

PROPLANT: MULTI-AGENT SYSTEM FOR PRODUCTION PLANNING

Vladimír Mařík, Michal Pěchouček, Olga Štěpánková, Jiří Lažanský

e-mail: {marik, pechouc, step, lazan}@labe.felk.cvut.cz

The Gerstner Laboratory for Intelligent Decision Making and Control
Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University,
Technická 2, CZ 166 27 Prague 6, Czech Republic

Abstract

The so-called “tri-base” acquaintance model of the agent’s behaviour is presented in this paper. This represents an extension of the twin-base model [Cao 97]. Based on practical experience, the new model tries to cope with parallel processing, precedence constraints, and sparse resources. The idea of substituting the inter-agent negotiation processes by the periodical internal planning activity of the agents is stressed. A multi-agent system *ProPlanT* as an application of the tri-base model for the project-oriented production planning developed for TESLA TV company is described in detail. Three types of agents (PPA – Production Planning Agent, PMA – Production Management Agent and PA – Production Agent) are distinguished. The corresponding tri-base models and potential role of meta-agents are discussed.

1. Introduction and Motivation

The growing complexity of problems to be solved in industry requires new software system architectures integrating the already existing “islands” of well-working algorithms into more robust global systems. Currently ill-suited information flows, lack of communication among different production units and far from full utilisation of available information processing tools is what makes all of the production processes difficult to understand, model, plan and consequently optimise. Attempts to create a global, monolithic software solution, no matter how well hierarchically structured, fail.

We have been invited to participate in design of a production-planning system for the TESLA-TV factory, a Czech manufacturing enterprise delivering TV and FM transmitters, within the framework of EUREKA No. 1439 project. The manufacturing process can be classified as a project-oriented production. Consequently, rather than planning and simulation of a flow of semi-products to be assembled, there is a need for a solution which would facilitate easy planning and subsequent optimisation of the discrete manufacturing of the unique final product.

After thorough analysis of the plant it became obvious that one has to suggest a method how to integrate partial software solutions (“legacy” software systems). On the other hand, flexibility and frequent changes in the production facility inevitably require a highly distributed solution. The results of the analysis strongly supported the choice of a multi-agent system as a suitable technology for building *ProPlanT* (*Production Planning Tool*) system [Mařík 98].

There is a wide range of possible multi-agent systems’ architectures. The mass of identical, conform agents and a highly granular community of deeply specialised agents are the extreme cases in this spectrum. Whereas in the case of a large community of identical agents (*collective architecture*) the “social” and “sociological” aspects of behaviour play a dominant role, in the latter case the

community is expected to be strongly internally organised and co-ordinated. The latter architectures (*integration architectures*) seem suitable for (legacy) system integration purposes, for designing and implementing highly distributed modelling, decision making and control algorithms.

The integration architectures usually involve a small number of well-developed agents with a clearly defined, highly specialised functionality. Each agent, in our understanding, consists of a *functional body* (usually a stand-alone, already well running program with a precisely specified functionality) and a *wrapper* which is responsible for agent's engagement in the agents' community. The wrapper contains a *model of the agent's behaviour* [Nwana 97]. The wrapper also "translates" the inter-agent communication into the instructions for the activity of the functional body and mediates the results of the body activity into the agents' community. Nevertheless, the integration architectures can also consider some sort of the community behaviour.

Various kinds of models of agent's behaviour exploring different communication strategies can be found [Wooldridge 95], [Lashkari 94], [Wittig 92]. Some are based on *broadcasting of task-announcement messages* or on the presence of a *central communication agent*. Another solution is offered by so-called *acquaintance models* [Jennings 93] that create a wide group of behavioural models. Here, information on the capabilities and behaviour of the given agent as well as the relevant pieces of information about the other agents is stored in the wrappers of individual agents. Thus, the information is not concentrated in a central agent but kept highly distributed (while well organised and co-ordinated) across the community. The acquaintance models help to decrease necessary communication when an agent is searching for the best-qualified agent(s) for co-operation.

For system integration purposes in industrial tasks, each existing software system/module can be understood as a body of one agent. Whenever appropriate methods for communication and co-operation among the agents are defined and corresponding techniques are implemented inside the wrappers, a highly integrated solution can be achieved.

2. Twin-base Acquaintance Model

A specific acquaintance model technique, called a twin-base approach was proposed recently [Cao 96]. This approach suggests organising the relevant information about co-operating agents into two separate information bases in the wrapper, namely into

- CO-OPERATOR BASE, which has an auxiliary nature and which contains/collects information concerning the other agents. It can provide information on the data and message formats used by the other agents and their addresses. Moreover, it contains information on current capabilities of peers of the agent, e.g. statistics concerning the other agents' efficiency, trust values etc.
- TASK BASE which contains relevant particular information on possible problem decomposition and task delegation solving process in the form of triples

$$\langle T_j, A_i, \bigcup_{k=1}^{m'} \langle T_{jk}, A_{jk} \rangle \rangle,$$

where T_j denotes the task, A_i represents the co-ordinating agent and the third member of the triple is the decomposition of the task T_j into m' subtasks fulfilled by the subcontracted helping agents.

An integral part of the twin-base model is a revision process aimed at keeping the contents of the bases fresh. This way, the task-base always contains up-to-date information on the current capabilities of the peers. This facilitates directing the co-operation requests to the most suitable agent in the community. Therefore, the communication traffic is significantly reduced and the system responses are very fast since the non-addressed task announcements are avoided.

3. Tri-Base Acquaintance Model

The efficiency of a good solution of any task depends on the *current* and *future* availability of agents within the community. The notion of *future* has to be regarded mainly when considering planning and scheduling tasks mentioned in the introductory remarks of this paper. In order to cope with this aspect, we have introduced a *tri-base model* as a powerful enhancement of the *twin-base model*. The novel idea is based on the strict separation of data specifying the current task from permanent and transient knowledge, which each agent maintains within its wrapper.

The agent can solve the task either *directly* by itself, or *indirectly* by taking responsibility for the task. In the latter case, the agent acts as a *co-ordinator* and contracts some others to help with the solution. In this situation we speak about the *SuperTask* that the agent co-ordinates.

In the tri-base approach, the original CO-OPERATOR BASE has been split into two separate bases: the CO-OPERATOR BASE and the STATE BASE. The CO-OPERATOR BASE keeps rather static information concerning the peers while the STATE BASE is updated very frequently reflecting the changes of the other agents' states and their current capabilities. The structure of the bases is as follows:

- The CO-OPERATOR BASE stores static information on the peer agents, such as their IP addresses, communication means and predefined responsibilities. Moreover, it specifies the set of *subscribers* – the agents that subscribed for reports on change of the considered agent's status.
- The TASK BASE contains knowledge concerning possible task decomposition with respect to problem solving processes. The *problem section* contains general knowledge on possible task decomposition and contingent time precedence and prerequisites (if any). There is a *plan section* that stores deduced plans on how to solve particular tasks through co-ordination of

subcontracted helping agents. These plans are maintained on-line during spare time of the agent to have them ready whenever their need arises.

- The STATE BASE reflects the peer agents' states that may evolve rapidly in time. The *agent section* of the STATE BASE reflects the internal states of the peer agents like their current load, attainability and trust, as well as capabilities (e.g. speed and price of processing) and schedule of the considered agent. The *task section* describes the current states of the solution of the tasks that have been contracted and are co-ordinated by the agent. The peers are expected to report the solution progress.

The multi-agent community may be in charge of a number of parallel problem-solving projects (tasks). This is why we have to distinguish among three different stages of the community operation with respect to a single project. The system may be engaged in *pre-planning*, *planning*, and *plan fixing*.

When engaged in *pre-planning*, the community creates possible variations of plans how to cope with a requirement for the task T and elaborates an estimation of the resources (time, cost) requirements for each option. In the phase of *planning* are plans re-evaluated throughout the time so that any relevant changes within the STATE BASE are reflected and thus the plans are maintained most up-to-date (may be ordered with respect to their suitability). When asked for plan fixing a task T , the agent tries to select the plan from its TASK BASE. Each agent thus takes the currently best available plan and decomposes the requirement.

The main motivation behind splitting the original CO-OPERATOR BASE into two bases is to maintain and administer separately (1) the information which is changing frequently (the frequency of changes corresponds to the rhythm of problem solving phases of co-operating agents) and of which is not very much (data about collaborating agents only) and (2) the semi-static knowledge about all members of the community, that does not change very often (if at all) and is of much bigger volume

here. As the process of planning is rather more complicated than in the twin-base model, the efficiency of retrieval the right data is critical.

The tri-base model, unlike twin-base, facilitates creating agent's scopes of reasoning where it is having information, it uses often or it is quite likely to use soon, pre-prepared and maintained separately. The information in the state-base concerns only agents, the particular agents collaborate with (is still extendible). In the CO-OPERATOR BASE there are not stored all possible courses of actions the particular agent is capable of, as it was in the twin-base. We rather keep generic rules how to cope with different tasks separate from its instantiations valid at the particular moment, that are computed throughout the planning process and stored in the STATE BASE.

4. Structure of Model Bases

In the following, we define in more detail the structure of bases located in the wrapper of an individual agent a .

4.1 Co-operator Base

All the permanent data about co-operating agents are located within this base. The CO-OPERATOR BASE is composed of two sections: the *collaborators section* and the *subscribers section*.

The *collaborators section* of each agent contains three main kinds of data: agent's address, message format, and predefined responsibility. Following quadruples are used:

$$\langle a, Address_a, Language_a, Set_of_Tasks_a \rangle,$$

where agent a is the symbolic name of a particular co-operating agent, $Address_a$ specifies its physical location such as IP address, port etc., $Language_a$ is its communication format, and the set $Set_of_Tasks_a$

specifies responsibilities of the agent a , i.e. the tasks the agent a is designed to be accountable for. The agent is accountable for a task whenever it is able to solve the task by itself or by further decomposition under its own co-ordination.

The complementary part of the agent's a CO-OPERATOR BASE is the *subscribers section*. It contains addresses of those agents, which have shown interest in changes of the state of the agent a by subscribing for this information. Whenever the subscribed agent a changes its state or its planning agenda, it *advertises* this change to all subscriber agents that are referenced in the *subscribers section* of a 's CO-OPERATOR BASE. The subscription mechanism supported by the *subscribers section* specifies the group of agents that are interested in the agent's activity.

4.2 State Base

An agent's STATE BASE does not have to contain information on the entire community. It reports on load and capabilities of those agents and on tasks that belong to the scope of reasoning of the agent a . The STATE BASE is divided into the *agent section* and the *task section*. This base comprises non-permanent information:

- *Agent section* provides data on the actual status of co-operating agents regarding their load and reliability. Quintuples as follows are proposed to be used for this purpose:

$$\langle a, Load_a, Trust_a, Capabilities_a, Schedule_a \rangle,$$

where a refers the particular agent, $Load_a$ specifies its load, and $Trust_a$ is the level of confidence of the agent a into such information. Agent load can be described using various measures: It can be simply a flag with values of engaged/vacant, a number of requests per time unit, or the agent's agenda, specifying the nearest available time-slot in future. The level of confidence – $Trust_a$ is derived from information on how old the triple is, and/or how frequently does it usually change. The $Capabilities_a$ characterise current resources of agent a that can dynamically change. $Schedule_a$

specifies slots, which have been already assigned for a certain type of activity of the agent, and it covers a reasonable time horizon.

- The *task section* maintains knowledge related to the current state of the solution process. Each agent stores information on tasks in the solution of which it is involved. The *task section* of the STATE BASE is composed of the following quadruples:

$$\langle Task, Agent, Solution, Trust \rangle$$

Solution describes the state of the *Task* solved directly or indirectly by *Agent*. There are three distinct solution states, which require different *Solution* descriptions. They are

- (i) *Task* has not started, yet;
- (ii) *Task* is being processed;
- (iii) *Task* has been finished.

In the first two cases, there must be some indication when the task can start or when it is expected to be completed. In the third case, the *Solution* description should contain the description of solution quality, duration, and the solution results.

The maintenance of such information is easy if the *Agent* solves the task *directly* by itself. If, however, the *Agent* is a co-ordinator for the *SuperTask*, it has to ask its subcontracted agents to report the solution progress whenever any of them changes its state. This is implemented through *subscribing* the subcontracted agents for this information. We consider subscription as a particular kind of co-ordination.

4.3 Task Base

This structure contains knowledge on possible decomposition of tasks considered. The decomposition can be used to devise various plans how to fulfil a requested task either by means of

actions of the agent's body, or by contracting appropriate agents with a request for help. The TASK BASE must also prevent task request looping, when agent *a* asks agent *B* for help and *B* delegates the solution of the same task back to *a*. Creation of plans has to be based on fresh information on capabilities of co-operating agents with respect to the relevant goal. As mentioned above, the TASK BASE is divided into its *problem* and *plan sections*.

The *problem section* of the TASK BASE comprises information on possible decomposition of the tasks. Triples as follows are used within this sub-base:

$$\langle SuperTask, \{T_j\}, Precedence, Conditions \rangle,$$

This triple claims: “provided *Conditions* are met, the *SuperTask* can be successfully accomplished by performing all the tasks from the set $\{T_j\}$ in such a way that all the time *Precedence* constraints are honoured”. *Conditions* specify applicability conditions for the given decomposition. Two separate parts of any given set of conditions have to be distinguished: One is independent of the actual state of participating agents (for instance, requirements on maximal number of agents engaged in the problem solving or on total length of *SuperTask's* processing); the other refers to actual state of the relevant agents at the moment of processing their assigned subtask of the *SuperTask* (e.g. size of available free memory).

If the set *Precedence* is empty (no precedence constraints), then all the subtasks of the considered decomposition $\{T_j\}$ can be processed in parallel.

If the agent solves the task by itself, the triple looks as follows

$$\langle SuperTask, \emptyset, Conditions \rangle.$$

The *plan section* comprises production rules representing plans how a given task can be accomplished under actual configuration of the distributed community. Quadruples as an extension of twin-base triples have been proposed within the philosophy of the tri-base model:

$$\langle \textit{SuperTask}, \langle \{ \textit{Agent}_j, T_j \}, \textit{Precedence} \rangle, \textit{Constraints}, \textit{Trust} \rangle$$

Constraints appearing in the decomposed *SuperTask* plan are derived from the original *Conditions*. *Constraints* represent that part of the *Conditions* that is related to the actual state of sub-contracted agents (e.g., the available computational power of those agents). The actual mapping from *Conditions* to *Constraints* is task-domain specific and depends on the nature of the *Conditions*. *Conditions* cannot be checked in advance, as the states of the participating agents change in time. If the *Constraints* hold, the agent *a* plans to ask *Agent_j* to carry out the sub-task *T_j* of the *SuperTask*. This way, agent *a* chooses an appropriate *Agent_j* for each of the subtasks *T_j*. The last argument stands for the *Trust* of the agent *a* in successful fulfilling of the *SuperTask* using the suggested decomposition.

Whereas the pre-planning phase stores a number of various plans in the *plan section* of the TASK BASE, the plan fixing phase results in selection of the currently best plan and storing it in the *task section* of agents' STATE BASE. The overall structure of the bases within the agent's *a* wrapper is sketched in Figure 1.

STATE BASE	CO-OPERATOR BASE	TASK BASE
<p>Task section</p> <p>Review of tasks assigned to agent \mathcal{A} for direct solution or co-ordination</p> <p>$\langle Task, Agent, Solution, Trust \rangle$</p>	<p>Collaborators</p> <p>(Semi-)permanent data about agents \mathcal{A} collaborates with</p> <p>$\langle Peer_agent, Address, Language, Set_of_Tasks \rangle$</p>	<p>Problem section</p> <p>Description which agent \mathcal{A} can use to handle tasks it can solve by itself directly or it can co-ordinate for indirect solution</p> <p>$\langle SuperTask, \{T_j\}, Precedence, Conditions \rangle$</p>
<p>Agent section</p> <p>Up-to-date data on agents co-operating with \mathcal{A} including their current capabilities, load and schedule for reasonable time horizon</p> <p>$\langle Agent, Load, Trust, Capabilities, Schedule \rangle$</p>	<p>Subscribers</p> <p>Addresses of agents to whom agent \mathcal{A} should report its state changes</p> <p>$\langle \{Agent\} \rangle$</p>	<p>Plan section</p> <p>Up-to-date suggestions of plans how to achieve tasks from problem section under current state of the whole system</p> <p>$\langle SuperTask, \{Agent, T_j\}, Precedence, Constraints, Trust \rangle$</p>

Figure 1

5. The Process of Model Updating

Let us review how to maintain the contents of the bases. One option is to use a special agent called *facilitator* [Cao 96, 97] devoted to maintain the content of the agents' bases by *periodical revisions*. In idle times, facilitator keeps checking agents' load, using the pre-defined information in the CO-OPERATOR BASEs, and computes the most up-to-date information of agents' current capabilities. This information is represented in the form of trust in plans to be used by the agent, so that the best suitable plan is preferred.

This approach loses its transparency as soon as we want to deal with parallel processing and sparse resources. That is why we search for another solution based on more active role of each agent in the community. Either an agent can find out all relevant information itself, or it can get it from the

contracted and subscribed agents. The first method is considered expensive in terms of communication, as an agent would have to ask the collaborator every time it would go for revision. This revision must be done periodically whereas the subscribing approach makes the agent to revise its information only if necessary. That is why we prefer the latter possibility. This approach tries to avoid the need to rely on a kind of central communication agent, the *facilitator*, which is in charge of updating agents' plans. The *plan section* of the TASK BASE re-computation was shifted to the agent itself. Each agent uses its idle time so that it combines its knowledge in the CO-OPERATOR BASE, the STATE BASE and permanent information on decomposition in the TASK BASE in order to create up-to-date plans. This process will be described later in section 6.

The planning agent has to be aware of the state of its peers. This is a necessary condition for the success of the suggested approach. To be informed the agent subscribes some others for information on their state. This way, the agent gets the relevant data directly without being forced to rely on the facilitator.

Instead of the facilitator, we suggest to introduce another unit in the agents' community, the *meta-agent*. Our meta-agent is in charge of monitoring the overall load and communication traffic. The main aim of the meta-agent is to reason about these data in order to identify knowledge, which might prove useful for conduct of the community. The meta-agent searches for example for possible mutual relations among some seemingly independent tasks. Such explicit knowledge can significantly extend the strength of the agent community by making it able to cope with more tasks or to solve them more efficiently. The meta-agent can update knowledge in all the bases of the agents.

Let us review the updating process in each of the three bases more carefully:

TASK BASE: The *problem section* concerning decomposition of tasks can be understood as permanent. Changes in this section reflect the results of the meta-agent's reasoning mostly. These are treated in

the Section 7. Updating of the *plan section* consists of checking the available plans and in construction of the new ones – cf. Section 6.

STATE BASE: Reliable information in the *agent section* of STATE BASE of any agent represents its awareness about the activity of its co-operating agents. This knowledge becomes the central issue of our approach. Data in the *agent section* characterise the dynamic (changing) resource of the planning process - which is why they have to be well updated and maintained. This is ensured by the subscription mechanism. Any agent can contract some of its collaborators from the CO-OPERATOR BASE and subscribe these to report on their status. Every time the state of the subscribed agent changes, this subscribed collaborator sends a message to the original agent in order to let it know that the STATE BASE is to be updated. Update of the *task section* is a result of the planning activity, which is triggered whenever a new task is passed to one of the agents of the multi-agent community.

CO-OPERATOR BASE: This base stores basically such data regarding collaborating agents which do not require any often updates because they are semi-permanent. The meta-agent that is in charge of monitoring the agent's operation may detect some suspicious irregularities or changes in its activity. Either loss or enhancement of an agent's capabilities is viewed as examples. Thus, the meta-agent is supposed to update the CO-OPERATOR BASEs of the agents.

6. Generation of Plans

Suppose an agent a is in charge of a *SuperTask*. The agent can either

- (i) use an existing plan stored in the *plan section* of its TASK BASE or
- (ii) generate a new plan through the pre-planning phase using its own methods and skills.

In the latter case the agent a takes problem knowledge found in the *problem* section of the TASK BASE of the agent a . Here it finds a piece of general decomposition knowledge for the considered *SuperTask* in the form

$$\langle \text{SuperTask}, \{\{T_i\}, \text{Precedence}\}, \text{Conditions}\rangle,$$

where $\{T_i\}$ is the set of tasks, the *SuperTask* is to be decomposed into.

For simplicity, let us suppose that the set *Precedence* is empty. We will return to the more general case later. The agent a consults its CO-OPERATOR BASE in order to detect the possible collaborators for all the tasks in $\{T_i\}$. The agent a is supposed to find a good match for each task from the set $\{T_i\}$, i.e. to complement each task T_j with the name of an agent A_j that is ready to cope with that particular task. The requested result is a set of couples $\{\langle A_j, T_j \rangle\}$ meeting the following requirements:

- For each task T_j the agent a finds an agent A_j such that there is a $\langle A_j, _ , _ , T _ \rangle^1$ stored in the CO-OPERATOR BASE of agent a and $T_j \in T$.
- For the set $\{\langle A_j, T_j \rangle\}$ the applicability *Conditions* are checked. Only those couples, which do not contradict *Conditions*, are evaluated further.
- The evaluation of the *Trust* is carried out under consideration of the $Load_{A_j}$ and $Trust_{A_j}$ of the agents A_j as parameters, where $\langle A_j, Load_{A_j}, Trust_{A_j} \rangle$ is contained in the STATE BASE of a . Just those couples $\{\langle A_j, T_j \rangle, \text{Precedence}\}$ with the best evaluation are considered and they form possible plans:

$$\langle \text{SuperTask}, \text{Agent}, \{\{\langle A_j, T_j \rangle\}, \text{Precedence}\}, \text{Constraints}, \text{Trust}\rangle.$$

¹ The symbol $_$ stands for an anonymous variable the value of which is not relevant

Trust can be either minimal trust of all the agents the task is delegated to, average, or weighted average. The quadruple with the highest *Trust* is viewed as the actual plan.

If the contents of the STATE BASE (the agent section of which is an important resource for the plan construction) are updated, *Trust* is to be re-computed and each of quadruples is to be re-evaluated consequently. This kind of re-planning activity makes the plan the most up-to-date and bridges the planning and execution stages of the problem solving process.

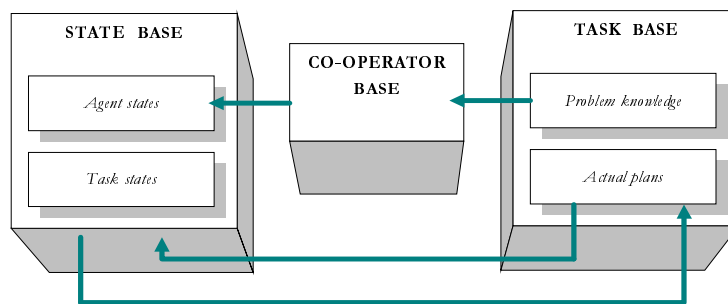


Figure 2

This is reasonable to be done for a limited number of tasks. To identify them is an interesting task for application of machine learning in the multi-agent systems. Now let us consider the general case, where precedence constraints for some tasks apply. One possibility how to address this is based on the idea of planning without any scheduling mechanism, which is one of the options offered within the tri-base model. In this case, we substitute planning by the negotiation process. Let us consider a task T . When solving task T , an agent has to find out when the required preceding task P_1 , specified in the precedence constraints, should be finished. If the plan for P_1 has not been created yet, the agent has to plan P_1 first and return to the task T later. If P_1 is in the *task section* of STATE BASE of the considered agent already and its due time is b_1 , then the agent proceeds in following way: Instead of contracting the responsible agent for T immediately, the agent consults *agent section* to choose among the candidates that one which will be available as close as possible to b_1 . This is the way the agent specifies the time when it can start planning the task T . If no such information is found in the

STATE BASE, the agent finds an appropriate collaborator in its CO-OPERATOR BASE and *subscribes* it for information on the availability.

If the agent is to delegate the task T further onto other agents, it has to let them know the starting time given by the end of the preceding task. This piece of information is stored in the *task section* of the STATE BASE.

7. The Meta Agent

7.1 The Role of the Meta-Agent

As already claimed, the main distinction between the *tri-base* meta-agent and a classical facilitator or broker in the GENIE terminology [Toomey 96] is based on the fact that the community can well survive without the meta-agent. The meta-agent contains higher level knowledge [Zhong 97]. The community needs the meta-agent in order to optimise the communication traffic, whereas the facilitator really facilitates inter-agent communication. The facilitator is the central agent, which can become easily the communication bottleneck in the case of intense inter-agent information exchange.

In the *tri-base* model agents are encouraged to communicate in the ‘peer-to-peer’ fashion and the meta-agent is supposed to provide information for revision of agents’ bases in idle times.

Suppose a system uses peer-to-peer communication only. Methods how a single agent can conclude that one of its co-operating agents is dead can be designed. However, this informed agent has not many means to pass this important news to the rest of the community while opposite is true about the meta-agent. The meta agent is aimed at continuous observation of the pre-specified part of the community with the following goals:

- to discover a failure of a particular agent,
- to find out some behavioural patterns appearing in the community, e.g. dependencies among tasks solved by the system (e.g. “task T_1 often follows immediately after task T_2 ”) or among load distributions of various agents with the aim to predict future behaviour of the system.

In the tri-base approach, the meta-agent provides information it can deduce from the experience during the lifecycle of the community. The meta-agent can observe behaviour of a single agent and thus make better estimate of its statistics (see [Hazdra 96]). Nevertheless, this activity can be achieved within a single agent, too. The more important part of the meta-agent’s capability is due to qualitative reasoning mentioned in [Štěpánková 97].

Knowledge obtained by the meta-agent can be used to achieve more efficient system’s behaviour. The meta-agent can do so by revising knowledge the agents of the system use for creating their behavioural plan. The main used revisions will be described in detail. They are concerned mainly with the problems how

- to re-distribute the load by task re-allocation,
- to create a new agent if appropriate (by copying or ‘cloning’ some agents).

Communication messages are the input information for the meta-agent. The desired output of the meta-agent is actions aimed at revisions of the content of the tri-bases of all the agents.

The meta-agent can for example predict critical overload of an agent a . In such a case it updates the CO-OPERATOR BASEs of all agents in the community in order to prevent them from contracting the overloaded agent a causes huge waiting time.

If there is time enough the meta-agent can carry out more sophisticated, qualitative reasoning. The results of such reasoning are heuristics that help the planning process within particular agents (e.g. by

introducing new precedence constraints). They correspond to problem solving '*shortcuts*' regarding triggering of the task planning or suggesting a specific type of solution. The proper place, where to store such information is the *problem section* of the TASK BASE. As the agent itself may also induce a heuristic of an arbitrary kind this section is to be updateable by both the meta-agent and the agent itself.

As another example of heuristics, we may see formulae describing the frame/ramification problem of planning to appear in conditions of decomposition. Careful observing of the periodical revision may reveal what pieces of information within the STATE BASE are substantial and which are irrelevant to the plan revision process.

The meta-agent is currently implemented so that it monitors nothing but the communication traffic among agents within the community. We have experimented with few possible ways of its operation. Firstly, each agent when sending a certain message to another forwards a copy to the meta-agent. Although agents are knowledgeable of types of message which are of any importance to the meta-agent and thus the communication requirements does not grow dramatically, we thought of using a blackboard architecture for assuring that the meta-agent is having the right piece of information. Though agents communicate peer-to-peer they share a blackboard structure where each of agents is allowed to write and the meta-agent is allowed to read. When writing on the blackboard, an agent forks out a parallel thread that waits until it the blackboard is available and does not make the agent wait.

The meta-agent technology was successfully applied in some of the projects carried out at The Gerstner Laboratory at CTU Prague. For instance, a simple version of this approach was implemented in the DISCIM multi-agent prototype environment for decision-making [Mařík 96a]. Similar ideas have been proposed to solve the reconfiguration problems in distributed control

systems [Mařík 96b]. Some trends of making the community behaviour more flexible and efficient were already outlined in [Štěpánková 97].

7.2 Revision Processes

Capacity of an agent a to participate in problem solving activity of a multi-agent system described by a tri-base model is determined by

1. the agent's problem solving capabilities summarised in the problem section of the TASK BASE of a and
2. by actual availability of the agent a characterised by data on its actual load, trust, capabilities and schedule described in STATE BASEs of those agents which have subscribed for this information.

All of these characteristics of an agent do not remain fixed during the life-cycle of the agent community [Kiss 96]. That is why corresponding revision processes have to be designed. Since the tri-base model is an extension of the twin-base model, the set of tri-base revisions has to include all the revisions considered in the twin-base model [Cao 96, 97]. They correspond to birth or death of an agent or to change of the agent's problem solving capabilities described by addition/deletion of a specific *SuperTask* decomposition into/from the *problem section* of its TASK BASE, respectively. These are the revisions 1-4 described below. The additional revisions specific for the tri-base model have to care for all those dynamic properties, which are not considered in the twin-base model. This is the case of a change of actual availability of the agent (revisions 5, 6). The list of revisions is concluded by that one corresponding to a specific change in problem solving capabilities of an agent due to re-definition of *Precedence* or *Conditions* in an existing *SuperTask* decomposition. This is the last item, the revision 7, in our review of available revision processes. These revisions cover all possibilities in which problem solving activity of an agent described by tri-base model can change.

When are these revisions fired? We can distinguish at least two reasons to trigger certain changes in the knowledge bases of any agent c :

1. either the agent c identifies itself something important in its environment (among its co-operators), or
2. the need for change is identified by the meta-agent.

In the former case, the revision processes to be described appear within the wrapper of the agent itself only. If the need for change is identified by the meta-agent, the meta-agent has to define the set of agents concerned - often it can contain all the members of the community.

Let us describe all the relevant revisions which any specific agent c has to do within its wrapper as soon as c identifies the need for revision (itself or due to information from the meta-agent). The same types of revisions have to be done simultaneously within the wrappers of all the agents if the source of impact for revision is the meta-agent, of course.

Revision 1: Agent A_i died: All the bases of the agent c have to be revised: whenever a_i occurred in the CO-OPERATOR BASE of an agent c , then the corresponding quadruple (introduced by a_i) has to be erased from both sections of CO-OPERATOR BASE and *agent section* of the STATE BASE. If a_i appears in a quadruple R in the *plan section* of the TASK BASE of c , this quadruple R has to be erased, too. Moreover, if there is no alternative solution for the *SuperTask* of the erased tuple in the TASK BASE, new solution has to be suggested for this *SuperTask*. The same is true for those tasks in the *task section* which are assigned to the agent A_i and which have not been finished, yet.

Revision 2a: The task T_j cannot be solved under the co-ordination of or by the agent A_i any more (the contents of agent's A_i problem section is changed but the agent didn't die and is still capable to solve some remaining tasks). Both the CO-OPERATOR BASE and TASK BASE of the agent c have to be revised. Suppose $\langle a_i, Address_i, Language_i, S \rangle$ occurred in the CO-OPERATOR BASE of the agent c , then this

quadruple has to be replaced by a new quadruple, $\langle a_i, Address_i, Language_i, S_1 \rangle$, where S_1 is $S - \{T_j\}$. The plan section of the TASK BASE of c has to be revised: whenever the pair $\langle T_j, a_i \rangle$ occurs in any quadruple R within this base, that quadruple R has to be erased. If the task T_k appears in the *task section* of STATE BASE of the agent c assigned to a_i , and the task T_k has not been finished, the corresponding entry in the TASK BASE has to be erased and a new alternative solution of T_i has to be designed.

*Revision 2b: Single specific decomposition of a given SuperTask, namely $\langle SuperTask, \langle \{T_j\}, Precedence \rangle, Constraints \rangle$, is to be deleted from the problem section of the TASK BASE of the agent a_i (a_i has some more alternative solutions for this *SuperTask*). This revision influences all the bases of the agent a_i - the change in the *problem section* is obvious. Those collaborators in the corresponding base can be deleted which have been present for the sake of the considered decomposition only. Simultaneously agent a_i cancels its subscription to the agents, which have been deleted from the *collaborators section*. The same agents are deleted from the *agent section* of STATE BASE of a_i . Revision of the *task section* proceeds as in *Revision 2a*.*

Revision 2c: More sophisticated version of this revision describes influence of simultaneous deletion of a set of decompositions from the TASK BASE of a_i .

*Revision 3: A new agent b is born. The new agent is defined by the contents of its CO-OPERATOR BASE and of the *problem section* of its TASK BASE. The CO-OPERATOR BASE of any agent c appearing in the *subscribers section* of b is extended by data expressing all necessary information about the new agent (the address and the ability to treat those problems present in b 's *problem section*).*

Revision 4: An existing agent b gained a new capability to solve/co-ordinate T_j by, e.g., learning (the problem section of the TASK BASE of b is enriched by this new information). The CO-OPERATOR BASE of

the agent c is revised as follows: any member $\langle b, Address_b, Language_b, S \rangle$ is replaced by a new quadruple $\langle b, Address_b, Language_b, S_1 \rangle$, such that S_1 is $S \cup \{T_j\}$.

The remaining three types of revisions are triggered by the meta-agent often. That is why we will describe them with respect to a set of concerned agents.

Revision 5: Changes of the agent's capabilities (as a result of increasing/decreasing the computational power of the agent, or of changing the transmission capabilities of the communication channel) have to be reflected in the *agent sections* of all the agents, which have b among their collaborators. The tasks, assigned to b , which have not been finished and which appear in the task section of their STATE BASE have to be verified under new conditions. If necessary, suggested solution has to be replaced by another one.

Revision 6: Change in the load estimate L_i , trust T_i or capabilities of the agent b as a result of expected problem solving activities to be performed by the agent A_i . The new estimate L_i (or T_i) is broadcast to the STATE BASEs of some (or all) relevant agents (those agents which have b mentioned in their CO-OPERATOR BASE). This change has to be followed by revision of the plans in the plan sections of the TASK BASE of the same agents.

*Revision 7: More precise formulation of the conditions C_1 or precedence constraints P_1 as conditions for specific decomposition $\langle SuperTask, Tasks, _ \rangle$ of a *SuperTask*.* This process influences the TASK BASEs of all the agents that are concerned with decomposition of *SuperTask*. First, their problem section has to be updated by replacing each triple $\langle SuperTask, \langle Tasks, P_0 \rangle, C_0 \rangle$ by a new one, namely $\langle SuperTask, \langle Tasks, P_1 \rangle, C_1 \rangle$. Then the corresponding plan section of the same agent has to be checked and re-computed.

8. TESLA-TV Application: ProPlanT System

TESLA TV is a Czech manufacturing enterprise producing TV transmitters, FM transmitters and passive elements. The manufacturing process within TESLA TV can be classified as a project oriented production. There is no conveyor-belt type of assembly line used and it is very difficult to formalise the production as a continuous process. The production goal is to manufacture (mainly assemble) a single particular transmitter. Consequently, rather than planning and simulation of a flow of semi-products to be assembled, there is a need for an information technology solution which would facilitate simple planning and subsequent optimisation of the unique final product manufacturing.

Currently ill-suited information flows, lack of communication among particular production units and far from full utilisation of available information processing solutions is what makes all of the production difficult to understand, model, plan and consequently optimise. Attempts to create a global, “monolithic” software solution, no matter how well hierarchically structured, have failed. There is a set of diverse software tools (from our point of view legacy software systems) used in the plant currently to solve some partial tasks. On one hand, it is crucially important to integrate these partial solutions and on the other hand, the flexibility and frequent changes in the production facility inevitably require highly distributed solutions. Multi-agent approach seems to be suitable in order to address this problem.

The TESLA-TV plant was thoroughly analysed first. This analysis served as a basis for design of the global architecture of the proposed distributed system that was intended to model planning and managerial activities in the factory. The following factory units have been included into consideration:

- SALES DEPARTMENT involving (1) *Sales and marketing group*, (2) *Sales management group*, and (3) *Technical/Commercial department*.
- PROJECT DEPARTMENT is responsible for (1) *engineering* and (2) *technical management*.
- PRODUCTION MANAGEMENT DEPARTMENT is accountable mainly for (1) *assembling to order* and for (2) *prototype manufacturing*.
- PURCHASE DEPARTMENT is responsible for the (1) *material supply* and (2) *parts supply*.
- RESEARCH AND DEVELOPMENT DEPARTMENT
- STORE DEPARTMENT maintains the physical store of (1) *raw materials*, (2) *finished parts* and (3) *completed solutions*
- CONSTRUCTION AND TECHNOLOGY DEPARTMENT comprises of (1) *Project group* and (2) *Technology group*.
- IT DEPARTMENT is accountable for *maintenance and upgrading all the information system* within the enterprise.
- MARKETING DEPARTMENT cares about global accountancy including wages and employee information.

Analysis of information flows among these departments strongly supported the choice of a multi-agent system as a suitable environment for building ProPlanT - Production Planning Tool. Its architecture is based on the following principles:

- Any detailed project plan/schedule is prepared because of information exchange among finite number of agents (each agent corresponds to one or more company units).
- Each agent in the ProPlanT belongs to one of the following disjunctive sets of agents:

- (i) PRODUCTION AGENT (PA) representing the function of a foreman – each production or store department is represented by this kind of agent that models its behaviour/performance.
 - (ii0) PRODUCTION MANAGING AGENT (PMA) playing the role of a production manager – the production management department is represented by a set of independent PMAs.
 - (ii0) PROJECT PLANNER AGENT (PPA) – a single agent of this kind simulates the functions of the project department as well as the construction and technology department.
- From the functional point of view, the agents are organised into a hierarchical structure (see Figure 3). The outputs of the production planner agent (PPA) appear first in the form of a set of tasks to be carried out and a set of precedence constraints for these tasks. Later, PPA structures the original set of tasks into a sequence of task groups in such a way that each group contains only those tasks which can be processed in parallel without violation of the fixed precedence constraints. The tasks from one group (those which can be fulfilled in parallel) are sent out to relevant PMAs according to the PMA specialisation. PMA solves the task, if asked to, by further dividing it into jobs, which are mostly assigned to PAs.

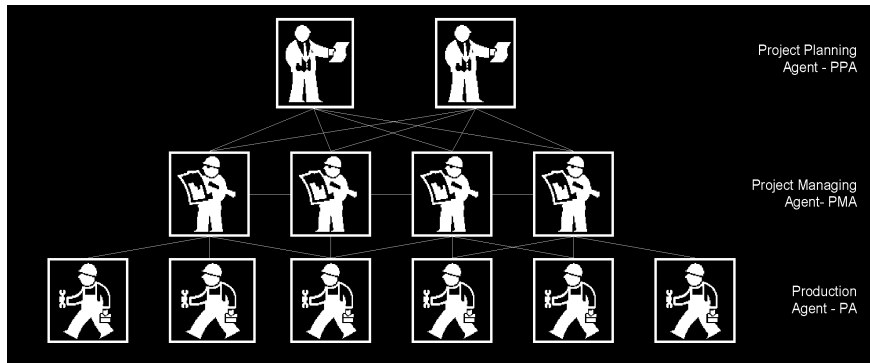


Figure 3

- The tri-base model formalises activities of the three agent types mentioned above. The CO-OPERATOR BASEs contain permanent or semi-permanent knowledge on production facilities (their structure, transportation paths, capacity, etc.) as well as knowledge how to organise and plan projects. Part of the co-operator-base knowledge may be of heuristic nature. The STATE BASEs contain temporary information on load, processing time, trust, etc. The TASK BASEs are expected to store intermediate results of the planning process, i.e. the directives how to plan tasks and jobs.
- Agents incorporated within the ProPlanT model communicate through KQML (knowledge query manipulation language) message [Finin 95], [Labrou 97] with KIF (knowledge interchange format) content [Genesereth 92].
- Various meta-agents can be introduced into the agents' community in a simple way. They can e.g. collect information on the activities of the other agents in the community, evaluate data and utilise the obtained results in order to increase efficiency of the overall system.

8.1 Project plan construction

In the following, we describe the process of creating a plan distributed alongside the community. As all community agents play a specific role in the planning process, we comment on planning of each separately.

8.1.1 PROJECT PLANNING AGENT

The duty of PPA agent is threefold. It has to specify a set of fundamental planning components – *configuration*, resolve prerequisite constraints – *ordering* and contracting the appropriate PMA peers – *delegation*. Result of the first stage is a plan in form of a quintuple as follows:

$$\langle P_j, PPA, T_j = \{T_j^k\}, C_j, V_j \rangle,$$

where P_j represents the project under consideration, PPA stands for identification of the responsible PPA agent, T_j is a set of tasks the project P_j consists of, C_j is a set of constraints concerning precedence between considered tasks denoted by the relation $<$ (saying that T_j^i proceeds T_j^k if $T_j^i < T_j^k$) and V_j is a trust in plan, some sort of subjective value. The couple $\langle T, C \rangle$ represents the current solution status of the project planning on the PPA level. This working plan is placed in a TASK BASE of the responsible PPA. The plan is gradually refined and made more accurate by the agents on lower levels of the agents' hierarchy. In order to ensure proper time sequencing of tasks $T_j^k \in T$ the agent searches for partial ordering of the task with respect to constraints C_j . The plan $\langle T, C \rangle$ is transformed into a partially ordered set $B = \{B_n\}$ suggesting grouping T into a sequence of task sets B_i of those tasks that can be carried out in parallel without violating the constraints in C . Moreover, for each group B_i there must hold that its tasks have to be completed before tasks in B_{i+1} are started.

We have suggested two different extreme strategies for ordering T . Let's call them *lazy* and *eager* strategies. The former tries to postpone all activities as late as possible, while the *eager* strategy

executes each task as early as possible in such a way that all the constraints are met. With respect to the relation \prec defined in previous paragraph, *lazy* production shall meet the following requirements:

$$\begin{aligned}
& \bigcup_{i \leq n} B_i = T \\
& (\forall i, j \leq n)(i \neq j \Rightarrow B_i \cap B_j = \emptyset) \\
& (\forall x \in B_1) \neg [(\exists y \in T)(y \prec x)] \\
& (\forall i > 1)(\forall z \in B_i)(\forall w \in T)(w \prec z \Rightarrow (\exists j < i)(w \in B_j)) \\
& (\forall i < n)(\forall z \in B_i)(\exists y \in B_{i+1})(z \prec y)
\end{aligned}$$

Example 1:

Let's consider the project aimed at manufacturing and assembling an aerial with the corresponding electrical circuit. The Solution1 is as follows:

Solution₁ = $\langle \{ \text{manufacturing of amplifier } (T_1), \text{ manufacturing of power-supply } (T_2), \text{ manufacturing of cover } (T_3), \text{ coating of components } (T_4), \text{ assembly of amplifier and power-supply } (T_5), \text{ assembly of antenna } (T_6), \text{ coating of antenna } (T_7), \text{ electrical attachment of antenna } (T_8), \text{ mechanical attachment of the electric circuit to the power-supply } (T_9), \text{ maintenance testing and verification } (T_{10}) \}, \{ T_1 \prec T_3, T_2 \prec T_5, T_3 \prec T_4, T_4 \prec T_5, T_6 \prec T_7, T_7 \prec T_8, T_5 \prec T_9, T_9 \prec T_{10}, T_8 \prec T_{10} \} \rangle$, where \prec is a symbol for time precedence. Figure 4 depicts the “eager” Solution₂ described in symbols as follows:

$$\text{Solution}_2 = [B_1 = \{T_1, T_2, T_3, T_6\}, B_2 = \{T_4, T_7\}, B_3 = \{T_5, T_8\}, B_4 = \{T_9\}, B_5 = \{T_{10}\}]$$

Analogously, Figure 5 shows the “lazy” solution corresponding to the sequence:

$$\text{Solution}_3 = [B_1 = \{T_3\}, B_2 = \{T_1, T_2, T_4, T_6\}, B_3 = \{T_5, T_7\}, B_4 = \{T_8, T_9\}, B_5 = \{T_{10}\}]$$

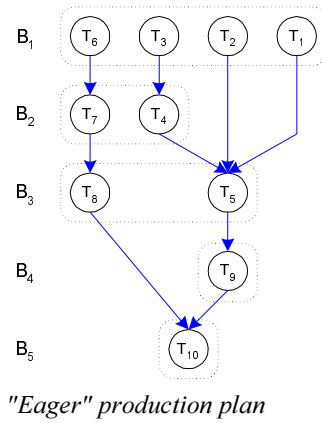


Figure 4

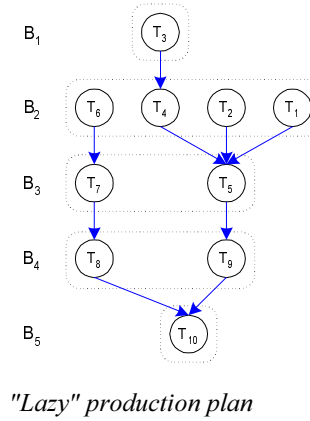


Figure 5

Here, the tasks that have no “direct follow-ups” are executed as late as possible in order to minimise the overall “waiting” time of intermediate products. Let’s consider the production management agents (mentioned in a part of the CO-OPERATOR BASE of the PPA) given in the Table 1:

PMA ₁	Manufacturing of electrical systems	T_1, T_2
PMA ₂	Mechanical workshop	T_3
PMA ₃	Coating studio	T_4, T_7
PMA ₄	Mechanical assembling	T_5, T_6, T_9
PMA ₅	Electrical assembling	T_8, T_{10}

Table 1

Table 1 recommends a logical task assignment. It suggests PMA to direct the tasks T_4 and T_7 aimed at coating directly to PMA₃ “coating studio”, the task T_3 to PMA₂ “mechanical workshop”, etc.

8.1.2 PROJECT MANAGING AGENT

The PMA agents decompose requirements detected (*decomposition*) and reason on available sub-community of collaborating agents such that the decomposed subtasks would be delegated in the best (cheapest, fastest) way (*delegation*). The decision-making activity of a PMA is based on the tri-base

model. Knowledge on how to decompose the tasks into jobs is contained in the CO-OPERATOR BASE, the current situation in the manufacturing workshops is represented by the state of the STATE BASE and the intermediate results of the PMA planning activity are stored in the TASK BASE in the form of quintuples as follows:

$$\{\langle T_j, PMA_j, T = \{\langle T_j^k, A_j^k \rangle\}, C_j, V_j \rangle\},$$

having the following interpretation: the production management agent PMA_j manages the solution process of the task T_j by carrying out of the sequence T of jobs under the limitation of the constraint set C_j and the accompanying parameters V_j . The PMA_j decomposes each task in subtasks, each of them being carried out by just one agent belonging to the scope of PMA_j 's subordination. As the structure of PMA agents may be distributed alongside a number of levels either PA agent or another PMA agent may be considered as a subordinated agent if such a relation exists within the community. The decomposition then looks as follows:

$$T = \langle \{\langle T_j^k, PMA_j^k \rangle\}, \{\langle T_j^k, PA_j^k \rangle\} \rangle$$

The process of generating a plan within a tri-base model is thoroughly treated in Section 6. If the content of the STATE BASE (the agent section of which is an important resource for the plan construction) are updated, $Trust$ is to be re-computed and each of the quadruples is to be re-evaluated consequently. This kind of re-planning activity makes the plan the most up-to-date and bridges the planning and execution stages of the problem solving process.

8.1.3 PRODUCTION AGENT

The PA agent simulates the duties of a foreman on the shop floor level. It reasons about scheduling parallel jobs on a number of machines. As the main objective of the paper is to show that planning may be carried out through negotiations instead of engaging sophisticated scheduling mechanism, we have been using the tri-base model also on the level of Production Agents. Instead of monitoring

states of the collaborating agents, the STATE BASE contains information on agendas on parallel machines administrated by the agent. The knowledge stored in its CO-OPERATOR BASE enables the agent to consider schedules of each relevant job as either a single operation on a single machine or as a sequence of operations carried out on (possibly) different machines. Each job schedule is equipped by an additional piece of information describing the time and costs of carrying out the job under current manufacturing circumstances. This additional information is computed from the data in the STATE BASE. When planning, the best possible prepared plan from the *plan section* of the TASK BASE, where a set of plans is kept up-to-date, is chosen and the STATE BASE is updated correspondingly.

Example 2:

It is necessary to distinguish between pre-planning, planning, and plan fixing phases. Plan fixing is applied just before the manufacturing process starts. During the preliminary planning acceptable quintuplets are created and inserted into the TASK BASE. At the operational planning, the currently most appropriate quintuplets in the TASK BASE are chosen. While planning, the PA receives sequence of jobs to be carried out from the PMA. That means that the decomposition of the task into jobs is performed by PMA using its CO-OPERATOR BASE. There are two options:

- One job represents just one operation in one machine
- One job requires a sequence of operations carried out on the same or different machines.

In the latter case, the PA decomposes the job into a sequence of operations using the content of its CO-OPERATOR BASE. This process is strictly analogous to that of decomposing tasks into a sequence of jobs on the PMA level.

Thus, without any loss of generality, it is possible to consider that one job is carried out by just one operation (and this is the case, which just occurs in TESLA TV factory). The request message from a PMA (a sequence of jobs = a sequence of operations) is compared with the content of the PA's CO-OPERATOR BASE and the STATE BASE. As a result an optimal couple $\langle O, M \rangle$ or a set of acceptable couples (for each operation) are shifted to the TASK BASE in the form of complete corresponding quintuple or quintuples.

Samples of the contents of both, the CO-OPERATOR BASE and the STATE BASE are shown in tables below.

Operation	Machine	Time	Cost
093390063000	16611	15	96
023460340000	16315	10	0.2
023400735000	09615	10	0.4
023400735000	16517	10	0.6
023965018000	16313	5	0.3

Table 2

Table 2 represents a fraction of the CO-OPERATOR BASE of the PA agent which simulates an activity of the department no.: 685 (a big part assembly line) in TESLA TV. There are 5 machines to carry-out 4 kinds of operations. A job 023400735000 (frequency mixer installation) can be performed using either 09615 or 16517 machine. Both are equivalent time-wise but using the former one is cheaper.

Machine	Capacity	Trust
09615	10	0.904
16315	45.2	0.811
16517	8	0.862
16611	0	1.00
16313	22	0.989

Table 3

Table 3 represents a fraction the agent section of the STATE BASE of the PA agent which simulates an activity of the department no.: 685 (a big part assembly line) in TESLA TV. The capacity column shows the load of the machine in terms of number of time units needed to finish its actual job. In this case, the 16611 machine is vacant.

During the second, operational planning, the pre-prepared quintuples already stored in the TASK BASE are matched to the actual data located in the STATE BASE. Only the best fitting quintuples (=local plans) are included into the final plan and the required capacity of the corresponding machine is claimed.

8.2 Message Conventions and Treatment Scripts in

ProPlanT

Using the tri-base acquaintance model within the wrapper of each of the agents significantly reduces the need of inter-agent communication. As it was seen above, substantial part of negotiation processes is replaced by planning carried out internally - without any vast communication with the others - by the agents responsible for the given task. Instead of the necessity vastly to communicate, the agent consults its STATE BASE to find the best peer to co-operate.

There are five KQML performatives used in the message passing process:

- EVALUATE – we have used *evaluate* performative for the process of production pre-planning. An addressee is supposed to consider possible variants on how to achieve the request, insert those into plan section of the TASK BASE and reply with best possible due-times and budgetary requirements.
- ACHIEVE/UNACHIEVE – we have used *achieve* performative for the process of task planning. An addressee shall take the best possible prepared plan from the plan section of the TASK BASE and trigger the planning accordingly. This activity results in updating task timetables of the PA agents. The timetable of an agent is stored in the task section of the STATE BASE. *Unachieve* performative facilitates backtracking when planning fails at a certain level of decomposition. It removes unneeded tasks from the STATE BASES.
- REPLY/SORRY – *Reply* performative reports on successful (pre-)planning process invoked by both *evaluate* and *achieve*. A sender replies with due-times and total costs. While *reply* reports on successful fulfilment of the requirement, *sorry* reports on inability to do so.
- SUBSCRIBE – This performative allows a sender to ask an addressee for reporting information on his status, on certain tasks it is responsible for respectively.

- ADVERTISE – Advertise performative broadcasts to a given sub-community of agents, which subscribed for, information on its state of the course of problem solving of a certain task. Advertise is a reply for *subscribe* performative.

The content of a message is either included into the message itself, or the content is to be transmitted through a shared database medium. That is why we distinguish between *information-rich* message, where the processed data are encapsulated within the message content, and *information-poor* messages, where the only information within the message content is a pointer to the shared database and an appropriate filter. An example of the former case is as follows:

```
(
  :performative evaluate
  :sender pma-243
  :content (job:[90342, 12390, 23657, 00120])
  :receiver pa-685
  :reply-with [?time, ?cost]
  :language KIF
  :ontology nil)
```

where the pa-685 agent is asked to plan list of tasks (90342, 12390, 23657, 00120) and to reply with information on time and costs needed to the pma-243 agent.

8.2.1 PRODUCTION AGENTS (PA)

On the PA level there is a local planning carried out through processing a distributed database of machines and other resources available to the given workshop. The jobs are scheduled if requested.

A PA agent administers two tables of a distributed, shared database medium - (1) its internal data specifying its responsibilities, times and costs of any operation it is capable to undertake and (2) a time database simulating the time-flow. The latter one is updated when a request for simulation and scheduling is received. Talking in terms of the tri-base model, the former is encapsulated within the CO-OPERATOR BASE, whereas the latter one is stored within the STATE BASE of the given A.

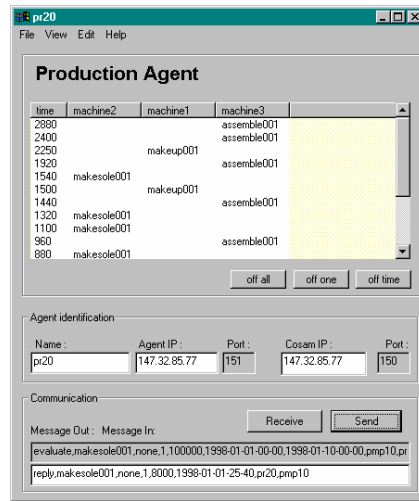


Figure 6

There are three kinds of messages the PA can accept: *evaluate*, *achieve* and *subscribe*. The message with the performative *evaluate* invokes the action *query* which offers a solution of the required task/job and estimates costs and deadlines. The message with the performative *achieve* triggers an *execute* action which does the same as the query, but simultaneously refines in the corresponding way the time database – updates the STATE BASE of the tri-base model. *Subscribe* reports on changes within its STATE BASE to all subscribed agent *A*. PA agent can send a message with two kind of performative – *reply* and *sorry*. The agent has been implemented in C++ and runs in Windows NT environment. It maintains its time agenda in a SQL database.

Behaviour of a PA agent is given by a set of general treatment scripts part of which is depicted in Table 4.

TREATMENT SCRIPT: <i>evaluate</i> performative	
IF: - content : <i>(job or operation)</i> - sender : <i>PMA or PA</i> - other conditions : <i>task-base conditions</i>	THEN : <i>decompose_task(tb)→select_machines(cb)→ check_states(sb)→append_plans(tb) select_best(tb) →</i> PRIORITY : <i>Send peformative: reply/sorry to: sender nil</i>
TREATMENT SCRIPT: <i>achieve</i> performative	
IF: - content : <i>(job or operation)</i> - sender : <i>PMA or PA</i> - other conditions : <i>task-base conditions</i>	THEN : <i>select_best(tb)→append_pl ans(sb)→</i> PRIORITY : <i>Send peformative: reply/sorry to: sender nil</i>
TREATMENT SCRIPT: <i>subscribe</i> performative	
IF: - content : <i>(job or operation)</i> - sender : <i>PMA or PA</i> - other conditions : <i>nil</i>	THEN : <i>set_subscriber(sb, sender)</i> PRIORITY : <i>Send peformative: advertise to: sender content: state nil</i>

Table 4

8.2.2 PROJECT MANAGING AGENTS (PMA)

PMAs manage the process of project planning and simulation within the multi-agent community. Each of PMAs is responsible (1) for a group subordinated PAs and/or (2) a group of subordinated PMAs (if such subordination has been created). Consequently, it is supposed to be aware of agents it manages, about their abilities and limitation. At the same time, it shall know the product structure in order to allocate tasks and delegate further responsibilities if necessary.

This agent is fully based on the tri-base acquaintance model. The pre-planning phase is invoked by *evaluate* KQML message, the re-planning phase is started by the *advertise* KQML message sent from its peers, and the plan fixing phase is triggered by the *achieve* KQML message. Apart from those three performatives used by PA agent, the PMA agent accepts also *advertise* message, which starts the updating of its STATE BASE.

As the previous, the PMA agent was implemented in C++.

The functionality of the PMA agents used in the first ProPlanT prototype is represented by a general treatment scripts with its specific instantiation. Part of it is depicted in Table 5.

TREATMENT SCRIPT: <i>evaluate</i> performative			
IF:	- content : <i>task</i>	THEN :	<i>decompose_task(tb)→select_peers(cb)→check_states(sb)</i>
	- sender : <i>PMA or PPA</i>		<i>→append_plans(tb), select_best(tb) →</i>
	- other conditions : <i>task-base conditions</i>	PRIORITY :	Send performative: reply/sorry to: sender
			<i>nil</i>
TREATMENT SCRIPT: <i>achieve</i> performative			
IF:	- content : <i>task</i>	THEN :	<i>select_best(tb) → peers, jobs → append_state(sb)</i>
	- sender : <i>PMA or PPA</i>		<i>Send performative: achieve to: peers content: jobs</i>
	- other conditions : <i>task-base conditions</i>	PRIORITY :	<i>nil</i>
TREATMENT SCRIPT: <i>reply</i> performative			
IF:	- content : <i>j</i>	THEN :	<i>select_parent(sb) →</i>
	- sender : <i>o</i>		<i>compose_jobs(sb) →</i>
	- other conditions : <i>b</i>	PRIORITY :	<i>task_state, parent</i>
	<i>PMA or PA</i>		<i>Send performative: reply to: parent content: task_state</i>
	<i>task-base conditions</i>		<i>nil</i>
TREATMENT SCRIPT: <i>advertise</i> performative			
IF:	- content : <i>agent_state, task_state</i>	THEN :	<i>update_state(sb)</i>
	- sender : <i>PMA or PA</i>		<i>update_task(sb)</i>
	- other conditions : <i>nil</i>	PRIORITY :	<i>nil</i>

Table 5

8.2.3 PROJECT PLANNING AGENTS (PPA)

Project planning agent is a different kind of agents as far as its structure and behaviour is concerned. In the case of the project driven production it is quite tricky to specify thoroughly a product that would both satisfy needs of a customer and would be possible (and cheap enough) to be produced. Certain performance/budget related compromise must be undertaken and considerable effort is usually devoted to translation of customer requirements into the 'language' of the enterprise. For this, we have used a special Project Configuration Agent – an expert system written in LPA Prolog and encapsulated within the tri-base wrapper (Figure 6). It is based on proof planning programming methodology and is able to configure the solution while it keeps the object level data and inference knowledge separate [Pěchouček 97].

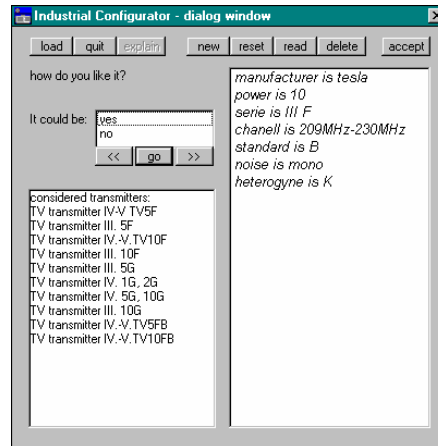


Figure 6

Apart from this, the PPA is supposed to analyse entire enterprise ontology and set permanent bases of other agents with respect to possible production on the shop floor. It sets the knowledge section of the TASK BASE in order to advice necessary decomposition. This is done outside an ordinary lifecycle of the community – in the phase of initialisation. For this, the PPA agent uses an encapsulated database Project Architecturing Agent that parses data stored in the format of SQL Anywhere and updates the agents bases. This can be done either through direct access or via KQML messaging where agents would perform the update by themselves.

The general functionality of the PPA agent is summarised as follows:

- set an appropriate knowledge base with respect to initial constraints and considered theory
- decompose the global ontology into agents TASK BASEs (problem sections)
- configure an acceptable solution in the form of a complete list of operations (jobs) to be carried out and a bill of material to be purchased,
- carry out preliminary planning and contract appropriate PMAs.

The examples of the general PPA treatment scripts are shown in Table 6.

TREATMENT SCRIPT: <i>evaluate</i> performative		
IF:	- content : <i>constraints, theory</i> - sender : <i>user</i> - other conditions : <i>nil</i>	THEN : <i>set_knowledge_base(constraints, theory) →</i> <i>configure → project,</i> <i>select_peers(cb),</i> <i>order_tasks</i> PRIORITY : <i>append_project(tb)</i> <i>Nil</i>
TREATMENT SCRIPT: <i>evaluate/achieve</i> performative		
IF:	- content : <i>project</i> - sender : <i>user</i> - other conditions : <i>nil</i>	THEN : <i>get_project(tb) → peers, project_plan</i> PRIORITY : <i>Send performative: evaluate/achieve to: peers content: project_plan</i> <i>nil</i>
TREATMENT SCRIPT: <i>achieve</i> performative		
IF:	- content : <i>global ontology</i> - sender : <i>user</i> - other conditions : <i>nil</i>	THEN : <i>decompose(ontology) →</i> PRIORITY : <i>initialise_base(tb)*, append_plans(tb)*</i> <i>nil</i>
TREATMENT SCRIPT: <i>reply</i> performative		
IF:	- content : <i>task_state</i> - sender : <i>P</i> - other conditions : <i>M</i> <i>A</i> <i>Nil</i>	THEN : <i>compose_task(sb) → solution</i> PRIORITY : <i>Send performative: reply to: user content: solution</i> <i>nil</i>

Table 6

8.3 Meta-Agent

The meta-agent in ProPlanT is currently under development and it is expected to be an entity that extends the system in order to improve its global behaviour. It proved to be useful especially during run-time operation of a multi-agent system. Its role in planning has to be fully clarified, yet. So far, we have identified the following contributions:

- Discovering agent failures and providing instructions on how to continue in both the negotiation and/or manufacturing processes (the meta-agent can update arbitrary PMA's CO-ORDINATOR BASE by adding and/or removing some of its elements).
- Updating the PMA's state-base variables by exploring the information gained through the monitoring of the agents' load and message traffic. In this way, the so-called capability revision process of PMA's TASK BASE is evoked.

- Updating the PPA's and PMA's CO-OPERATOR BASEs with respect to recent changes among PAs and their subordinated machines.

The ProPlanT system is currently in the form of a well-running prototype, which satisfies the expectations of the end-users. It has brought a new quality of work into the field of the production preparatory phases as similar complex coverage of all the planning and scheduling aspects was not available.

When trying to express efficiency of the whole system we were facing the huge complexity of the planning and scheduling tasks, which cannot be expressed in terms of just a few parameters and their relationships. It is necessary to consider that the designed, highly distributed solution in ProPlanT enables to granularize knowledge and to make it clearly understandable and maintainable. Knowledge and data can permanently propagate through the system in a very efficient way. Negotiation processes replace the complex tasks of planning and scheduling. The methodology of the Tri-Base model leads to decrease communication traffic and is more knowledge -based than that which explore the classical Contract-Net-Protocol strategy. There is no simple way to quantify these facts.

We have been measuring intensity of the message traffic in ProPlanT (number of messages needed to create a new plan). Utilisation of the Tri-Base model, properly designed knowledge bases of individual agents and good overall community structure results in approximately linear dependence between the number of agents and intensity of the communication traffic, in average. These results contrast with expected communication needs of the pure broadcasting strategy that tends to be quadratic.

When we speak about performance improvement (due to the Tri-Base dynamics and/or meta-agent activities), we mean improvement in the common-sense understanding.

The current results are promising, but they have to be more thoroughly verified in a benchmarking environment.

8.4 Some remarks on ProPlanT efficiency

The ProPlanT system is currently in the form of a well-running prototype, which satisfies the expectations of the end-users. It has brought a new quality of work into the field of the production preparatory phases as similar complex coverage of all the planning and scheduling aspects was not available.

When trying to express efficiency of the whole system we were facing the huge complexity of the planning and scheduling tasks, which cannot be expressed in terms of just a few parameters and their relationships. It is necessary to consider that the designed, highly distributed solution in ProPlanT enables to granularize knowledge and to make it clearly understandable and maintainable. Knowledge and data can permanently propagate through the system in a very efficient way. Negotiation processes replace the complex tasks of planning and scheduling. The methodology of the Tri-Base model leads to decrease communication traffic and is more knowledge -based than that which explore the classical Contract-Net-Protocol strategy. There is no simple way to quantify these facts.

We have been measuring intensity of the message traffic in ProPlanT (number of messages needed to create a new plan). Utilisation of the Tri-Base model, properly designed knowledge bases of individual agents and good overall community structure results in approximately linear dependence between the number of agents and intensity of the communication traffic, in average. These results contrast with expected communication needs of the pure broadcasting strategy that tends to be quadratic.

When we speak about performance improvement (due to the Tri-Base dynamics and/or meta-agent activities), we mean improvement in the common-sense understanding.

The current results are promising, but they have to be more thoroughly verified in a benchmarking environment.

9. Conclusions

The tri-base formalism for describing an agent's activity introduced in the paper seems to be a suitable vehicle for description the activities of multi-agent systems for production planning, especially in the case of the project oriented manufacturing.

The clear separation of the three types of bases enables a very simple maintenance and up-date of the available knowledge and data. It also enables much simpler integration of legacy knowledge/data structures of a diverse nature. There is one very interesting feature considered in the tri-base approach: the generated detailed plans are stored in a highly distributed way; their parts are stored in the autonomous agents. There is no central plan available. This fact enables minor changes in the "local" plans whenever it is appropriate without any change of the overall plan structure. Once created, the plans can be checked/up-dated with respect to the current situation.

The novel idea behind introducing the tri-base acquaintance model is further rooted in the separate activity of creating the TASK BASE planning quadruples in the *pre-planning phase*. In the classical twin-base model [Cao 96], the TASK BASE comprehends entire scope of possible decomposition activities. In the case of production planning we can hardly describe all the possible planning in terms of reasonable number of plans stored in a TASK BASE and consequently we rather maintain those which are currently the subject of interest. The *pre-planning* phase of the agent planning activity is responsible for creating the relevant plans, while periodical *planning* maintains them up-to-date. In our

system, the user is responsible for triggering the *pre-planning* phase of the planning lifecycle. Nevertheless, we propose a meta-agent, through its ability to monitor the community and present a certain kind of qualitative reasoning, to suggest an appropriate task to be considered for planning.

Certain amount of records found in the *plan section* of the TASK BASE, prepared plans, is what needs to be balanced in order to achieve proper behaviour and optimal decision making. Two marginal cases illustrate this issue. If there are all possible plans prepared in the *plan section*, we reduce the tri-base model into the simple twin-base model. On the other hand, an empty *plan section* reduces the model into a simple expert system with the *problem section* as its rule base.

The tri-base model structures knowledge of agents in MAS in a very natural way. Dynamics of the system and development of agents' bases during the life-cycle of the whole system poses many interesting questions which have to be solved taking into account domain knowledge about intended application domain.

One of such questions is closely related to division of responsibilities (or agent's specialisation) within the agent community. The more tasks (and agents) any agent is able to co-ordinate the more agents it has to be subscribed to. The result is growth of number of messages running through the system, which can finally fully paralyse its function. That is why the size of the CO-OPERATOR BASE of any agent has to be reasonable. To limit this size the designer of the system can take over some background knowledge from the application target domain. This was the case of ProPlanT System. On the other hand, search for some self-tuning mechanisms in the multi-agent system can be viewed as a challenging theoretical problem. One option is to try to shrink the number of the decompositions in the problem section of the agent's TASK BASE and then restrict the CO-OPERATOR BASE to the relevant agents only, i.e. to agents able to solve considered subtasks. Decompositions in the TASK BASE could be scaled for this purpose by some function expressing

their usefulness from the view of the system. (This can take into account, e.g. how often the particular decomposition was tried in pre-planning phase and accepted in the planning one).

Each particular application of the agent technology in a production planning task requires a detailed analysis of the production facility, information and material flows, company habits, software modules available etc. Three types of agents (namely PA, PMA and PPA) have been distinguished as a result of detailed analysis of production processes at TESLA-TV factory. From the methodological point of view, there is no problem to extend the family of agents and to include them into the agents' community if appropriate. Moreover, there are a couple of interesting features of the generic structures (like ontology, complaints etc.), that have not been considered in the particular TESLA-TV solution. Possible utilisation of these additional features can lead towards much more complicated inter-agent relationships.

AGENT	COOPERATOR BASE	STATE BASE	TASK BASE
PPA	$\{\langle PMA, \{T_i\} \rangle\}$	$\{\langle PMA, load, trust \rangle\}$	$\{\langle P, PPA, T=\{\langle T_{jk} \rangle\}, P, V \rangle\}$
PMA	$\{\langle PA, \{J_i\} \rangle; \langle PMA, \{T_i\} \rangle\}$	$\{\langle PMA/PA, load, trust \rangle\}$	$\{\langle T_i, \{J_j\}, C \rangle\}; \{\langle T_i, \{T_j\}, C \rangle\}$ $\{\langle T_i, PMA_i/PM_i, S=\{\langle J_n, PA_n \rangle\}, C, V \rangle\}$
PA	$\{\langle M, \{\langle O, time, cost \rangle\} \rangle\}$	$\{\langle M, load, schedule \rangle\}$	$\{\langle J_i, \{O_j\}, C \rangle\}$ $\{\langle J_i, PA_i, S=\{\langle O_j, M \rangle\}, C, V \rangle\}$

Table 7

The particular contents of the bases for the different types of agents (PA, PMA and PPA) are shown in a simplified way in . All of three instances of CO-OPERATOR BASEs contain semi-permanent knowledge needed for their activity. The STATE BASEs contain complementary actual data e.g. on load, costs, and subjective trust in activities of some other agents. Typically, the TASK BASE contains knowledge on how to solve the local planning problem (*problem section*), and the already developed plans (*plan section*). Only the *plan section* of the TASKS BASE are displayed in . Whereas the *problem sections* of the TASK BASEs direct the way to reach the main goal, i.e. to create the

production plan/schedule, the results of the planning process itself are stored in the *plan section* of the same base. The knowledge representation schemes used in both the *plan* and *task section* is the same, but those in the *plan section* represent instances of the more general records of the *problem section*.

Let's remark that the knowledge contained in the TASK BASEs enables dynamic, periodically invoked (re)scheduling which avoids negotiating in the Contract-Net-Protocol style. The role of PMAs in the scheduling process is very similar to that of task holons in [Sousa 99]; the PAs then work analogically to resource holons.

The ProPlanT is a modelling system for production planning. The physical execution of the plan is out of the scope of this planning activity. As might be expected, our agents are in a slightly different mode of operation during the plan execution. In this case, the similar tri-base formalisation can be applied, but the bases then get a different content. The CO-OPERATOR BASE contains quintuplets of already prepared plans, the STATE BASE information about current loads, costs, trusts, etc., and the TASK BASEs store information on the current status of the plan execution.

References

- [Cao 96] Cao W., Bian C.-G., Hartvigsen G.: *Cooperator-Base and Task-Base for Agent Modeling: The Virtual Secretary Approach*. In: Proc. of AAAI-96 Workshop on Agent Modeling, AAAI Press, 1996, pp. 105-111.
- [Cao 97] Cao W., Bian C.-G., Hartvigsen G.: *Achieving Efficient Cooperation in a Multi-Agent System: The Twin-Base Modelling*. In: Cooperative Information Agents (Kandzia P., Klusch M. eds.), LNAI 1202, Springer-Verlag, Heidelberg, 1997, pp. 210-221
- [Finin 95] Finin, T., Y. Labrou and James Mayfield. 1995. *KQML as an Agent Communication Language*. In: Software Agents (Bradshaw J. Ed.), MIT Press, Cambridge, Ma, USA.
- [Genesereth 92] Genesereth M.R. and Fikes R.E., *Knowledge Interchange Format Version 3.0, Reference Manual*, Report Logic 92-1, Computer Science Department, Stanford University, 1992.

- [Hazdra 96] Hazdra T.: *Agent Substitution in a Heterogeneous Multi-agent System*. Research report GL-02/96, The Gerstner Lab, CTU Prague, 1996.
- [Jennings 93] Jennings N.R., Wittig T.: *ARCHON: Theory and Practice*. In: Distributed AI: Theory and Praxis (Avouris M., Gasser L. eds.), Kluwer Academic Publishers, Dordrecht, 1993, pp.179-195
- [Kiss 96] Kiss G.: *Agent Dynamics*. In: Foundations of Distributed AI (O'Hare G. M. P., Jennings N. R. Eds.), John Wiley & Sons, 1996, pp. 247-267
- [Labrou 97] Labrou Y., Finin T. *A proposal for a new KQML Specification*, Internal Report TR CS-97-03, Computer Science and Electrical Engineering Department (CSEE), University of Maryland Baltimore County (UMBC), February 1997.
- [Lashkari 94] Lashkari Y., Metral M., Maes P.: *Collaborative Interface Agents*. In: Proc. of 12th National Conf. on AI, AAAI Press, 1994
- [Mařík 96a] Mařík V., Štěpánková O., Flek O.: *Multi-Agent System with Qualitative Simulation Unit*. In: Proc. of the XIIIth European Meeting on Cybernetics and Systems Research, Vienna, 1996, pp. 1138-1188
- [Mařík 96b] Mařík V., Kraus K., Flek O., Bezdíček J.: *Multi-Agent Decision-Making Architecture and Distributed Control*. In: Balanced Automation Systems'96, Chapman and Hall, London, 1996, pp.315-328
- [Mařík 98] Mařík V., Pěchouček M., Hazdra T., Štěpánková O.: *ProPlanT – Multi-Agent system for production planning*. Proc. of the IXth European Meeting on Cybernetics and Systems Research, Vienna, 1998, pp. 725-730
- [Nwana 97] Nwana H.S., Ndumu D.T.: An Introduction to Agent Technology. In: Software Agents and Softcomputing (Nwana H. S. and Azarmi N. eds.), LNAI 1198, Springer-Verlag, Heidelberg, 1997, pp. 3-26
- [Pěchouček 97] Pěchouček M., Lowe H., Bundy A.: *Proof Planning and Industrial Configuration*, Proceedings of the Fifth Practical Application of Prolog, London, April, 1997, pp. 233 – 241.
- [Sousa 99] Sousa P. Ramos C., Neves J.: Contracting Tasks between Autonomous Resources. In: Proceedings of the PAAM'99 Conference, London, UK, 1999, pp. 345-362
- [Štěpánková 97] Štěpánková O., Mařík V., Lažanský J.: *Role of Qualitative Reasoning in a Multi-agent System*. In: Proceedings of EUROCAST'97, LNCS 1131, Springer Verlag, Heidelberg, 1997.

- [Toomey 96] Toomey C., Mark W.,: *Satellite Image Dissemination via Software Agents*. IEEE Expert, Vol. 10 (1996), No. 5, pp. 44-50
- [Wooldridge 95] Wooldridge M., Jennings N.: *Intelligent Agents: Theory and Practice*. The Knowledge Engineering Review, 10 (1995), No.2, pp. 115-152
- [Wittig 92] Wittig T.(Ed.): *ARCHON: An Architecture for Multi-agent System*. Ellis Horwood, Chichester, 1992.
- [Zhong 97] Zhong N., Kakemoto Y., Ohsuga S.: *An Organized Society of Autonomous Discovery Agents*. In: Cooperative Information Agents (Kandzia P., Klusch M. eds.), LNAI 1202, Springer-Verlag, Heidelberg, 1997, pp.183-194