Procura:
A Project Management Model of Concurrent Planning and Design

Sigrid Goldmann
Universitaet Kaiserslautern
in cooperation with the
Center for Design Research,
Stanford University
goldmann@informatik.uni-kl.de

Abstract

Procura is a Project Management model which allows the planning and scheduling of agent based design projects in a hierarchical top-down approach. Since the information which is available while the top-level planning and scheduling is done is usually incomplete, the decisions made at this level will often need to be revised as soon as more information becomes available, and conditions change. Procura facilitates the necessary replanning, making use of Redux' [Petrie 93], a design support tool which can keep track of decisions, rejection of decisions, and their rationales. In the Procura model, Redux' tracks planning and scheduling decisions as well as the actual design decisions, thereby integrating planning, scheduling and design, and supporting a notification system which keeps all involved agents (designers as well as scheduling agents) informed about occurring changes.

1 Background and Motivation

In the past, there have been two different approaches to planning and scheduling: The *Operations Research (OR) approach* provides tools for graphical planning and scheduling support. CPM and PERT [Obrian84], [Muth63] are the two major planning techniques that have been developed in the course of the OR research. The *Artificial Intelligence (AI) approach* of *automatic* planning and scheduling additionally allows a tool to plan a problem by the automatic application of predefined operators. NONLIN [Tate77] and DEVISER [Vere83] are examples for such automatic planners.

The problem of automatic planners, though, is the necessity of predefined operators, which obviously can only be provided in a fairly simple context. Neither approach supports the user with replanning necessary for practical applications, even though project plans are usually invalidated soon after the begin of the project. Procura supports the user with planning and *replanning*, as well as scheduling and rescheduling, without trying to build the complex planning operators into the system. This approach will allow us to extend the system by an automatic planner where the domain allows for one, but Procura can also be applied in complex domains where it is not feasible to predefine all possible operators. Additionally, in Procura planning (i.e. the division of tasks into subtasks), scheduling (the assignment of resources and a start and end time to tasks) and plan execution (the assignment of values to design variables) are interleaved: The part of the plan that has not been executed yet is automatically updated whenever the actual execution of a subtask leads to different results than expected. In other words (since Procura has mainly been developed to support design) the plan and schedule are *incrementally* revised while the design is going on and is gradually getting more detailed: Planning, scheduling and design take place concurrently.

The Procura model is part of the Process-Link system, which has been developed as a framework for the distributed application of generic engineering services. Process-Link is an agent-based system which consists of several components. It allows a set of application agents to make design decisions and keeps track of decision rationales and rationales for the rejection of decisions, applying the Redux model of revisable design [Petrie91]. In case of an opportunity of improving the current solution of a design problem, the user is notified.[1]

To summarize, Procura is intended to solve the fol-

1. For an example scenario, see [Petrie95].

lowing problems:
1) Most “real world” domains do not lend themselves easily to a strict formalization in form of predefined planning operators.
2) Replanning and re-scheduling is often necessary, and needs to be supported by a planning tool.
3) Planning, scheduling and plan execution/design depend on each other and have to be interleaved. Scheduling can never be more than an estimation of the duration and resources needed to complete a task, and the execution results have to be compared to the schedule in order to incrementally update it, if necessary.

Our approach works with two basic principles: Scheduling is *prediction*, and has to be verified (and, if necessary, updated) by comparing the schedule with the actual facts, and a *model of design planning and scheduling* is needed in order to allow the necessary comparison between the actual design state and the schedule, and make change notification possible. The Procura model extends the Redux model of design revision to build a planning and scheduling model. It does not attempt to automate the planning process, but provides a bookkeeping service which supports the interleaved planning and execution, and facilitates necessary replanning. Rescheduling can partly done automatically: The original decision about who is going to do what task, and when, is made by the human user, but the propagation of changes through the schedule can be automated.

2 Working Model

In Redux’, a *goal* is an aim to be achieved by making a *decision* for it. The *decision* can be to divide the goal into subgoals, or make variable assignments, or both.

Procura uses the Redux’ model and extends it. It allows the user to specify the goals to be achieved in a specific project. For example, the project might be the invention of a new motorcycle, with the main goal “Develop Motorcycle”. A *task* is a chore to be done to achieve a goal. In our system, it is represented as a type of goal with a *duration* that can be specified by the user. Tasks produce *outputs* which serve as *inputs* for other tasks. These inputs and outputs are *predicted*, and have to be verified by *comparing* them to the actual inputs and outputs which are determined by making a decision for the task. Inputs and outputs are passed in form of parameters or features, which can represent data as well as physical objects.
Besides tasks, there is another type of goal which can be handled in our system: A *scheduling goal* corresponds to a task to be solved by *scheduling*[1] this task.
Figure 1 states the relationship between goals, tasks and scheduling goals.

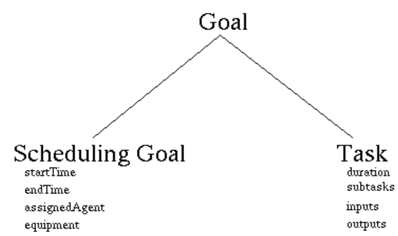

Figure 1

By determining which of the tasks’ inputs have to be in place before work on it can be started, and by finding out which tasks produce the required parameters or features, the precedence relationship between the tasks is determined, and a *plan* can be stated. A *plan* is a partially ordered sequence of tasks; the order is enforced by the tasks’ input/output relationships.
Tasks are done by *agents*, using shared *resources*. In this initial version of Procura, only nonconsumable shared resources are considered,[2] namely *workers* and *equipment*. An *agent* is a human or computer using an interface to Procura, which provides a sophisticated notification system and keeps the agent up to date on all changes in the plan, schedule, and design, by which the agent is concerned. A *worker* is a person who does not have access to Procura and therefore does not get any messages concerning the state of the plan. *Equipment* can be a tool or machine. Agents and (nonconsumable) resources need to be *scheduled* to do the various tasks of the plan, by assigning them to tasks for a specified time interval. Each task takes up a certain percentage of an agent’s or a resource’s time. The sum of all the tasks scheduled for an agent (or resource) at a time is its *utilization*. At times when an agent or resource is *available*, its utilization will vary between 0% and 100%. The utilization of an agent or resource that is unavailable (e.g. because of vacation or necessary repairs) must be 0%. By assigning an agent and the needed resources to each of the tasks of the plan, the timing of each task is determined and the *schedule* is created. The created schedule is similar to a CPM plan; it states the critical path, and for each task its earliest and latest start and end time.

Procura allows a plan to be hierarchical, i.e. a task can have subtasks, which form subplans and sub-schedules,

1. See definition below.
2. In a later version, we will also consider shared consumable resources, and let the system keep track of the consumption of those resources in a similar way to the trackkeeping of the utilization of nonconsumable resources which is done in this first version of Procura.

so that the overall plan can be divided into several levels of detail. A task which has subtasks is called a *complex* task.

The planning is done top-down: First, the timing of the top-level task is determined, then it is divided into subtasks, and further planning and scheduling takes place.[1]

As mentioned earlier, each task has a corresponding scheduling goal. The task and its scheduling goal are solved by different decisions:
- For the scheduling goal, a *scheduling decision* is made. This kind of decision assigns an agent and equipment and workers to the corresponding task, and determines the task's *timing*: its start and end time. (Figure 4 shows an example for a schedule, created by scheduling decisions for each of the tasks T2.1 - T2.5)
- The task has to be either solved by making assignments for the specified output parameters and features in a *doing decision*, or further divided into subtasks with the help of a *planning decision*. (In Figure 3, a task hierarchy is shown after two planning decisions have been made for T0, dividing it into T1 - T4, and T2, dividing it into T2.1 - T2.5)

As the planning and scheduling proceeds further down in the tree, more detailed information about the duration, timing, inputs, outputs and required resources becomes available, and the planning and scheduling decisions made at higher levels will have to be corrected, the overall schedule updated.
Procura allows agents to add new inputs to tasks assigned to them, or refuse[2] inputs that have been stated to be necessary. In this case, the (predicted) plan and schedule have to be corrected (adapted to the actual requirements of the "real world") accordingly, and all concerned agents will be notified. The same is true for the execution of leaf nodes, i.e. doing decisions: once a task has been done by making assignments for its output parameters and features, its actual timing, inputs, outputs and utilized resources become known. This has to be noted, and the plan accordingly updated.

As the planning and plan execution proceed, *conflicts* between decisions might occur. For example, an agent might be scheduled to utilize more than 100% of its time, or two agents make variable assignments which are mutually exclusive. The agent who detects the conflict makes it known to the system, which then notifies all concerned agents. These agents will then find a solution for the conflict (using Redux' as a Dependency Directed Backtracking utility, see [Petrie95]), and reject one or more of the decisions involved in the conflict. The conflict is stated as the justification for the rejection, so that the user will be notified if at a later point in time the conflict is resolved by some other means and it is possible to return to the originally rejected decision.
The system also keeps track of already *committed* tasks. There are three levels of *commitment*:
1) A task has been done by making assignments to its output parameters and features, and time has been consumed. The decision can be rejected by simply invalidating the assignments, without further time and resource expenditure. For example, the design decision to use engine-A for a newly developed motorcycle could simply be overwritten by a new decision to use engine-B, but in the course of making the original decision for engine-A, time has been spent which cannot be regained.
2) A task has been done by making changes to the real world (represented in Redux' as parameter and feature assignments). The decision can be rejected, but in order to get back to the original state of the world, additional time and resources have to be spent. An example for this level of commitment would be to make the decision for engine-B after the machines for constructing engine-A have already been installed. It would take time and resources to get rid of the old machines before the new ones could be installed.
3) By performing a task, the real world has been changed in a way that cannot be undone. This level of commitment might occur if there is only one object of a certain kind which has been destroyed or changed in a way that is irreversible.
The level of commitment has to be determined and stated by the Application Agent. Redux' will send a warning whenever a user attempts to reject a decision that would lead to the rejection of a committed decision.

3 Architecture

The Project-Link system consists of four parts, each of which are implemented as one or more independent agents:
1) The Project Manager (PM)
2) The Agent Manager (AM)
3) Redux'
4) The Constraint Manager (CM)[1]

1. The top-down approach is not mandatory, though. It is also possible to introduce a low-level task, plan it first, and associate it to a high level task later in the planning process.
2. "Refusing" an input for a task means stating that the refused parameter is not needed to solve the task. It might also happen that an agent finds that it cannot work with an actual value of an input. This problem has to be dealt with "manually", i.e. outside the system.

In this system, a set of Application Agents is “plugged in”.

The PM and AM components together implement the Procura extensions described here. Figure 2 shows the system architecture. The arrows between the components represent the message flow between the agents.

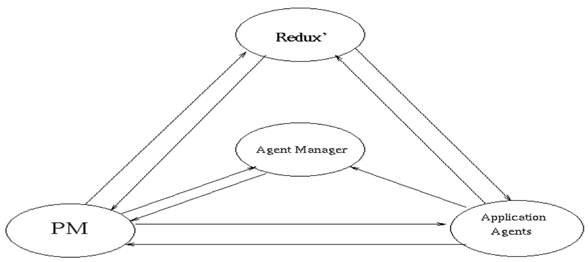

Figure 2

The Project Manager (PM)[1]

The PM is responsible for constructing the plan and schedule according to its hierarchical structure and its tasks' input/output relationships. The PM has a user interface that can show the plan and schedule. It can either show a whole level of the plan or schedule, or it can present a sub-plan or sub-schedule consisting of all the subtasks of a certain task.

The scheduling decisions for all tasks are made by the PM. To do this, the PM requests information about the availability of agents and equipment, as well as their capabilities, from the AM. The PM then constructs the schedule and calculates its critical path, and for each task, its earliest and latest start and end time.

As the application agents request or refuse inputs to the tasks they are working on, the PM has to update the plan and schedule and inform all concerned agents. It is also responsible for making necessary changes (and notify Redux') as the plan gets more detailed and the higher levels of the plan have to be updated.

During the execution of the plan, the PM monitors the start and end times of all tasks, and sends warnings to the concerned agents (the agent assigned to the task and the agent responsible for its supertask), if a task's latest start time or end time has been violated.

1. In the current version of the tool, Procura does not make use of the CM agent. In a future version, though, we are planning, to use constraint propagation for partly automating the scheduling.

1. In the current version of the model, the PM agent is a combination of a human (who makes the scheduling decisions) and a tool (which provides communicates with other agents and provides the information needed to make scheduling decisions).

The Agent Manager (AM)[2]

The AM controls the utilization and availability of all agents and (non-consumable) resources. For each agent, worker or piece of equipment, it provides a schedule which states its utilization and availability at each point in time. When an application agent comes on-line, it will send a message to the AM, stating its name, location, and a description of its skills and capabilities. On request, the AM makes this information available to the PM or other agents. Should an agent or resource become unexpectedly unavailable (e.g. fall ill), the AM will inform the PM, which then will do re-scheduling and inform all concerned agents. The same holds, if an agent which is supposed to be available is not on-line at a certain point in time.

Redux'

By keeping track of the decisions that have been made so far, and in determining the consequences of proposed changes to the plan and schedule, Redux' assists the user with the frequently required replanning. Redux' provides the notification mechanism for conflicts, decision rejections and necessary re-evaluation of rejected decisions. When a decision is rejected, all other decision which are dependent of the rejected decision, are invalidated.

Redux' sends notifications to the concerned application agents when

- a new task assignment occurs,
- the input for one of an agent's tasks becomes available or changes,
- a task or task assignment is invalidated, or
- the timing for a task has been changed.

Redux' also keeps track of committed tasks, and sends a warning if the user tries to reject a decision that would lead to the invalidation of an already committed task.

The Application Agents

The application agents are responsible for solving the plan's task(s). In the solution process, they make planning decisions, which divide tasks into subtasks, and doing decisions, which make assignments to output parameters and features. They also state the level of

2. The tracking of agents' and equipment's schedules could also have been done in the PM component. We decided to invent the AM component in order to allow for a future extension of the system that will enable multiple PM agents to manage the same pool of agents and equipment.

commitment of a decision. For each of its decisions, an application agent should, on request of the PM, state the suggested inputs and outputs of the created subtasks. It should also be able to state the agent skills and equipment properties which are required for solving the task, and estimate the number of hours the task will take up of the time of an agent with the stated skill, or a piece of equipment with the required properties. This information will enable the PM to make a scheduling decision for the task.

Messages between the Components

To organize the information exchange between the components, a message protocol has been defined. Other than exchanging these predefined messages, the components do not communicate with each other, and no component can call any functions in any of the other components. Therefore, our approach is truly agent-based, and the involved agents can easily be exchanged.

4 Example

Planning Tasks

We will now illustrate our approach on the motorcycle example we have already mentioned.
The overall task of the example project is T0: “Develop Motorcycle”. Let us assume that several decisions have already been made: A planning decision for T0 has been made, creating the subtasks T1: “Specify Requirements”, T2: “Design Motorcycle”, T3: “Construct Motorcycle”, and T4: “Testing”. T2 has also been divided into subtasks, namely T2.1: “Design Engine & Frame”, T2.2: “Design Carburettor”, T2.3: “Design Suspension”, T2.4: “Design Wheels & Brakes”, and T2.5; “Design Fairing”. Figure 3 shows the task hierarchy tree created by these planning decisions.

These two planning decisions have been made by application agents, and Redux’ has been notified of the decisions.

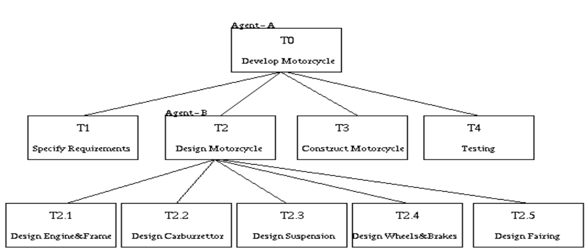

Figure 3

Let us say that T0 has been planned by Agent-A, and the planning decision for T2 has been made by Agent-B, as stated in Figure 3. The two agents might have previously been assigned to the tasks by the PM agent, or they just made their decisions in the Process-Link system without the participation of the scheduling functionality of Procura.
If now the task hierarchy is presented to Procura’s PM agent, it will need additional information to schedule the tasks: For all the created subtasks, it will request the following information from the agents which made the planning decisions: the inputs, outputs, suggested skills and properties of the responsible agent and the needed equipment, and the amount of time the agent and equipment will be needed for. I.e., Agent-A will be requested to state the required information for T0’s subtasks T1-T4, and Agent-B will be asked to provide additional information on T2’s subtasks T2.1-T2.5.

For example, a message from the PM to Agent-B will look like this:

RequestSchedulingInfo(T2.1)

In reply, Agent-B will send the following message:

SchedulingInfo(`task` T2.1; `duration` 40 agent-hours; `input` requrements_document; `output` engine_design, frame_design; `agent-skill` “Design”`,` `equipment-property` NIL)

This message states that an agent with the skill “Design” will have to spend 40 hours on the task T1, which requires the input “requirements_document, and produces the output parameters “engine_design” and “frame_design”. No workers or equipment are needed.[1]
As soon as the PM knows the scheduling information for all the subtasks of T2, it can proceed to state the sub-plan of T2 by ordering the subtasks according to their input/output relationships. It can also generate a schedule by selecting an agent, assigning the required workers and equipment to each subtask, and determining the order of the subtasks which have to be done in series, because of a capacity limit of an agent of piece of equipment assigned to them. Once the schedule is established, the PM can proceed to compute the critical path, and the earliest and latest start and end times.

Assigning Agents

The PM then sends one message per subtask to Redux’, stating that the task has been scheduled. For example, the message concerning the task corresponding to T2.1 might look like this:

CreateSchedulingDecision(`task` T2.1;
`assigned-agent` Agent-B;

1. As stated before, this scheduling information is *predicted*, and might have to be corrected later, according to the principle of comparing actual state to plan.

equipment NIL; startTime day-7;
endTime day-14).

As a reaction to this message, Redux' will inform Agent-B of the newly assigned task with the following message:

TaskAssigned(task T2.1; startTime day-7;
endTime day-14).

When all the scheduling decisions for the subtasks of T2 have been made, the PM can construct the schedule and calculate the critical path. Figure 4 shows a possible schedule with its earliest and latest start and end times; the thick line marks the critical path.

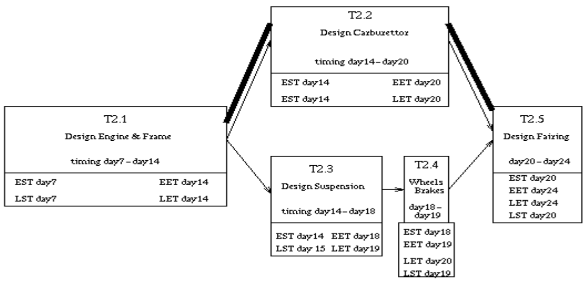

Figure 4

To select the agents and equipment to assign to a task, the PM has to find out which agents with the required skills are available at the time the task has to be scheduled for. For example, to find out the availability of agents capable of working on T2.1, the PM will send the following message to the AM:

RequestAgentUtilization(skill "Design";
start day-4; end day-15)[1].

In reply, the AM will send a message, stating that Agent-A and Agent-B have the skill "Design", and showing the utilization tables for the two agents between the dates "start" and "end". The PM can then determine which agent can work on the task, and assign the task to the agent, stating the timing it has determined with the help of the agent's utilization table. It will then inform the AM that the agent has been assigned a new task, and how much of the agent's time the task will utilize, so that the AM can update the agent's utilization table.

Redux' allows an agent to refuse to work on a task assigned to it. If that happens, the task assignment will be invalidated, and the PM will be required to make a new scheduling decision. It will also notify the AM of the change in the concerned agent's utilization.

1. The time frame for which to request the agent's utilization can be determined in several ways. In our case, it is simply the (already determined) timing of the supertask.

Revision

Now suppose that the AM is notified that Agent-B will not be available between day-7 and day-14. The AM will notify the PM that Agent-B will be unavailable to work on T2.1,[2] and the PM will then reject the decision for T2.1:

RejectSchedulingDecision(T2.1)

After finding out the utilization schedule for the other possible agent Agent-C, the PM can make a new decision for T2.1:

CreateSchedulingDecision(task T2.1; agent
Agent-C; equipment NIL; startTime day-8;
endTime day-15)

Note that the timing for T2.1 has changed (possibly because Agent-C is not available on day-7). This will necessitate a change in the scheduling decisions for all tasks which get their inputs from T2.1. A new schedule has to be constructed and it has to be calculated if the critical path has been violated. If so, the agents responsible for the super-tasks of T2.1 have to be notified by the PM. Should Agent-B become available again, the AM will send a message to Redux', which will then notify the PM that it might go back to the old decision which assigned Agent-B to T2.1.[3] The PM will then be able to decide if it wants to go back to the earlier decision, which is supposedly the better decision, or if it will stay with the new decision because it would be too much effort to change the schedule again (e.g. if T2.1 or one of its dependent tasks have already been committed).

If now Agent-C makes a doing decision for T2.1, the task will be committed at level 1. The doing decision is sent to Redux' in the following message:

CreateDoingDecision(task T2.1; inputs I1;
assignments O1 = x, O2 = y; commitment 1)

Redux' will notify the PM of the decision, which will then find out the actual start and end time of the task, correct the schedule if the actual facts differ from the planned data, and mark the task as "done". Should Agent-C decide to reject and remake the decision, the PM will know that there has already been time spent on the task, so that it will not remove the committed task from the schedule, but rather insert a new copy of the task.[4] Figure 5 shows the schedule after such a copy of T2.1 (with a duration of 2 days) has been inserted. Note that in comparison to the schedule shown in Figure 4,

2. To be able to do this, the PM will look up the schedule of Agent-B, and inform the PM of all the tasks that have been assigned for the agent to be done during the time in question.

3. See [Petrie95] for details.

the timing of the tasks succeeding T2.1 have been changed in order to allow for the additional time spent on T2.1'.

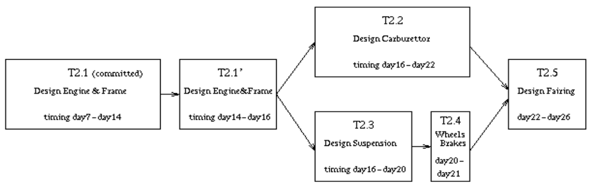

Figure 5

5 Related Work and Conclusions

As mentioned before, the Procura model is based on the Redux model of revisable design, which can be categorized as a conflict management model. Other related research areas are workflow management, and traditional project management approaches. One example of a workflow management system is the CoMo-Kit tool [Dellen96], [Maurer96], which, like Procura, integrates planning and plan execution, but does not address the issue of scheduling. Procura's novelty in contrast to traditional project management tools are the *integration* of planning, scheduling and plan execution, the support for *incremental revision* of the plan and schedule as well as the artifact design, and the *notification* system which keeps the concerned agents up to date on any changes, while agents who are not affected by a particular change do not get a notification. Also, traditional systems work on a client-server basis, in contrast to the *agent-based* approach of Procura. The predefined message set that has been defined for the communication between the agents allows any application agents to be plugged into the system by simply defining a software wrapper for the agent which implements the predefined message protocol.

The Procura model (i.e. the PM and AM components) is currently being implemented and will be tested with an example scenario similar to the example introduced in this paper.

The PM is designed to leave the scheduling decisions to a human user. Future work on Procura will include automating the scheduling with the help of the CM agent. Another future extension of the tool will be the implementation of a notion of hierarchical control and authority, by introducing a notion of the organizational structure between the agents involved in the planning and execution process, and limiting the authority of individual agents to plan and/or do specific tasks.

In the current version of the Procura model, the Redux' agent acts as a central bookkeeping mechanism which has to track all the decisions made inside the system. The possibility to decentralize the system by splitting up the bookkeeping responsibilities between several Redux' agents is currently being considered.

4. This is necessary, because T2.1 has already been executed, and time and resources have been consumed (i.e. the task has been committed). These effects cannot be undone or ignored, even if the decision for the task is rejected, but have to be stated in the schedule.

Acknowledgments

I would like to thank Charles Petrie for all the constructive discussions about the Procura model, and for many helpful suggestions concerning the style of this paper.

Literature

[Dellen96] B. Dellen, F. Maurer, "Integrating Process Planning and Execution", WETICE '96, Stanford, USA, JUne 1996.

[Maurer96] F. Maurer, "Project Coordination in Design Processes", WETICE '96, Stanford, USA, June 1996.

[Muth63] J. F. Muth and G. L. Thompson, "Industrial Scheduling", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, USA, 1963.

[Obrien84] J. J. O'Brien, "CPM in Construction Management", 3. Edition, McGraw-Hill, Inc., USA, 1984.

[Petrie91] C.J. Petrie, "Constrained Decision Revision", PROC AAAI-92. Also MCC TR. EID-414-91, December, 1991.

[Petrie93] C. J. Petrie, "The Redux' Server", Proc. Internat. Conf. on Intelligent and Cooperative Information Systems (ICICIS), Rotterdam, 1993.

[Petrie95] C. J. Petrie, Teresa A. Webster, Mark R. Cutkosky, "Using Pareto Optimality to Coordinate Distributed Agents", AIEDAM special issue on conflict management Vol. 9, pp 269-281, 1995.

[Tate77] A. Tate, "Generating Project Networks", IJCAI-77, Boston, MA, USA, 1977.

[Vere83] S. A. Vere, "Planning in Time: Windows and Durations for Activities and Goals", IEEE Trans. on Pattern Analysis and Machine Intelligence, PAMI-S, No. 3, pp. 246-267, 1983