment. It should be equipped with features providing sufficient motivation for a user to maintain an up-to-date on-line calendar.
- The user interface should be simple and preferably graphical, similar to a classical paper calendar.
- The process of fixing an appointment should not be too strict but should offer a certain degree of freedom to the participant.

- The system should try to minimize the negotiation effort by taking into account several alternatives and selecting the most appropriate one depending on a given situation.
- The system should provide assisting functionality, i.e.actively take part in the scheduling process; informing participants about rescheduling of an appointment or triggering a new appointment process are examples.
- A comprehensive set of functional capabilities should be provided, for instance the integration of appointment scheduling with the reservation of a meeting room is desirable.

3.2 Involved Agents

As in DAI, we use the general notion of *agent* to denote any type of participant in a meeting. Furthermore, we have adapted the multi-agent framework provided from DAI and developed a generalized agent model for designing and implementing HCCW scenarios like appointment scheduling.
Our agent model distinguishes between the part of an agent which is responsible for the executive tasks (the *body* of the agent) and the part which is responsible for communication and coordination (the *head* of the agent).
The body consists of all skills and functionalities an agent is able to perform on its own, i.e. without any cooperative embedding.
The head is the 'intelligent' part of an agent which can manage the participation of the agent within the cooperation process. With the help of this knowledge the agent can decide whether it can contribute to the solution of the overall problem, which cooperation method is the most appropriate for a given problem to solve, etc. The goal is to be able to add a head to any existing software to create a cooperative agent.[1]
Within the scenario of automatic appointment scheduling different groups of participants can be distinguished:

- the persons who initiate a meeting or ought to attend a meeting
- resources like meeting rooms, pieces of equipment etc.

Humans are more sophisticated than machine agents and therefore have a prominent position. The human participants are linked to each other and to resource agents via a special type of agent, the *user agent*. It is designed according to the basic agent model; however, it provides extra powerful facilities:

- knowledge about the human (preferences, skills, abilities),
- the ability to represent the human when she or he is not present,
- a graphical user interface presenting cooperation processes to the user and functionalities so that the human does not necessarily have to handle cooperation in terms of messages, and
- sophisticated functionalities, from advising the human in selection of cooperation methods to relieving the human from having to deal explicitly with management of cooperation.

The concept of an agent alone is not enough to model a human-machine cooperation as it is necessary for intelligent appointment management. What is missing is a cooperation model by which humans (through their user agents) and machine agents can interact in an effective manner. Such a kind of model is further elaborated in the next section.

[1] Our agent model is described in more detail in [SMH90].

3.3 Cooperation Model

In the following, a conceptual model for supporting integrated human-computer cooperation is shortly described. The model is based on cooperation objects, cooperation primitives and cooperation methods. A *cooperation object* is a unit of work, something an agent or a group of agents has to do. It subsumes concepts like goal, plan, schedule, task assignment, etc. A *cooperation primitive* is a basic unit of communication among agents. Cooperation primitives are messages types drawn from speech-act theory. They convey cooperation objects, thus providing the operational basis for interaction between agents (see (Figure 3.1).

Figure 3.1: Cooperation Primitives

A *cooperation method* provides a common framework for the participation of agents within a cooperation and can be seen as a procedure prescribing how the agents can efficiently conduct a cooperation. Cooperation methods are composed of cooperation primitives and functions for decision making. We have already shown somewhere else [LBS92] how well-known cooperation methods like Contract Net, Negotiation or Master-Slave can be composed by using cooperation primitives. Now, the same will be done for scheduling meetings involving human interaction and automatic assistance by intelligent calendar management.
Tables 3.1, 3.2 and 3.3 sketch the flow of control and the temporal ordering of interactions in fixing an appointment in different ways respectively. Whereas Table 3.1 represents a very optimistic strategy, the other two high-level protocols correspond to more realistic procedures. The procedures are described in more detail in the next section. The different high-level protocols can be regarded as new fixed cooperation methods named `APPOINTMENT1`, `APPOINTMENT2` or `APPOINTMENT3`.
Whereas, here, emphasis is given to cooperation primitives and methods and to show their applicability to model a cooperative scenario like appointment scheduling, the next section is concerned with the basic procedure for making an appointment.

3.4 Basic Cooperation Strategies

To schedule and manage an appointment a wide variety of different conditions have to be considered. The whole spectrum of making appointments has to be supported, ranging from fully specified proposals up to very vague ones. For specification, the following parameters are of special importance:

- *participants*: Two types of participants can be distinguished: mandatory ones and optional ones. This principle distinction can be further generalized and handled by partial ordering. Another feature of participants is the differentiation between the status within an organisation and with respect to a certain meeting; it can also be important within the evaluation process. The specification of a group identifier instead of all names of these people as well as the different cases whether the initiator wants to attend or not, are further interesting questions in that context.

Initiator	Participants
APP_PREPARE_PROPOSAL	
send_app_proposal	
— **PROPOSE (appointment, exact_time, ...)** ⟶	
	receive_app_proposal
	APP_EVALUATE_PROPOSAL
	send_app_reply
⟵ **ACCEPT** —	
⟵ **REJECT** —	
⟵ **MODIFY** —	
receive_app_reply	
APP_EVALUATE_REPLIES	
— **ORDER (appointment, ...)** ⟶	
goto(send_app_proposal)	goto(send_app_reply)

Table 3.1: Optimistic Appointment Management

- *time*: This attribute is essential for the evaluation procedure. It can be a time point in which case the attribute *duration*, see below, has to be specified (e.g. tomorrow, 14:00), a continuous time interval (e.g. Tuesday, next week, 14:00 - 16:00) or a set of disjunct time intervals (every day next week in the morning) or even unspecified (e.g. soon, as soon as possible).
- *duration*: It can be a fixed value specifying the duration in hours or minutes (e.g. one hour) or an interval identifying an approximate duration (e.g. between two or three hours, less than two hours).
- *topic*: The topic is relevant for a participant's personal assessment of the importance to attend a meeting or not.
- *priority*: This value indicates the initiator's personal assessment of the meeting's importance. The priority value is also a measure how easy it would be to reschedule the meeting.

Optional parameters might be:

- *type*: Different kinds of gatherings are imaginable, e.g. meeting, appointment, visit, talk, conference, class. These different types can be associated with different default values of duration.
- *place*: Here, the initiator specifies where the meeting should take place.
- *frequency*: This attribute indicates whether the meeting should take place more than once and in what rotation (e.g. daily, weekly, monthly, every first Monday of a month, yearly).
- *general information*: The user can provide some free textual information about things concerning the meeting.

The basic negotiation procedure is mainly based on the priority of a meeting and on the authority relationships between the initiator and the participants of the meeting. The priority of a meeting specifies the human's individual preference to attend the meeting. With reference to the authority relationships mainly two different levels can be distinguished:

- superior-subordinate, e.g. head of department and employees of the department
- peer-to-peer, e.g. colleagues of a research group

In order to make an appointment, different strategies can be used: Within the *optimistic strategy* the initiator sends a request with a specific time schedule to all specified participants' appointment managers

Initiator	Participants
APP_PREPARE_PROPOSAL	
send_app_proposal	
— **PROPOSE (appointment, rough_time, ...)** ⟶	
	receive_app_proposal
	APP_EVAL_FREE_TIME
	send_app_info
⟵ **REFINE (free_time)** —	
receive_app_info	
APP_EVAL_INFO	
send_app_proposal	
— **PROPOSE (appointment, exact_time, ...)** ⟶	
	receive_app_proposal
	APP_EVALUATE_PROPOSAL
	send_app_reply
⟵ **ACCEPT** —	
⟵ **REJECT** —	
⟵ **MODIFY** —	
receive_app_reply	
APP_EVALUATE_REPLIES	
— **ORDER (appointment, ...)** ⟶	
goto(send_app_proposal)	goto(send_app_reply)
goto(app_prepare_proposal)	goto(send_app_info)

Table 3.2: Realistic Appointment Management, Centralized Method

Initiator	Participant_1	Participant_i	Participant_j
APP_PREPARE_PROPOSAL			
send_app_proposal			
— **PROPOSE (appointment, rough_time, ...)** ⟶			
	receive_app_proposal		
	APP_EVAL_FREE_TIME		
	send_app_reply		
⟵ REJECT —			
	— **REFINE (appointment, constrained_rough_time, ...)** ⟶		
		receive_app_refinement	
		APP_EVAL_FREE_TIME	
		send_app_reply	
		— **REFINE (appointment, constrained_rough_time, ...)** ⟶	
⟵ REJECT —			
			receive_app_refinement
			APP_EVAL_FREE_TIME
			send_app_reply
⟵ ACCEPT —			
⟵ REJECT —			
receive_app_reply			
APP_EVALUATE_REPLY			
— **ORDER (appointment, ...)** — ⟶ — ⟶ — ⟶			

Table 3.3: Realistic Appointment Management, Circular

which are a part of the respective user agents. hoping that they will accept.If all appointment managers accept within a given time constraint, their users are asked for confirmation of the meeting.[2] If a user rejects or if she is already occupied at the specified time, the appointment manager rejects the proposal providing the reason for rejecting. With that knowledge in mind, the initiator will, then, try another schedule. As is often the case with even a small group of three or four persons, there may be no free time slot. The specific, very constrained meeting proposal has to be relaxed; a more *realistic strategy* has to be tried out. Within the realistic strategy different procedures can be distinguished. They are based on different authority levels of the attending persons and on priorities to assess the importance of a meeting and perhaps to reschedule other ones.

[2]If humans are involved in a cooperation, the final decision should always remain within their responsibility.

In the following, two of these realistic strategies are considered in more detail. They differ mainly in the point where the actual planning and scheduling of appointments takes place. The one is a more *centralized* approach where the initiator (respectively her user agent) of a meeting collects all information about the participants and then evaluates it, the other resembles a *circular* which is started by the initiator; here, fixing of an appointment is done locally by each participant by constraining the possible time intervals.
Within the centralized approach, the initiator of the appointment specifies one or more rough time slots (a set of perhaps disjunct time intervals) within which the appointment ought to take place and provides a subject for the appointment that enables the other participants to validate the necessity of their attendance to the meeting. Further constraints may be added which state the initiator's personal preferences concerning the appointment. These preferences can be relaxed during negotiation. The initiator's appointment manager samples these data and starts a negotiation with the appointment managers of all tentative attendees by proposing the meeting specification.
The addressed appointment managers reply to the proposal by communicating all their free time slots that match the proposed time slot to the initiator.
If some user answers are missing after a given time period (maybe because of a connection failure), the initiator's appointment manager takes respective steps. It informs the user about the failure and awaits user commands for further proceeding. If the user wants the same action to be taken several times, the user agent may even 'learn' its user's behaviour and invoke the command automatically the next time such a failure happens.
Upon receipt of the replies the appointment manager of the initiating agent superimposes all the time slots to determine an appropriate time. Two results can emerge from the evaluation process: either a list of time intervals where all participants have free time, i.e. where the meeting could take place[3] or no time interval is found where the meeting could take place.
If a possible solution is found immediately the appointment is proposed for that specific time. After receipt of respective confirmation messages from the participants the meeting is set up by ordering the participants to plan for the meeting at that time.
If a participant rejects she provides a reason for rejection to the initiator. Depending on her preferences, her user agent automatically marks the time interval with that reason as occupied; otherwise, the time interval is treated as free in a next meeting proposal.
If we differentiate between mandatory and optional participants of a meeting, we need not to reschedule in case an optional participant resp. her/his user agent rejected. The meeting can be scheduled without her/him.
If a mandatory person rejects or in the case where no commonly free time interval is found by the initiator's user agent rescheduling of meetings has to take place. A possible procedure might look like follows:
First, the initiator's appointment manager calculates the time slot which entails minimal conflicts between mandatory participants; it then requests the mandatory conflicting participants to evaluate their personal relative importance of the proposed meeting related to the meeting they have already scheduled for this time slot. The conflicting participants inform the initiator's appointment manager about their assessment of the new meeting by sending one of the two possibilities:

- *higher*: The new appointment is more important than my old one.
- *less*: It is less important than my old one.

If all conflicting participants have assessed the priority of the new meeting higher then their old one, the initiator's appointment manager orders all agents to reschedule.
If some conflicting participants have assessed the actual meeting priority less than their old one, the authority relationships are considered[4]:
If the highest authority is with the initiator, its appointment manager can forcibly order all participants to attend the meeting after having examined the conflicting reasons. The initiator, however, also may

[3] If no other meeting has been scheduled for that time in the meantime.

[4] Since we want to design a 'democratic' appointment system, authority levels are only employed if there is no other alternative to come to an agreement.

decide to give up the current proposal in case the participants' rejecting reasons are too strong (e.g. on holidays at that time, visit of the firm's president, etc.). To reason about such cases, knowledge and rules about other persons and events must be modelled and treated in the humans' user agents. The initiator has to start a new proposal with relaxed and/or new time constraints.
If the initiator and one or more conflicting agents are in a peer-to-peer relation the process of ordering to reschedule can not take place. The initiator is informed by its appointment manager that he has to specify new time slots or relax the constraints, e.g. the attendance of certain participants.
We are aware of the fact that rescheduling of appointments is much more complex than described here. However, as already stated, the point of work for now was not to model a comprehensive and fully working appointment system but to show the flexibility and generality of the agent model and cooperation model for inherently distributed real world applications.

Chapter 4

Graphical Interface

In general, the *user interface* is the specialised hardware and software used for interaction between the user and the system. In our multi-agent view, it is a portion of the body of a user agent and is mainly responsible for

1. presenting the ongoing cooperation processes to the user,
2. providing a means to define new cooperation methods and
3. conducting an application-specific cooperation by requesting input and presenting output data.

In the following, we shortly describe how a user-friendly interface for scheduling appointments should look like.
To keep electronic appointment management as natural as possible, the user interface should present a graphical presentation of the person's calendar to set up meetings.
User input should be mouse-driven. By starting an appointment process an appointment window pops up to the initiator (see Figure 4.1).
At the moment, the window provides entry fields for entering/displaying the above mentioned meeting-related attributes 'participants' (multiple selection menu), 'duration' (integer value in minutes) and 'meeting subject'.
For now, time intervals to set up a meeting can be chosen within a week. The procedure is as follows: the user leafs through the calendar months by the 'Next'- resp. 'Prev'-Buttons; she then selects a day whose week in turn will be displayed in the week frame. Within the week frame the user can specify non-continuous time intervals as proposal for possible meeting time; e.g., in Figure 4.1 the meeting should either take place somewhere on Wednesday, the 19th, in the afternoon or on Thursday morning next day.
The 'Start Monitor'-Button can be pressed to initialize a monitoring process of the ongoing cooperation (see below).
Finally, with the 'Make Appointment'-Button in the upper left corner, a cooperation method can be selected and started.
At given times during the scheduling process, user input is requested, e.g. whether the user accepts a meeting proposal, whether he is willing to reschedule an appointment etc.; respective pop-up windows are created.
Whereas the appointment window is necessary for starting an appointment process, a monitor window is necessary to display the ongoing cooperation process to the user. This is necessary because the process of automatic appointment scheduling is thereby made transparent to the user. A user-readable trace of certain messages exchanged within the cooperation process should be presented. For example, a user may want to trace all messages

- sent to one or more participants or resource agents,
- received by one or more participants or resource agents,
- exchanged as part of the specific appointment scheduling cooperation process, or

- exchanged whenever a condition is satisfied (e.g. whenever rescheduling takes place).

The monitoring process is handled by one or more so-called *monitor agents*. They are a special type of agent controlling parts of the cooperation within the application.
Technical and implementational aspects of the scheduling process and the user interface are presented in the next chapter.

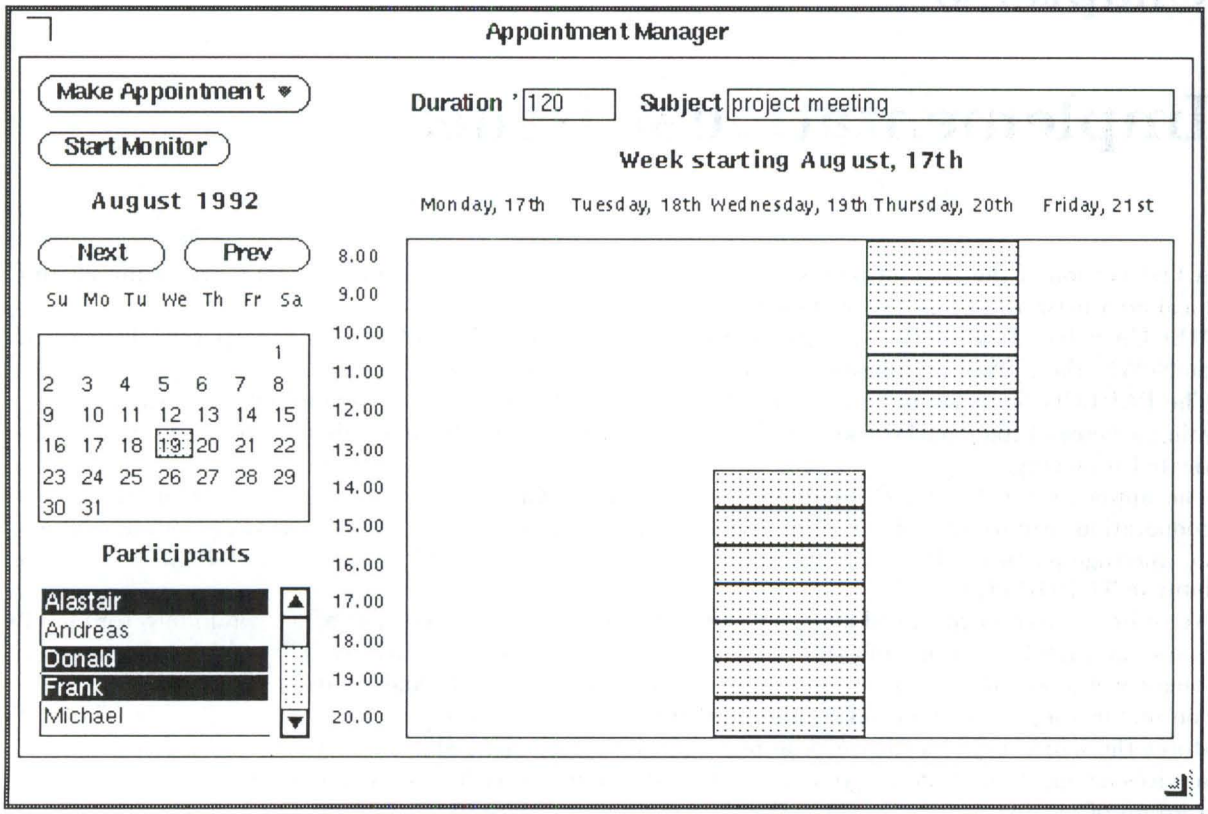

Figure 4.1: Appointment Manager - User Interface

Chapter 5

Implementational Issues

A first version of the appointment scheduling system has been implemented on MECCA[1], running on a local area network of Sun Sparc workstations.
MECCA is based on a logical programming system combining PARLOG and Prolog[CG87, Da90], and on NeWS, the Network extensible Window System (NeWS) from SUN.
The PARLOG language provides a natural and efficient embedding of parallelism into logic programming, whereas Prolog backtracking and meta-level programming facilities are used to implement sophisticated reasoning.
The application independent modules, i.e. a primary simple version of a general agent model, the cooperation primitives and the cooperation methods, are implemented in PARLOG. Communication, i.e. message passing between agents, takes place on top of the TCP/IP protocol with PARLOG/Prolog built-in TCP/IP primitives.
According to our agent model an agent is divided into three modules. The agent communicator mainly maps the high-level communication facilities down to TCP/IP primitives. The programmer need no longer worry about getting the network adresses of other agents and establishing connections. The communicator gets its information about other agents from an *agent directory service (ADS)*. The ADS stores the name, type and network adress of each agent registered at the particular ADS and also the services an agent offers to the system. There may be several ADS's running in a scenario each managing a group of agents. Currently the ADS's do not communicate to each other, but in future versions of MECCA they will be implemented as agents.
Incoming messages that trigger new cooperations are passed to the agent's head, where they are handled by a new process which becomes dedicated to that cooperation until the cooperation is finished. The head of the agent handles cooperations with other agents. If the agent is involved in more than one cooperation at a time, these are managed by several parallel Parlog processes. The cooperation methods are described in MECCA's "Parlog Meta Code", which is compiled down into Parlog. This code is still under development and will ease the way the programmer has to specify cooperation methods.
The agent's body may consist of Parlog, PROLOG, C or NeWS code. Body functions are usually called from within a cooperation method.
Humans participate in the appointment scenario by means of their user agents. Among other things the user agent provides a graphical user interface to set up a meeting and to show the flow of cooperation during appointment scheduling. The user interface is mainly implemented on the NeWS server side, and a control process on the Parlog side.[2] It allows the user to specify new tasks to the system in a graphical style. The user agent transforms the input from the user interface to a call of a specific cooperation method.
Main advantages of NeWS are:

- handling of most of the user interactions within the NeWS server by downloading code into the server, thus keeping the communication between server and client low

[1] Multi-Agent Environment for the Construction of Cooperative Applications

[2] The implementation of the NeWS user interface was done by F. Bomarius and M. Kolb.

- possibility to design and test the user interface independently from the application
- simple client-server communication allowing easy access from the PARLOG/Prolog system
- easy distribution across machines

One solution of providing a graphical user interface might have been to extend an existing calendar tool with respective hooks for making appointments. This, however, requires the source code of the tool. It also forces users of our system to use one particular tool, which might not be desirable, especially if they are used to using a different calendar tool.
This leads to a central point of our current implementation of an intelligent appointment system: the integration of existing calendar tools. This fact is important because of two reasons:

1. A mixture of different tools emphasizes the feasibility of our approach to model a general framework to construct multi-agent applications in human computer scenarios.
2. The user does not have to worry about yet another new calendar tool he is not well acquainted with. He can stay in his accustomed environment; he only has to learn how to make, reschedule or delete appointments.

However, taking this line, we had to specify corresponding interfaces for each of the calendar tools. Normally, each participant's data are stored in a specific calendar file located in his home directory. The interfaces have to read and write these files on the one hand; on the other hand, they have to use a common representation of appointment data which is understandable by the appointment scheduling algorithm and every calendar tool. The following main tasks of an interface can be identified:

1. read/write from/to the personal calendar file
2. cause the calendar tool to perform actions like updating, rereading or deleting data.
3. transform calendar tool specific data to a common *appointment format* and vice versa

At the moment, EMACS Calendar and Sun's Calentool are integrated into our appointment system. However, in fact every calendar tool whose calendar file is accessible and modifiable, could be integrated in the aforementioned way.
A last remark is dedicated to the problem of time. By describing the general appointment methods in Tables 3.1, 3.2, 3.3 we mentioned some functions as e.g. APP_EVALUATE_FREE_TIME which reside in each agent's head. The functions heavily rely on a representation of time uniform to all agents because time representation plays a central role in any appointment evaluation algorithm.
Humans' normal idea of time in terms of years, months, weeks, days and hours is not well suited for internal computer usage for reasons of storage. Comparisons between different time values would be complicated. Because time can be seen as a one-dimensional vector with arbitrary origin, time can be represented in a computer-manageable form as an integer value of minutes, e.g. integer value "1" can be assigned to time point "01.01.1992 0.00hrs."; then, the evaluation between integer values and time points and vice versa is straightforward.
For further details on the implementation, parts of the programming code are listed in Appendix A for those readers who are acquainted with PARLOG and Prolog.[3]

[3]M. Kolb contributed a part of this chapter.

Chapter 6

Conclusion and Outlook

In this report, we have shown that the multi-agent paradigm in combination with our proposed cooperation model is suitable for specifying inherently distributed cooperative scenarios like appointment management. This allows the integration of machines into human working environments as intelligent assistants. The integration is accomplished by the concept of the user agent, which can act on behalf of its user. A rough sketch towards implementation of a prototype system has been presented. We believe that the chosen approach is a promising one to build future sophisticated appointment management and calendar systems.
Based upon our experiences with developing this application and others [Bo92], the future global goal of our research group is to fully specify and develop *MAIL*, a formal multi-agent interaction language.

Acknowledgement

I would like to gratefully acknowledge F. Bomarius, A. Burt, M. Kolb and D. Steiner of the KIK-Teamware group for their fruitful discussions on this topic and their help on implementational work. This work was partially supported by the European Community as part of ESPRIT II Project 5362, IMAGINE (Integrated Multi-Agent Interactive Environment).

Appendix A

Selected Source Code

Cooperation Methods

```
%
% Opportunistic Appointment Scheduling implementation
%

% Authors: A. Lux, D. Steiner

%  initiator's role
mode optimistic(Coopid?,Agentlist?,TimeInterval?,Timeout?).
optimistic(Coopid,Agents,meet(D,TimeInterval,S),Timeout) <-
    % first propose from initiator to the user agents
    propose(Coopid,Agents,optimistic(meet(D,TimeInterval,S)),_) &
    % Answers from User Agents
    recv(Coopid,Agents,Answerlist,Timeout) &
    opti_continue(Coopid,Agents,TimeInterval,D,S,Answerlist).

mode opti_continue(?,?,TI?,D?,S?,Answerlist?).
% Acceptable to all user-agents
opti_continue(Coopid,Agents,Int,D,S,[(_, accept(Int))]) <-
    %second propose from initiator, now to the users
    propose(Coopid,Agents,meet(D,Int,S),_) &
    recv(Coopid,Agents,UserAnswers,30) &
    opti_process_user_answers(Coopid,Agents,Int,D,S,UserAnswers) :
    true;
% Acceptable to the next user-agent in the list
opti_continue(Coopid,Agents,Int,D,S,[(_, accept(Int)) | Rest]) <-
    opti_continue(Coopid,Agents,Int,D,S,Rest) : true;
% Some user-agent rejected, thus must reject all agents.
opti_continue(Coopid,Agents,Int,D,S,[(F, _) | _]) <-
    write(F) &
    write('s user agent rejected: Optimistic method failed.') &
    nl &
    reject(Coopid,Agents,Int,_).

mode opti_process_user_answers(?,?,?,?,?,UserAnswers?).
% Acceptable to all users
opti_process_user_answers(Coopid,Agents,Int,D,S,[(_, accept(Int))]) <-
    order(Coopid,Agents,(Int,D,S),_) &
    prolog(make_entry(Int,D,S)) : true;
```

```
% Acceptable to the next user in the list
opti_process_user_answers(Coopid,Agents,Int,D,S,[(_, accept(Int)) | Rest]) <-
    opti_process_user_answers(Coopid,Agents,Int,D,S,Rest) : true;
% Some user rejected, thus must reject all agents.
opti_process_user_answers(Coopid,Agents,Int,D,S,[(F, _) | _]) <-
    write(F) &
    write(' rejected: Optimistic method failed.') &
    nl &
    reject(Coopid,Agents,Int,_).

%  participant's role

mode optimistic(Msg?).
optimistic(Msg) <-
% Participant's user agent receives meeting proposal from initiator
    Msg = [Coopid,From,propose(meet(D,TimeInterval,S))] &
% user agent evaluates personal calendar
    prolog(evaluate_time(TimeInterval,D,S,Answer)) &
% user agent sends answer to initiator
    formulate_answer(Coopid,From,Answer,TimeInterval) &
% User herself now receives meeting proposal from initiator
    recv(Coopid,From,Msg1) &
% User accepts or rejects proposal
    eval_reply(Coopid,From,Msg1) &
% User receives final message whether meeting takes place or not
    recv(Coopid,From,Msg2) &
    eval_reply(Coopid,From,Msg2).
```

```
%
% Realistic Appointment Scheduling implementation
%

% Authors: A. Lux, D. Steiner

%  initiator's role
mode realistic(Coopid?,Agentlist?,TimeInterval?,TimeOut?).
realistic(Coopid,Agents,meet(Duration,TimeInterval,Subject),Timeout)
<-
    % Proposal is sent to user agents
    propose(Coopid,Agents,realistic(meet(Duration,TimeInterval,Subject)),_) &
    recv(Coopid,Agents,Msgs,Timeout) &
    % user agents have refined possible time slots
    extract_busy_slots(Msgs,BusySlotList) &
    Duration = [Dh,Dm] &
    Appt = appt(Dh, Dm, Subject) &
    prolog(findall(PosTime,
    schedule(Appt,TimeInterval,BusySlotList,PosTime),
    PosTimes)) &
    % loop over the list of times potentially acceptable to all users
    reali_try_times(Coopid,Agents,Duration,Subject,PosTimes).

mode reali_try_times(Coopid?,Agents?,D?,S?,PosTimes?).
reali_try_times(Coopid,Agents,D,S,[PosTime | RestTimes]) <-
% Proposed time (as a list) is sent to users themselves
    propose(Coopid,Agents,meet(D,[PosTime],S),_) &
    % user answers are either accept or reject
    recv(Coopid,Agents,UserAnswers,30) &
    reali_continue(Coopid,Agents,D,S,[PosTime],UserAnswers,RestTimes).

% Scheduling algorithm found no possible time for the meeting
reali_try_times(Coopid,Agents,_,_,[]) <-
    write('No available time in users calendars.') & nl &
    reject(Coopid,Agents,[],_).

mode reali_continue(?,?,?,?,?,?,?).
% all users accepted
reali_continue(Coopid,Agents,D,S,TimeInt,[(_, accept(TimeInt))],RestTimes) <-
    order(Coopid,Agents,(TimeInt,D,S),_) &
    prolog(make_entry(TimeInt,D,S)) : true;
% acceptable to the next user in the list
reali_continue(Coopid,Agents,D,S,TimeInt,[(_, accept(TimeInt))|Rest],
RestTimes) <-
    reali_continue(Coopid,Agents,D,S,TimeInt,Rest,RestTimes) : true;
% a user rejected a proposed meeting time
reali_continue(Coopid,Agents,D,S,PosTime,[(From,_) | _],RestTimes) <-
    write(From) & write(' rejected ') & write(PosTime) & nl &
    reali_try_times(Coopid,Agents,D,S,RestTimes).

%  participant's role

mode realistic(Msg?).
realistic(Msg) <-
```

```
    % first propose from initiator
    Msg = [Coopid,From,propose(meet(D,TimeInterval,S))] &
    prolog(evaluate_entries(meet(D,TimeInterval,S),Entries,Answer)) &
% refinement of the given time slots
    formulate_answer(Coopid,From,Answer,Entries) &
% evaluation of a possible meeting time
    reali_eval_postimes(Coopid, From).

mode reali_eval_postimes(?,?).
% loop over a list of possible times for a meeting
reali_eval_postimes(C,F) <-
    recv(C,F,Msg) &
    reali_eval_postimes_sw(C,F,Msg).

mode reali_eval_postimes_sw(?,?,Msg?).
% received message is a meeting proposal to the user
reali_eval_postimes_sw(C,F,propose(meet(D,PossibleTime,S))) <-
    eval_reply(C,F,propose(meet(D,PossibleTime,S))) &
    reali_eval_postimes(C,F) : true;

% stop, if user receives any other message (will be either reject or
% order for meeting)
reali_eval_postimes_sw(C,F,Msg) <-
    eval_reply(C,F,Msg).

% used by both opti and reali
mode formulate_answer(?,?,?,?).
% possible answers of the user or user agent
formulate_answer(Coopid,From,refine,TimeInterval) <-
    refine(Coopid,From,TimeInterval,_).
formulate_answer(Coopid,From,accept,TimeInterval) <-
    accept(Coopid,From,TimeInterval,_).
formulate_answer(Coopid,From,reject,TimeInterval) <-
    reject(Coopid,From,TimeInterval,_).

% First propose of time intervals do NOT get passed through
% eval_reply.

mode eval_reply(Coopid?,From?,Msg?).

% At some point some user or user agent didn't accept, so the whole
% process is cancelled.
eval_reply(_,_,reject(TimeInterval)) <-
    write(TimeInterval) & write(rejected) & nl;

% All is clear - all user agents and users agree to the time, so they
% are ordered to keep it.
eval_reply(_,_,order((TimeInterval,D,S))) <-
   write('Update appointment file with ') & write(TimeInterval) & nl &
    write('Duration ') & write(D) & write(' with subject ')
& write(S) & nl &
    prolog(make_entry(TimeInterval,D,S));


% This is a final propose in the reali method, to be sent to the user.
```

```
% His/her reply determines the answer (reject or accept) to be returned.
eval_reply(Coopid,From,propose(meet(D,PosTime,Subject))) <-
    D = [Dh,Dm] &
    Appt = appt(Dh,Dm,Subject) &
    time_ok(PosTime,Appt,Answer) &
    formulate_answer(Coopid,From,Answer,PosTime).

mode extract_busy_slots(Msgs?, BS^).
extract_busy_slots([],[]);
extract_busy_slots([(_,refine([])) | RestMsgs], RestBS) <-
    extract_busy_slots(RestMsgs, RestBS);
extract_busy_slots([(From,refine([entry(T1-T2,E) |
RestEntries]))|RestMsgs], [entry(T1-T2,E) | RestBS]) <-
    extract_busy_slots([(From,refine(RestEntries)) | RestMsgs], RestBS).

%% Some functions for checking with the user. Used by both opti and
%% reali.

mode time_ok(?,^).
time_ok(Time,Appt,Answer) <-
    prolog(ok_with_diary(Time,Appt)) :
    ok_with_user(Time,Appt,Answer);
time_ok(_,_,reject) <-              %% ok_with_diary failed
    write('Not ok with diary!').

% Appointment is of form appt(Dh,Dm,Subject)
% Answer is either reject or accept.
mode ok_with_user(Time?,Appt?,Answer^).
ok_with_user(Time,Appt,Answer) <-
query_user(['Meeting at time', Time, '?'], UserAnswer) &
process_user_answer(UserAnswer,Time,Appt,Answer).

% User accepted
mode process_user_answer(UserAnswer?,Time?,Appt?,^).
process_user_answer(accept,Time,Appt,accept) <-
    write('User accepted appointment ') & write(Appt) &
    write(' within time ') & write(Time) & nl.

% User rejected with reason
process_user_answer(reject(Reason),Time,_,reject) <-
    write('User rejected because of ') & write(Reason) & nl &
 % Should insert this time in user's diary file.
    prolog(ti_duration(Time,D)) &
    write('Entry for Reason ') & write(Reason) &
    write(' at time ') & write(Time) &
    write(' with duration ') & write(D) & write(' is made.') & nl &
    prolog(make_entry(Time,D,Reason)).
```

User agent functions

```
% Author: A. Lux

% The body of a user agent!

init_user_agent :- consult(user_db),consult(time),
                   load([calentool, scheduler, icp, emacs]).

% e.g. user_db entries for specific users and their respectively used
% calendar tool

appt_sys([frank,emacs]).
appt_sys([andi,calentool]).
appt_sys([al,emacs]).
appt_sys([don,calentool]).
appt_sys([mike,calentool]).


% The head of a user agent!

% Calendar tool specific values!
calfile(U,File_Ext,Fun_Ext,Cal_Dir,Entry_Fun,Make_Fun) :-
appt_sys([U, calentool]),
File_Ext = '.app',
Fun_Ext = 'calentool',
Cal_Dir = '/home/lux/ademo/',
Entry_Fun = 'calentool_entries',
Make_Fun = 'ct_append_entry',
writeseqnl([U, 'has File_Ext', File_Ext, 'and Fun_Ext', Fun_Ext]), ! ;
appt_sys([U, emacs]),
File_Ext = '.dy',
Fun_Ext = 'emacs',
Cal_Dir = '/home/lux/ademo/',
Entry_Fun = 'emacs_entries',
Make_Fun = 'em_append_entry',
writeseqnl([U, 'has File_Ext', File_Ext, 'and Fun_Ext',Fun_Ext]), ! ;
appt_sys([U,_]),
File_Ext = '.x',
Fun_Ext = 'zzz',
writeseqnl([U, 'has File_Ext', File_Ext, 'and Fun_Ext',Fun_Ext]).


% Time slot evaluation function

evaluate_time(TimeInterval,D,S,accept) :-
D = [Dh,Dm],
Appt = appt(Dh,Dm,S),
myself(Name,_),
calfile(Name,FiE,_,Dir,EntryF,_),
str_concat([Dir, Name, FiE], CTAppFile) ,
Call1 =.. [EntryF, Name, CTAppFile, TimeInterval, Entries],
call(Call1),
writeseqnl([Appt, TimeInterval, Entries]),
        schedule(Appt,TimeInterval,Entries,_).
```

```
evaluate_time(_,_,_,reject).

evaluate_entries(meet(Duration,TimeInterval,Subject),Entries,refine) :-
Duration = [DH,DM] ,
Appt = appt(DH, DM, Subject) ,
write(TimeInterval) , nl ,myself(Name,_),
calfile(Name,FiE,_,Dir,EntryF,_),
str_concat([Dir, Name, FiE], CTAppFile) ,
Call1 =.. [EntryF, Name, CTAppFile, TimeInterval, Entries],
call(Call1),
writeseqnl([Appt, TimeInterval, Entries]).


ok_with_diary(TimeInterval,Appt) :-
myself(Name,_),
calfile(Name,FiE,_,Dir,EntryF,_),
str_concat([Dir, Name, FiE], CTAppFile) ,
Call1 =.. [EntryF, Name, CTAppFile, TimeInterval, Entries],
call(Call1),
        setof(Sol,schedule(Appt,TimeInterval,Entries,Sol), S_List).

make_entry([T1 - _],D,S) :-
myself(Name,_),
D = [Dh,Dm],
Appt = appt(Dh,Dm,S),
calfile(Name,FiE,_,Dir,_,MakeF),
str_concat([Dir, Name, FiE], CTAppFile) ,
Call1 =.. [MakeF, CTAppFile, T1, Appt],
call(Call1).


ti_duration([T1 - T2],[Hours,Mins]) :-
    gregorian_to_absolute(T1,Abs1),
    gregorian_to_absolute(T2,Abs2),
    Hours is (Abs2 - Abs1) // 60,
    Mins is (Abs2 - Abs1) mod 60.
```

Simple Scheduling Algorithm

```
%% Author: A. Burt

%% Convert from an intelligible time representation to integers, schedule,
%% then convert the resulting time interval back from the integer
%% form; at the moment, no rescheduling is done

%% "target intervals" are those in which it is desirable to have an
%% appointment; "busy intervals/slots" are the inverse, we do not want the
%% new appointment to occur then.


schedule(appt(Hours, Mins, _), TargetIntervals, BusySlots, Time-Time1):-
    maplist(gregorian_to_absolute, TargetIntervals, AbsTargetIntervals),
    maplist(gregorian_to_absolute, BusySlots, AbsBusySlots),
    Duration is ((Hours * 60) + Mins),
    sched(Duration, AbsTargetIntervals, AbsBusySlots, AbsTime-AbsTime1),
    absolute_to_gregorian(AbsTime, Time),
    absolute_to_gregorian(AbsTime1, Time1).


%%  sched(Duration?, List(TargetTimeInterval)?, List(BusySlots)?,
%%  ResultInterval^).

sched(D, TI, BSs, RI):-
    duration(D, D1), % may want to backtrack
    time_interval(TI, TA, TO), % ditto

    %% pick a starting time: ST

    (
TA = ST
      ;
%% May want to replace calls to member/2 with a predicate that
%% filters out certain busy intervals --- e.g. those with a low
%% priority.
member(BS, BSs),
end_point(BS, ST),
before(TA, ST) % We have already tried
% TA = ST
    ),
    add_duration(D1, ST, ET),
    before_eq(ET, TO),

    %% check that no start point or end point of a busy slot occurs in the
    %% time ST to (ST + Duration)

    \+ (
member(BS1, BSs),
start_point(BS1, SP),
end_point(BS1, EP),
(
    before(SP, ET),
    before_eq(ST, SP)
```

```
  ;
    before(EP, ET),
    before(ST, EP)
)
    ),

    %% find the biggest free interval, i.e. the result interval, starting
    %% from ST; the end point of this interval will be either the end of
    %% one of the target intervals, i.e. time omega, or the beginning of a
    %% busy slot.

    (
ERI = TO
      ;
member(BS2, BSs),
start_point(BS2, ERI),
before_eq(ET, ERI),
before(ERI, TO) % We have already tried
% ERI = TO
    ),

    %% check that no busy slot starts during the result interval

    \+ (
member(BS3, BSs),
start_point(BS3, SP1),
\+ (
    (
SP1 = ERI
      ;
before(SP1, ST)
      ;
before(ERI, SP1)
    )
)
    ),
    result_interval(ST, ERI, RI).

% For later changes in time repesentation / backtracking

duration(D, D).
time_interval(TIs, TA, TO):-
    member(TA-TO, TIs).
result_interval(ST, ET, ST-ET).

%% Handling Time

add_duration(X, Y, Z):-
    Z is X + Y.
before(X, Y):-
    X < Y.
before_eq(X, Y):-
    X =< Y.

%% Slots
```

```
start_point(entry(T-_,_), T).
end_point(entry(_-T, _), T).

end_point_before(X, Y):-
    end_point(X, XEP),
    end_point(Y, YEP),
    before(XEP, YEP).

maplist(_, [], []).
maplist(F, [X-X1|List], [Y-Y1|List1]):-
    Call =.. [F, X, Y],
    call(Call),
    Call1 =.. [F, X1, Y1],
    call(Call1),
    maplist(F, List, List1).
maplist(F, [entry(X-X1, E)|List], [entry(Y-Y1, E)|List1]):-
    Call =.. [F, X, Y],
    call(Call),
    Call1 =.. [F, X1, Y1],
    call(Call1),
    maplist(F, List, List1).
```

Calentool Interface Functions

```
% Authors: A. Burt, A. Lux

% Creating a Calentool file with all entries within the proposed time
% intervals


calentool_entries(UserName, ApptFile, TargetIntervals, Entries):-
    extract_days(TargetIntervals, [] , Days),
    timestamp(Time),
    str_concat(['/tmp/', UserName, Time, 'caldump'], FileName),
    create_calentool_file1(FileName, ApptFile, Days),
    read_file(FileName, Chars), !,
    str_concat(['rm ', FileName], Cmd),
    unix_cmd(Cmd),
    phrase(ct_entries(Entries), Chars, []).

create_calentool_file1(_, _, []).
create_calentool_file1(FileName, ApptFile, [date(Year, Month, Day)|Days]):-
    Year1 is (Year mod 100),
    str_concat(['calentool -E -d ', Day, '/', Month, '/', Year1,
' -f ', ApptFile, ' -pd >> ', FileName], Cmd),
    unix_cmd(Cmd),
    create_calentool_file1(FileName, ApptFile, Days).



% Parsing the file created above, extracting the entries for further
% treatment in the user agent head

ct_entries(Entries) -->
    ct_initial_blurb,
    ct_entries_for_day(Entries, Entries1), !,
    (ct_notes -> [] ; []),
    ct_final_blurb,
    ct_entries(Entries1).
ct_entries([]) -->
    [].

ct_entries_for_day(Entries, Entries1) -->
    (
ct_entry(Entry)
      ->
{Entries = [Entry|Entries2]},
ct_entries_for_day(Entries2, Entries1)
      ;
[], {Entries = Entries1}
    ).

ct_entry(entry(TimeInterval,String)) -->
    ct_time(TimeInterval),
    "   ",
    ct_entry_string(String).
```

```
ct_initial_blurb -->
    [10],
    ct_blurb1,
    [10, 10].

ct_blurb1 -->
    [Char],
    {Char =\= 10, !},
    ct_blurb1.
ct_blurb1 -->
    [].

ct_final_blurb -->
    [10],
    ct_dashes,
    [10].

ct_notes -->
    [10],
    "      ===== Notes =====",
    [10],
    ct_notes1.

ct_notes1 -->
    not_ten, [10],
    ct_notes1.
ct_notes1 -->
    [].

not_ten -->
    [C], {C =\= 10},
    not_ten1.

not_ten1 -->
    [C], {C =\= 10}, !,
    not_ten1.
not_ten1 -->
    [].

ct_dashes -->
    ("-" -> ct_dashes; []).

ct_time(time(Year, Month, Day, H, M) - time(Year, Month, Day, H1, M1)) -->
    ct_dayname,
    " ",
    ct_date(Year, Month, Day),
    " -- ",
    ct_hours(time_of_day(H,M) - time_of_day(H1, M1)).

ct_dayname -->
    "Mon".
ct_dayname -->
   "Tue".
ct_dayname -->
    "Wed".
```

```
ct_dayname -->
    "Thu".
ct_dayname -->
    "Fri".
ct_dayname -->
    "Sat".
ct_dayname -->
    "Sun".

ct_date(Year, Month, Day) -->
    ct_day(Day),
    "/",
    ct_month(Month),
    "/",
    ct_year(Year).

ct_day(Day) -->
    ct_integer(N),
    ct_integer(N1), !,
    {Day is (10 * N) + N1} .
ct_day(Day) -->
    " ",
    ct_integer(Day).

ct_month(Month) -->
    ct_integer(N),
    ct_integer(N1),
    {Month is (10 * N) + N1} .

ct_year(Year) -->
    ct_integer(N),
    ct_integer(N1),
    {Year is (10 * N) + N1 + 1900} .

ct_integer(N) -->
     [C],
     {C >= 0'0, C =< 0'9, N is C - 0'0 }.


ct_hours(time_of_day(H,M) - time_of_day(H1, M1)) -->
    ct_hour(H),
    ":",
    ct_integer(N2),
    ct_integer(N3),
    " to ",
    ct_hour(H1),
    ":",
    ct_integer(N6),
    ct_integer(N7),
    {M is (N2 * 10) + N3, M1 is (N6 * 10) + N7}.

ct_hour(H) -->
    ct_integer(I1),
    ct_integer(I2), !,
    {H is (I1 * 10) + I2}.
```

```
ct_hour(H) -->
    " ",
    ct_integer(H).


ct_entry_string(X) -->
    ct_entry_string1(Cs),
    {icp_hack(X, Cs)}.

ct_entry_string1([Char|Cs]) -->
    [Char],
    {Char =\= 10},
    !,
    ct_entry_string1(Cs).
ct_entry_string1([]) -->
    [10].


% Appending new entries to a Calentool File

ct_append_entry(File, time(Y, M, D, H, Mi), appt(H1, Mi1, Description)):-
    YY is (Y mod 100),
    ApproxDuration is (((H1 * 60) + Mi1) // 30),
    (0 is (((H1 * 60) + Mi1) mod 30) -> Duration is ApproxDuration - 1;
     Duration is ApproxDuration),
    ct_add_leading_zero(YY, Y0),
    ct_add_leading_zero(M, M0),
    ct_add_leading_zero(D, D0),
    ct_add_leading_zero(H, H0),
    ct_add_leading_zero(Mi, Mi0),
    ct_add_leading_zero(Duration, Duration0),
    append_sequence(File, [Y0, M0, D0, H0, Mi0, Duration0, Description]).
```

Bibliography

[Ap86] Applix Inc. Alis from Applix. User's Guide. Westboro MA. 1986.

[Bo92] F. Bomarius. Multi-Agent Model of an Urban Traffic Scenario. DFKI Research Report RR-92-xx. 1992. (to appear)

[BG88] A.H. Bond, L. Gasser. Readings in Distributed Artificial Intelligence. Morgan Kaufmann Publishers, San Mateo, CA. 1988.

[BPH+90] D. Beard, M. Palanlappan, A. Humm, D. Banks, A. Nair, Y.-P. Shan. A Visual Calendar For Scheduling Group Meetings. *in: Proceedings of the Conference on CSCW. Los Angeles. 1990.*

[BR84] K. H. Beckurts, R. Reichwald. Cooperation in the Management Area with Integrated Office Technics. CW Publications. Munich. 1984. (in German)

[Ci83] J.J. Cimral. Integrating coordination support into automated information systems. Master's Thesis. Dept. of Electric Engineering and Computer Science. Massachusetts Institute of Technology. Cambridge. 1983.

[CG87] K. S. Clark, S. Gregory. Parlog and Prolog United. *Proceedings of the 4th International Logic Programming Conference.* Melbourne, Australia. 1987.

[Da90] A. Davison. Hermes: A combination of Parlog and Prolog. ESPRIT Projekt 5362 IMAGINE. Technical Report #3. August 1990.

[Eh87a] S. F. Ehrlich. Social and Psychological Factors influencing the Design of Office Communication Systems. *in: Proceedings of the CHI+GI'87 on Human Factors in Computing Systems.* Toronto. April 1987.

[Eh87b] S. F. Ehrlich. Strategies for Encouraging Successful Adoption of Office Communication Research. *in: ACM Transactions on Office Information Systems, Vol. 5, No. 4, p.340-357.* Oct. 1987.

[EE92] N. Eisinger, N. Elshiewy. MADMAN - Multi-Agent Diary Manager. *in: Proceedings of the DAI Workshop at European Conference on AI.* August 1992.

[GMN+91] P. de Greef, D. Mahling, M. Neerincx, S. Wyatt. Analysis of Human Computer Cooperative Work. ESPRIT Projekt 5362 IMAGINE. Technical Report #4. July 1991.

[Gr84] I. Greif. The user interface of a personal calendar program. *in: Human Factors and Interactive Computer Systems.* Ablex. Norwood, N.J. 1984.

[GS87] I. Greif, S. Sarin. Data Sharing in Group Work. *in: TOOIS 5(2), pp. 187-211.* April 1987.

[HA85] C. Holman, G. Almes. The Eden Shared Calendar System. Department of Computer Science, University of Washington, Seattle WA. 1985.

[KDK85] C.M. Kincaid, P.B. Dupont, A.R. Kaye. Electronic Calendars in the Office: An Assessment of User Needs and Current Technology. *in: ACM Transactions on Office Information Systems, Vol. 3, No. 1, Jan. 1985.*

[LBS92] A. Lux, F. Bomarius, D. Steiner. A Model for Supporting Human Computer Cooperation. *in: Proceedings of the AAAI Workshop on Cooperation among Heterogeneous Intelligent Systems. San Jose, CA. July 1992.*

[MS88] F. Mattern, P. Sturm. Distributed Programming Concepts. Experiences made by Development of a Decentralized Appointment System. Technical Report SFB124-30/88. Department of Computer Science. University of Kaiserslautern. Germany. 1988.

[MS89] F. Mattern, P. Sturm. An Automatic Distributed Calendar and Appointment System. Technical Report SFB124-24/89. Department of Computer Science. University of Kaiserslautern. Germany. 1989.

[SD92a] S. Sen, E. Durfee. A Formal Analysis of Communication and Commitment in Distributed Meeting Scheduling. *in: Eleventh International Workshop on DAI. The Homestead. Glen Arbor, MI. February 25-29,1992.*

[SD92b] S. Sen, E. Durfee. Automated Meeting Scheduling among Heterogeneous Agents. *in: Proceedings of the AAAI Workshop on Cooperation among Heterogeneous Intelligent Systems. San Jose, CA. July 1992.*

[SG85] S. Sarin, I. Greif. Computer-based real-time conferences. *in: IEEE Computer, Vol.18, No.10, 1985.*

[SMH90] D. Steiner, D. Mahling, and H. Haugeneder. Human Computer Cooperative Work. In *Proc. of the 10th International Workshop on Distributed Artificial Intelligence*, MCC Technical Report ACT-AI-355-90, Austin/TX, 1990.

[WF86] T. Winograd, F. Flores. Understanding Computers and Cognition. A New Foundation for Design. Ablex. Norwood. 1986.

[Wo91] M. Woitass. Coordination in Structured Conversations. PhD. thesis. GMD Report No. 190. Oldenbourg. Munich/Vienna. 1991. (in German)

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
D-6750 Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse bezogen werden.
Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all publisched papers so far can be ordered from the above address.
The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-91-24
Jochen Heinsohn: A Hybrid Approach for Modeling Uncertainty in Terminological Logics
22 pages

RR-91-25
Karin Harbusch, Wolfgang Finkler, Anne Schauder:
Incremental Syntax Generation with Tree Adjoining Grammars
16 pages

RR-91-26
M. Bauer, S. Biundo, D. Dengler, M. Hecking, J. Koehler, G. Merziger:
Integrated Plan Generation and Recognition
- A Logic-Based Approach -
17 pages

RR-91-27
A. Bernardi, H. Boley, Ph. Hanschke, K. Hinkelmann, Ch. Klauck, O. Kühn, R. Legleitner, M. Meyer, M. M. Richter, F. Schmalhofer, G. Schmidt, W. Sommer:
ARC-TEC: Acquisition, Representation and Compilation of Technical Knowledge
18 pages

RR-91-28
Rolf Backofen, Harald Trost, Hans Uszkoreit:
Linking Typed Feature Formalisms and Terminological Knowledge Representation Languages in Natural Language Front-Ends
11 pages

RR-91-29
Hans Uszkoreit: Strategies for Adding Control Information to Declarative Grammars
17 pages

RR-91-30
Dan Flickinger, John Nerbonne:
Inheritance and Complementation: A Case Study of *Easy* Adjectives and Related Nouns
39 pages

RR-91-31
H.-U. Krieger, J. Nerbonne:
Feature-Based Inheritance Networks for Computational Lexicons
11 pages

RR-91-32
Rolf Backofen, Lutz Euler, Günther Görz:
Towards the Integration of Functions, Relations and Types in an AI Programming Language
14 pages

RR-91-33
Franz Baader, Klaus Schulz:
Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures
33 pages

RR-91-34
Bernhard Nebel, Christer Bäckström:
On the Computational Complexity of Temporal Projection and some related Problems
35 pages

RR-91-35
Winfried Graf, Wolfgang Maaß: Constraint-basierte Verarbeitung graphischen Wissens
14 Seiten

RR-92-01
Werner Nutt: Unification in Monoidal Theories is Solving Linear Equations over Semirings
57 pages

RR-92-02
Andreas Dengel, Rainer Bleisinger, Rainer Hoch, Frank Hönes, Frank Fein, Michael Malburg:
Π_{ODA}: The Paper Interface to ODA
53 pages

RR-92-03
Harold Boley:
Extended Logic-plus-Functional Programming
28 pages

RR-92-04
John Nerbonne: Feature-Based Lexicons:
An Example and a Comparison to DATR
15 pages

RR-92-05
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner, Michael Schulte, Rainer Stark:
Feature based Integration of CAD and CAPP
19 pages

RR-92-06
Achim Schupetea: Main Topics of DAI: A Review
38 pages

RR-92-07
Michael Beetz:
Decision-theoretic Transformational Planning
22 pages

RR-92-08
Gabriele Merziger: Approaches to Abductive Reasoning - An Overview -
46 pages

RR-92-09
Winfried Graf, Markus A. Thies:
Perspektiven zur Kombination von automatischem Animationsdesign und planbasierter Hilfe
15 Seiten

RR-92-10
M. Bauer: An Interval-based Temporal Logic in a Multivalued Setting
17 pages

RR-92-11
Susane Biundo, Dietmar Dengler, Jana Koehler:
Deductive Planning and Plan Reuse in a Command Language Environment
13 pages

RR-92-13
Markus A. Thies, Frank Berger:
Planbasierte graphische Hilfe in objektorientierten Benutzungsoberflächen
13 Seiten

RR-92-14
Intelligent User Support in Graphical User Interfaces:
1. InCome: A System to Navigate through Interactions and Plans
   Thomas Fehrle, Markus A. Thies
2. Plan-Based Graphical Help in Object-Oriented User Interfaces
   Markus A. Thies, Frank Berger

22 pages

RR-92-15
Winfried Graf: Constraint-Based Graphical Layout of Multimodal Presentations
23 pages

RR-92-16
Jochen Heinsohn, Daniel Kudenko, Berhard Nebel, Hans-Jürgen Profitlich: An Empirical Analysis of Terminological Representation Systems
38 pages

RR-92-17
Hassan Aït-Kaci, Andreas Podelski, Gert Smolka:
A Feature-based Constraint System for Logic Programming with Entailment
23 pages

RR-92-18
John Nerbonne: Constraint-Based Semantics
21 pages

RR-92-19
Ralf Legleitner, Ansgar Bernardi, Christoph Klauck
PIM: Planning In Manufacturing using Skeletal Plans and Features
17 pages

RR-92-20
John Nerbonne: Representing Grammar, Meaning and Knowledge
18 pages

RR-92-21
Jörg-Peter Mohren, Jürgen Müller
Representing Spatial Relations (Part II) -
The Geometrical Approach
25 pages

RR-92-22
Jörg Würtz: Unifying Cycles
24 pages

RR-92-23
Gert Smolka, Ralf Treinen:
Records for Logic Programming
38 pages

RR-92-24
Gabriele Schmidt: Knowledge Acquisition from Text in a Complex Domain
20 pages

RR-92-25
Franz Schmalhofer, Ralf Bergmann, Otto Kühn, Gabriele Schmidt: Using integrated knowledge acquisition to prepare sophisticated expert plans for their re-use in novel situations
12 pages

RR-92-26
Franz Schmalhofer, Thomas Reinartz, Bidjan Tschaitschian: Intelligent documentation as a catalyst for developing cooperative knowledge-based systems
16 pages

RR-92-27
Franz Schmalhofer, Jörg Thoben: The model-based construction of a case-oriented expert system
18 pages

RR-92-29
Zhaohur Wu, Ansgar Bernardi, Christoph Klauck: Skeletel Plans Reuse: A Restricted Conceptual Graph Classification Approach
13 pages

RR-92-33
Franz Baader: Unification Theory
22 pages

RR-92-34
Philipp Hanschke: Terminological Reasoning and Partial Inductive Definitions
23 pages

RR-92-35
Manfred Meyer:
Using Hierarchical Constraint Satisfaction for Lathe-Tool Selection in a CIM Environment
18 pages

RR-92-36
Franz Baader, Philipp Hanschke: Extensions of Concept Languages for a Mechanical Engineering Application
15 pages

RR-92-37
Philipp Hanschke:
Specifying Role Interaction in Concept Languages
26 pages

RR-92-38
Philipp Hanschke, Manfred Meyer: An Alternative to H-Subsumption Based on Terminological Reasoning
9 pages

RR-92-41
Andreas Lux: A Multi-Agent Approach towards Group Scheduling
32 pages

DFKI Technical Memos

TM-91-11
Peter Wazinski: Generating Spatial Descriptions for Cross-modal References
21 pages

TM-91-12
Klaus Becker, Christoph Klauck, Johannes Schwagereit: FEAT-PATR: Eine Erweiterung des D-PATR zur Feature-Erkennung in CAD/CAM
33 Seiten

TM-91-13
Knut Hinkelmann:
Forward Logic Evaluation: Developing a Compiler from a Partially Evaluated Meta Interpreter
16 pages

TM-91-14
Rainer Bleisinger, Rainer Hoch, Andreas Dengel:
ODA-based modeling for document analysis
14 pages

TM-91-15
Stefan Bussmann: Prototypical Concept Formation
An Alternative Approach to Knowledge Representation
28 pages

TM-92-01
Lijuan Zhang:
Entwurf und Implementierung eines Compilers zur Transformation von Werkstückrepräsentationen
34 Seiten

TM-92-02
Achim Schupeta: Organizing Communication and Introspection in a Multi-Agent Blocksworld
32 pages

TM-92-03
Mona Singh
A Cognitiv Analysis of Event Structure
21 pages

TM-92-04
Jürgen Müller, Jörg Müller, Markus Pischel, Ralf Scheidhauer:
On the Representation of Temporal Knowledge
61 pages

TM-92-05
Franz Schmalhofer, Christoph Globig, Jörg Thoben
The refitting of plans by a human expert
10 pages

TM-92-06
Otto Kühn, Franz Schmalhofer: Hierarchical skeletal plan refinement: Task- and inference structures
14 pages

DFKI Documents

D-91-17
Andreas Becker:
Analyse der Planungsverfahren der KI im Hinblick auf ihre Eignung für die Abeitsplanung
86 Seiten

D-91-18
Thomas Reinartz: Definition von Problemklassen im Maschinenbau als eine Begriffsbildungsaufgabe
107 Seiten

D-91-19
Peter Wazinski: Objektlokalisation in graphischen Darstellungen
110 Seiten

D-92-01
Stefan Bussmann: Simulation Environment for Multi-Agent Worlds - Benutzeranleitung
50 Seiten

D-92-02
Wolfgang Maaß: Constraint-basierte Plazierung in multimodalen Dokumenten am Beispiel des Layout-Managers in WIP
111 Seiten

D-92-03
Wolfgan Maaß, Thomas Schiffmann, Dudung Soetopo, Winfried Graf: LAYLAB: Ein System zur automatischen Plazierung von Text-Bild-Kombinationen in multimodalen Dokumenten
41 Seiten

D-92-04
Judith Klein, Ludwig Dickmann: DiTo-Datenbank - Datendokumentation zu Verbrektion und Koordination
55 Seiten

D-92-06
Hans Werner Höper: Systematik zur Beschreibung von Werkstücken in der Terminologie der Featuresprache
392 Seiten

D-92-07
Susanne Biundo, Franz Schmalhofer (Eds.):
Proceedings of the DFKI Workshop on Planning
65 pages

D-92-08
Jochen Heinsohn, Bernhard Hollunder (Eds.): DFKI Workshop on Taxonomic Reasoning Proceedings
56 pages

D-92-09
Gernod P. Laufkötter: Implementierungsmöglichkeiten der integrativen Wissensakquisitionsmethode des ARC-TEC-Projektes
86 Seiten

D-92-10
Jakob Mauss: Ein heuristisch gesteuerter Chart-Parser für attributierte Graph-Grammatiken
87 Seiten

D-92-11
Kerstin Becker: Möglichkeiten der Wissensmodellierung für technische Diagnose-Expertensysteme
92 Seiten

D-92-12
Otto Kühn, Franz Schmalhofer, Gabriele Schmidt:
Integrated Knowledge Acquisition for Lathe Production Planning: a Picture Gallery (Integrierte Wissensakquisition zur Fertigungsplanung für Drehteile: eine Bildergalerie)
27 pages

D-92-13
Holger Peine: An Investigation of the Applicability of Terminological Reasoning to Application-Independent Software-Analysis
55 pages

D-92-15
DFKI Wissenschaftlich-Technischer Jahresbericht 1991
130 Seiten

D-92-16
Judith Engelkamp (Hrsg.): Verzeichnis von Softwarekomponenten für natürlichsprachliche Systeme
189 Seiten

D-92-17
Elisabeth André, Robin Cohen, Winfried Graf, Bob Kass, Cécile Paris, Wolfgang Wahlster (Eds.):
UM92: Third International Workshop on User Modeling, Proceedings
254 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-92-18
Klaus Becker: Verfahren der automatisierten Diagnose technischer Systeme
109 Seiten

D-92-19
Stefan Dittrich, Rainer Hoch: Automatische, Deskriptor-basierte Unterstützung der Dokumentanalyse zur Fokussierung und Klassifizierung von Geschäftsbriefen
107 Seiten

D-92-21
Anne Schauder: Incremental Syntactic Generation of Natural Language with Tree Adjoining Grammars
57 pages

A Multi-Agent Approach towards Group Scheduling

Andreas Lux

RR-92-41
Research Report