Agent Assistance in Planning[1]

De Causmaecker P. , Demeester P., De Pauw-Waterschoot Ph. and Vanden Berghe G.[2]
KaHo Sint-Lieven
Gebr. Desmetstraat 1
9000 Gent
Belgium

ABSTRACT

We describe how the use of agents in a lab planning application can improve the satisfaction of the users (in this case teachers) when a lab session has to be rescheduled. Regular lab planning software only takes into account the so-called hard constraints and sometimes proposes solutions that are not satisfactory to the teachers involved.

Keywords: agents, dynamic lab session planning, contract net protocol, blackboard mechanism.

1 INTRODUCTION

The goal of this paper is to show the usefulness of the intelligent agent's paradigm in the design and implementation of planning systems, more specific the planning of lab sessions in a polytechnic. Intelligent agents facilitate the formulation of the problem and also help in finding more satisfactory solutions for the parties involved. Agents are also tools that allow for the building of flexible and user-friendly software systems. Agents can represent the human users and defend their interests. They can be configured manually or built on experience to negotiate with other agents or users. In this way, sensitive information about the user's availability, willingness, feelings… need not be published at system level to be useful. The use of agents in a planning application can enable the system to use hidden and non-explicit information. The framework which has been developed will allow for agents to build a model of its user and of the system in which it is functioning. This model can evolve with time, and include former decisions and their evaluation. The model need not to be explicit, it can be represented by hidden variables and constraints, only to be interpreted by the agent himself.

2 DESCRIPTION OF THE SAMPLE PROBLEM

2.1 Problem statement

The example we have chosen to demonstrate the above-mentioned characteristics of agents is the dynamic planning of lab sessions in a polytechnic. It exhibits the possibilities of an agent approach in situations where negotiations between users are necessary. Anybody working at a university or a polytechnic has encountered the problem: the central scheduler carefully constructs a schedule for the lab sessions, however now and then during the academic year it happens that a session cannot take place (due to unforeseeable events such as illness of a teacher, unavailability of a lab room), and this session has to be rescheduled. The adaptive planning of lab sessions is a hard problem in the sense that it needs fast decision taking, involving many parties that cannot be contacted immediately. The central (human) scheduler knows from experience how (some of) the teachers will react on a rescheduling of the lab sessions and he can anticipate their reactions. He typically will select those possibilities with which he makes a chance of success when he starts negotiating with the teachers involved. When trying to automate this process one can incorporate agents in different ways.

2.2 Definitions of terms used in the example

To avoid confusion we first give our definition of the terms used in this article. Most of them are quite obvious, but some can have ambiguous meanings.

- **The head of department** is the person responsible for the daily operation of the department. One of his tasks is to make sure that the students can attend the lab sessions in optimal conditions.
- **The central scheduler** is the person responsible for constructing the lab session schedule: quite often this is done during the holidays.
- **A lab room** is a room where a lab session can be held. One lab room can hold a maximum number of students and has specific facilities.
- **A time slot** is a morning or an afternoon of a school day where one can hold a lab session.
- **A lab session** is a time slot with practical exercises on a specific subject.
- **A class** is a group of students who attend a lab session together.

3 AGENT ARCHITECTURE

3.1 A word about agents

It is not a simple task to explain the agent concept. There are as many definitions of what agents are as there are agent researchers. Most agent researchers agree that software agents (minimally) have the following properties:

- they act autonomously;
- they are reactive;
- they are pro-active;
- they run continuously.

In [1] there is comprehensive list of the properties that relate to agents: autonomous, interactive, adaptive, sociable, mobile, proxy, pro-active, intelligent, rational, unpredictable, temporally continuous, character, transparent and accountable, coordinative, cooperative, competitive, rugged and trustworthy. In the same green paper [1] the authors give a possible definition of a software agent: "a software agent is defined as an autonomous software entity that can interact with its environment".
Some researchers even discriminate between intelligent and truly intelligent agents when they have some of the above properties and some additional ones, like the possibility to learn,

[1] Part of a project supported by the IWT of the Flemish Government. Official title of the project: "Development of Object-Oriented Agents for Distributed Planning Systems."

[2] E-mail: {patdc, peterdm, phildp, greetvb}@kahosl.be

communicate through a natural language, … For a comparison between the different definitions of agents, see [2]. In this paper we use a notion of software agents as programs that act (or do something) on behalf of their human operators.

3.2 Types of agents

Most agent-oriented approaches focus on an interface agent: a simple agent with simple knowledge and problem solving capabilities, which has information filtering as primary task Centralised approaches like this have some shortcomings. Such a global agent quickly accumulates an enormous amount of knowledge needed to execute its tasks. It becomes a bottleneck, and supplementary tasks will require a lot of programming work, bypassing one real reason of existence of agents.
To overcome this problem researchers from the Carnegie Mellon University developed Retsina [3]. They introduce three types of agents: interface, task and information agents.

3.2.1 General definition

- **Interface agents** receive input from the user and display the results. These agents obtain and use the preferences of the user to represent him in the system.
- **Task agents** can execute a task: results are communicated with interface agents or with other task agents. Information that is needed can be found through an information agent.
- **Information agents** provide an intelligent access to sets of information. They assist in searching information. The information agents search information for interface agents as well as for task agents.
- We now introduce these agents in our problem domain.

3.2.2 Application of the general definition to our problem

- **Interface agents** are the closest to the user. They interact with the user. They need a model and use the preferences of the user to take decisions on his behalf in the system. These agents can be seen as the representatives of the user in the system. An example of an interface agent is an agent that represents the teacher in the system: only the agent knows the correct user preferences and tries to take them into account when negotiating about rescheduling a lab session.
- **Task agents** formulate plans and execute them. They have some knowledge about the task domain. For example, a task agent tries to find an available lab room to accommodate a lab session.
- **Information agents** give intelligent access to a heterogeneous set of information sources. They find information to respond to queries. For example, an information agent searches through the existing lab session planning.

We can conclude that an agent comes in three flavours: interface, task and information agents. These agents inherit from a super class Agent and can make other agents (see Figure 1).

3.3 Representation of actors by agents

In this sample problem, the actors are: the central scheduler, the head of department and the teaching staff. We provide each of them with an interface agent that can be configured, on the one hand to defend their personal interests as good as possible, on the other hand to make the system function better. When involved in decision-making, these agents first try to contact their actor and if this does not succeed they decide autonomously. When an actor is in front of his monitor his agent first tries to contact him with a pop-up window that appears on the actor's screen. The agent can ask the user to assist him in making a decision. If the actor however is not at his monitor since he has to teach a lab session for example his agent can decide autonomously. Autonomous decision-making is subject to configuration by the actor.

4 PROTOTYPE DEVELOPMENT

4.1 Used agent development software

The program is developed in CORRELATE [4], which is short for Concurrent object-ORiented REflective LAnguage TEchnology. It offers extensions of C++ and Java for distributed applications. It was developed at KU Leuven (Belgium) as an academic study object: it supports autonomous objects and a metalevel architecture. CORRELATE emphasizes a complete separation between the functional and the non-functional demands of an application. Functional demands describe what the application has to do to solve the problem. Non-functional demands are characteristics that an application has to have but that do not belong to the problem domain: these involve for example distribution, fault tolerance and security.
A CORRELATE application consists of a set of concurrent, interacting, active autonomous objects. The CORRELATE Object Model offers a logic view on a CORRELATE application and consists of five concepts:

- Active objects
- Autonomous behaviour
- Synchronisation
- Object interaction
- Object creation and destruction

4.2 Our goals

With this project we tried to reach the following goals:

- demonstrating the use of agent technology in a scheduling environment. Agents can be used to represent the teachers in a lab session schedule: these agents try to take into account the teacher's preferences as good as possible. To obtain the same result in a classical scheduling program you probably would have to introduce some strange parameters.
- developing a test case for CORRELATE. The CORRELATE environment itself is an academic (non-commercial) prototype and this project was a test case for

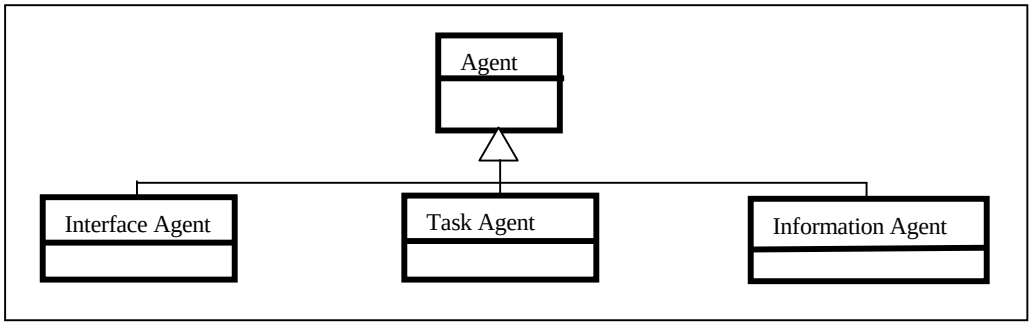

Figure 1: UML representation of an agency

the developed software to see if it would give good results in bigger multi-agent applications.
- showing that agents can be useful in software engineering. We also wanted to proof that agents provide a good model for larger software engineering projects.

5 DEVELOPED LAB SESSION PROGRAM

5.1 Example

Let us start with a situation from real life: teacher A, who is ill, phones on a Wednesday morning the department's head (DH) and tells him that he can not teach his lab session. This means that the lab session that this teacher was going to teach that afternoon has to be rescheduled to another (later) date. It is the responsibility of the DH to find a solution for this problem. The DH informs the central scheduler who collects some alternatives. This central scheduler works basically in two steps. First he searches for possible solutions to the problem, i.e. solutions that satisfy the hard constraints (see further). In a second phase, he filters out the solutions, of which he knows in advance (from years of experience) that they will not make a chance of success when proposed to the teachers involved. This means for example that, if the central scheduler has a possible solution that involves teacher A and a teacher B, but if he knows (from experience) that teacher B cannot stand teacher A, he can predict that teacher B will refuse to switch with teacher A and this solution will thus not be selected. The teachers evaluate the alternatives that concern them, and tell the central scheduler whether they want to co-operate or not. They will base their decision on essentially private grounds. A teacher C who is asked to switch his lab session of Friday morning with the lab session of teacher A on Wednesday afternoon can tell the central scheduler that this is impossible since he promised his wife to take care of the kids. So the interpersonal relations play an important role. It takes a rather experienced central scheduler to adjust the lab planning in such a case.

5.2 Program development

The lab session program that we worked out tries to take into account that teachers have their own preferences about the time slot they want to teach. For example there are not many people who want to teach on Friday afternoon or on Monday morning. Each teacher has his own agent (this is the so-called interface agent) who represents him in the system, only this agent knows about the preferences of his owner.

Since we have agents that have to communicate with other agents and users, we experimented with two kinds of negotiating protocols. In the first possibility we used the blackboard mechanism, this mechanism mimics what happens in real life, and as a second possibility we tried the Contract Net Protocol (abbreviated to CNP).

5.2.1 Blackboard mechanism

We can use the above example to show how the first version of the developed lab session program works. When the ill teacher warns the DH that he is unable to give his lab session, the central scheduler looks for a solution. He starts the search algorithm that only takes the hard constraints into account. This results in 'correct' solutions generated by the algorithm, but not all teachers involved in a particular solution are equally pleased. This is due to the fact that the solution doesn't take into account the soft constraints - the personal preferences of each teacher. Hence the two steps are: first the search algorithm generates some solutions and afterwards the teacher agents involved discuss about those solutions.

5.2.1.1 Hard and soft constraints

The hard constraints must be fulfilled. Our hard constraints are:
- One teacher can only be in one lab at the same time.
- In a lab only one lab session can take place at a time.
- A student can attend only one lab session at a time.
- Each lab room has a limited capacity.
- Lab rooms are in general not equipped to accommodate all kinds of lab sessions (we call this equivalence, E). Every lab session X has an ideal lab room Y where a lab session can be held in ideal circumstances without making any adjustments. Other lab rooms Z are totally incompatible to accommodate a lab session X. We visualised this in some kind of matrix (see figure 2), which we called the equivalence matrix. The rows represent the lab sessions and the columns represent the lab rooms of one department in our polytechnic. The couple (lab session, lab room) that is preferable is on the diagonal of the matrix. This is an ideal couple and no changes are necessary to accommodate the session in the lab room. Lab rooms which are totally unsuitable are indicated by a high cost (default is 1000).

The algorithm will impose these hard constraints.

Our soft constraints come in two classes.

5.2.1.1.1 SOFT CONSTRAINTS OF CLASS A

- Every teacher has his own preferences about the time slot he wants to teach a lab session.
- A teacher will not be equally prepared to concessions to all colleagues.

This first set of soft constraints has everything to do with the negotiation between the agents.

5.2.1.1.2 SOFT CONSTRAINTS OF CLASS B

- Of those lab rooms that are equipped some are more appropriate than others. Some lab rooms are appropriate to accommodate a session but it takes time or money to fit the room. A few lab sessions can be held in another lab room if an adaptation is made. In our example only the computer science sessions turned out to be interchangeable between rooms. This lab session is

session\room	B306	B115	G111	G005	G201	G012	G101	G106	C1T1	G209
Elo	0	1000	1000	1000	1000	1000	1000	1000	1000	1000
Eli	1000	0	1000	1000	1000	1000	1000	1000	1000	1000
Tm/Soup	1000	1000	0	1000	1000	1000	1000	1000	1000	1000
Meet	1000	1000	1000	0	1000	1000	1000	1000	1000	1000
Proc	1000	1000	1000	1000	0	1000	1000	1000	1000	1000
Uitv	1000	1000	1000	1000	1000	0	1000	1000	1000	1000
Cam	1000	1000	1000	1000	1000	1000	0	1000	1000	1000
Cad	1000	1000	1000	1000	1000	1000	40	0	1000	1000
Computer Science	1000	1000	1000	1000	60	1000	1000	20	0	1000
Mat	1000	1000	1000	1000	1000	1000	1000	1000	1000	0

Figure 2

normally held in the C1T1 lab room, but if some adaptations (for example: installing new software on the existing computers) are done, it can also be held in lab rooms G106 or G201 (see figure 2).

- The closer in time a lab session can be rescheduled to the original lab session, the better the solution will be (Time Constraint, Ti)
- Number of agents involved in a solution (indicated by A)

The second set of constraints concerns the infrastructure and organisation of a polytechnic.

The above-mentioned soft constraints (class A and class B) can be violated but if they are not, the quality of the system will be higher. This will be reflected in e.g. a higher satisfaction for the teachers (class A) or a more appropriate usage of infrastructure (class B). These soft constraints are implemented through a global cost function and negotiated over by the agents.

5.2.1.2 Search algorithm

This algorithm solves the hard constraints mentioned above and at the same time filters on excessive violation on part of the soft constraints. When the central scheduler starts the algorithm, a task agent is created who has the responsibility to find solutions for the problem. The algorithm this agent uses is based on an iterative deepening search. The algorithm recursively searches for alternatives for the session α to be displaced. It selects a room and a time slot, empty or not, satisfying the hard constraints for the teacher, the session and the students. If a session β has been scheduled in this same slot, thus violating the constraints on simultaneous occupation, the session β becomes the new session to be displaced. This process recourses *depth* times, depth being a parameter fixed in advance. When eventually the last level is reached, only free room-time slot combinations are considered.

With depth=3, this algorithm can generate displacements of the forms:

$$A \rightarrow B$$
$$A \rightarrow B \rightarrow C$$
$$A \rightarrow B \rightarrow C \rightarrow D$$

Majuscules A,B,C and D represent room-time slot combinations. The arrow $A \rightarrow B$ means that the session taking place in A is moved to B. The last room-time slot combination in a chain must of course be empty previous to the displacement.

The algorithm takes into account all hard constraints, especially the session - lab room compatibility and the time constraints.

The solutions generated by the algorithm are those where the sum of the above costs is less than a certain upper limit (we have chosen the ad hoc value 1000 as an upper limit) and from this set we choose the 10 best solutions, again in terms of the costs mentioned above.

5.2.1.3 Used Co-ordination technique

The solutions generated with respect to the hard constraints by the search algorithm (which was started by the central scheduler) are then put on the blackboard [5]. The teacher agents involved in a solution evaluate this plan and return a cost on the basis of the preferences of their operators (time slot and colleague, the so-called class A soft constraints) to the DH agent. If the DH is not physically present then his agent chooses the solution with the lowest cost.

The introduction of a co-ordination mechanism gives our agents room for intelligent applications of their users preferences. The agents can negotiate about the plans, which are generated by the search algorithm. To make this negotiation possible we introduce the term *money*. We use money as a metaphor for the amount of resources available to a teacher. Each teacher has a so-called "bank account", which in fact is a measure for the resources that the user has requested in the past and the assistance he has given to the other teachers.

As mentioned in the introduction we differentiate between two cases: a lab room that is unavailable and a teacher who is unavailable.

5.2.1.3.1 Principle

We use the following principle: "he who causes trouble has to pay for it": the teacher, who is unavailable (for whatever reason: illness, holiday, attending a conference…) pays the costs to reschedule a session. Initially every teacher gets an amount of money on his bank account that is proportional to the number of hours he teaches. Someone who gives a lot of lab sessions will more likely be unable to do all the scheduled lab sessions than someone who only teaches one session a week.

There is one exception to this rule: the case where a lab room is unavailable. This problem has to be solved by the DH, who has however no money. When he has to solve a problem (in this case: find a suitable lab room) the other teachers have to pay for it. This can be seen as a kind of uniform tax that the teachers have to pay to the DH, because they can use services offered by him.

5.2.1.3.2 Case 1: a lab is unavailable

This problem is for the DH to solve, but all teachers pay the costs. The DH starts the search algorithm, which generates a few plans (solutions) involving some teachers. These teachers compute their cost to participate in a proposed plan and inform the DH agent. The agent chooses the plan with the lowest cost. This cost is then divided by the total number of teachers and this amount is charged from the account of each teacher. After this has happened each agent *i*, who has co-operated in the accepted plan, gets an amount C_{T_i}. It is possible that a certain agent is first charged money and later credited money, if the agent is involved in the accepted plan.

5.2.1.3.3 Case 2: a teacher is unavailable

The unavailable teacher pays all the costs. He asks the help of the DH agent to solve this problem. The latter one starts the search algorithm, puts the plans on the blackboard, every involved teacher computes the cost to join the plan, and the DH agent takes the plan with minimal cost. The DH agent informs the agent of the unavailable teacher the *total* cost of the plan and this agent pays the DH agent. The DH agent then pays every teacher agent involved in the accepted plan. This is done to increase the privacy of each agent. Only the DH agent can deduce the preferences of each individual agent.

5.2.1.3.4 Remark

Teachers who are willing to co-operate to solve a problem see their bank account increase slowly while teachers who refuse to co-operate see their account decrease slowly. However, if their bank account becomes too low they are not able to make changes at the lab session planning, because they can not pay the other agents anymore. In the real world, this corresponds to a teacher who asks a lot of favours from his colleagues but refuses to co-operate when they ask him a favour. After a while his colleagues will refuse to do something for him.

5.2.1.3.5 Types of (class A soft constraints) costs

To evaluate how much a co-operation to a plan costs, a teacher computes the sum of the following three (soft constraints) costs:

- **Reason dependent cost (C_R)**: proportional to the reason a cost is defined, for example: the cost for being ill is 10 and the cost for taking a holiday is 30, this is a higher cost than the former because normally a teacher only goes on holiday when he does not have to teach.

- **Cost to change a session (C_C)**: this is the cost to swap a session with a colleague. This cost is computed with respect to the preferences of each teacher.
- **Cost to move a session (C_M)**: this is the cost to move a lab session to either an empty slot in the planning (this means that the students, the room and the teacher are all free) or to a slot that is already taken, this slot on its turn has to be moved. This cost is also computed with respect to the preferences of each teacher.
- **Cost to assist a specific teacher (C_{TC})**: this is an extra cost a teacher asks to co-operate with a specific colleague, the higher the cost, the more this teacher dislikes his colleague.

5.2.1.3.6 USED COST FUNCTION

With the help of the above types of (class A and class B) costs derived from the hard as well as the soft constraints, we can compute the total cost for a teacher agent (C_T) who is asked to co-operate to a plan:

$$C_T = C_C + C_M + C_{TC} + C_E + C_R + C_{TI}$$

The total cost of one solution (plan) is:

$$C_{Tot} = C_A + \sum_{i=1}^{n} C_{T_i} ,$$

with n the total number of agents.

The co-ordination here is rather primitive, every agent involved puts his cost to join a plan on the blackboard and the agent of the DH chooses the plan with the lowest cost.

5.2.2 Contract Net Protocol

The Contract Net Protocol is a classic coordination technique which is used for task and resource allocation between agents [5]. In this protocol, agents can play two different roles:

- A **manager** divides a problem in sub problems and searches for contractors who can execute these problems, and keeps also an eye on the global solution;
- A **contractor** executes a sub task. A contractor however can also become a manager who divides the subtask and contract it out to other contractors.

A manager can find a contractor through a bid process that goes like this:

- a task is announced by a manager;
- the contractors evaluate the task with respect to their own possibilities and commitments;
- the contractors make a bid that they send to the manager;
- the manager evaluates the received bids, chooses a contractor and gives him the contract;
- the manager awaits the result of the contract.

5.2.2.1 Negotiation between agents

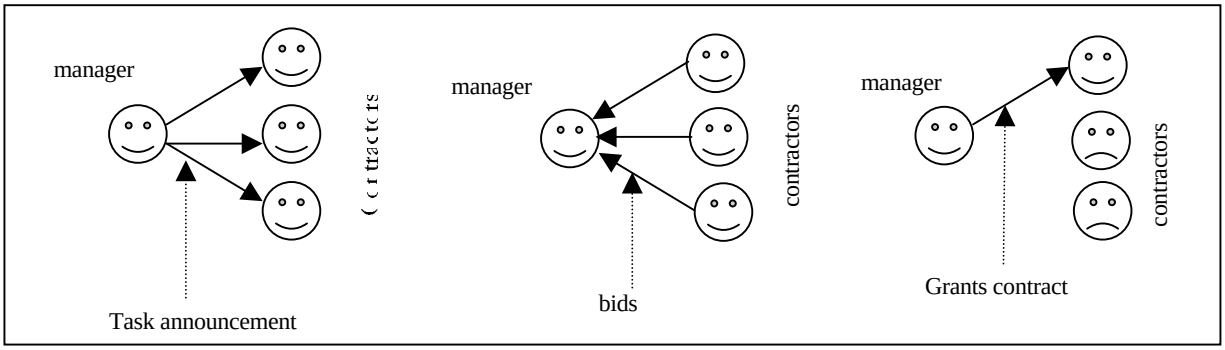

Figure 3

5.2.2.1.1 SHORT TERM BEHAVIOUR: NEGOTIATION CYCLE

Our agents have two possible conditions: the asking and the listening condition. An unsatisfied agent goes into the asking condition; this means that this agent will try to find a solution to improve its situation. Agents that are pleased enter a listening condition: these agents are satisfied with the proposed solution.
Agents that are in the listening condition are possible candidate-contractors. An unsatisfied agent becomes a manager and tries to find contractors to improve its situation. After announcing a task the manager gets bids from the contractors (these are agents that are satisfied). It is possible that when the manager grants a contract to a contractor that this previous satisfied agent (the contractor) becomes unsatisfied. This agent goes into the asking condition and becomes in this way on his turn a manager.

5.2.2.1.2 LONG TERM BEHAVIOUR: SYMPATHY

An agent in the listening condition will, while evaluating a question, not only take into account the change of cost that the move of a session will bring but also the sympathy he feels for the agent that started the negotiation. We introduced sympathy to optimise the long term behaviour of the agents. Agents remember who helped them in the past. The following example will make things clear: Agent A asks agent B to move or switch his lab session. Some time ago however agent A did agent B a favour by moving his own lab session to a bad time slot just to help out agent B. Agent B will remember this and will easier allow the change.
Up to now the function we used consisted of the cost (see above). From this cost we subtract the 'sympathy':

$$price = cost - sympathy$$

Every agent must remember how its relationship with the other agents is. This happens through the use of sympathy points. If the number of sympathy points of agent A with respect to agent B is positive then agent A knows that it has to do agent B a favour, and agent B knows also that it can expect a favour from agent A, since the sympathy points of agent B with respect to agent A will be negative. Total sympathy is in other words a conserved quantity in the system, which we arbitrarily fix to zero.
When the program runs for the first time the sympathy for every agent is zero. When the manager announces a task, the contractors begin to bid. This bid process takes 10 rounds in our implementation. After each bid round the manager informs every contractor (except the contractor who did the best bid in that particular round) that there is a contractor who offers a solution for a lower price. They have the opportunity to make a better offer. To do this, they need to lower their price, but since their cost is fixed they can only play with the sympathy they are offering to the manager. The contractor who has, the best bid after ten rounds, receives the contract. At the same time the sympathy matrix is updated with the new data. The original

sympathy that the manager felt for the (winning) contractor is increased by the number of sympathy points that the contractor showed for the manager and the original sympathy that the contractor felt for the manager is decreased by the number of sympathy points that the contractor showed for the manager.

5.2.2.1.3 START CONDITION OF THE NEGOTIATION

The CNP is engaged whenever a lab session of a certain teacher (for example represented by agent A) cannot take place. Agent A is in this case the manager and is in the asking condition. The other (pleased) agents are in the listening condition and can make

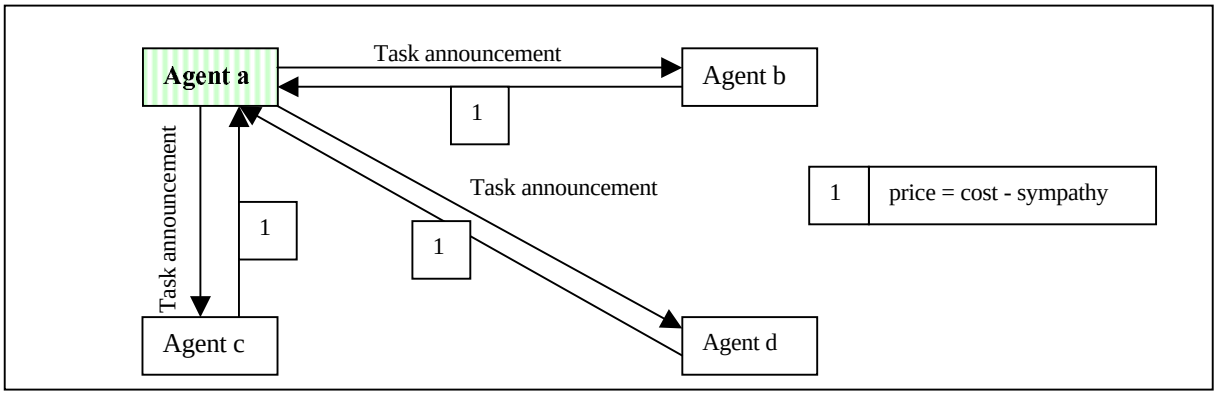

Figure 4

a bid on Agent A's task announcement. This bid depends on the cost and also on the sympathy that an agent feels (or does not feel) for agent A (see Figure 4). The bids that are sent to the manager by the involved agents are actually possible solutions for the problem

6 CONCLUSION AND FUTURE PLANS

We implemented a prototype of the two above possibilities and used an existing database as a starting point. The database consists of all the teachers that give lab sessions in one particular department of our polytechnic. We started from an existing lab session plan and each time a session could not take place, we tried to solve the problem, keeping in mind that each teacher had his own peculiar preferences regarding a particular time slot in the plan. Both implementations lead to the same solutions, but we remarked that the CNP was generally faster than the blackboard principle.

In the near future we plan a test day where the human central scheduler and our prototype will 'compete' against each other. The real test lies however in the fact that we will have to persuade the teachers to trust their interface agents.

The model showed that this kind of software development can be based on agents. As a test case, we actually applied our agent model to a dynamic scheduling problem in transportation. The analysis turned out to be a straightforward job, strengthening our belief in the model.

In a new project called COALA (http://coala.tsx.org), which started in November 1999, we will develop agents that learn. The goal of the project is to develop agents that can model their environment and their users. To bring this to a good end, the agents will need to have learning capabilities. An interface agent for example has to learn what the preferences of the teacher it represents are. In the current program the teacher explicitly needs to inform the interface agent about his teaching preferences. The DH agent could also learn the preferences of the teachers, so that he can make predictions about who will co-operate and who will not.

References

1. Agent Technology, Green Paper. Agent Working Group, OMG Document ec/2000-03-01, version 0.91.
2. Is it an Agent, or just a Program?:A Taxonomy for Autonomous Agents (http://www.msci.memphis.edu/~franklin/AgentProg.html). Stan Franklin and Art Graesser. Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.
3. Multi-agent Integration of Information Gathering and Decision Support. Katia Sycara and Dajun Zeng. ECAI 96. 12th European Conference on Artificial Intelligence.
4. Language Technology and meta-level architectures for distributed objects (http://www.cs.kuleuven.ac.be/~xenoops/CORRELATE/PUBLICATIONS/phd.ps.gz). Bert Robben. PhD thesis. 1999.
5. Issues in Multiagent Design Systems. Susan E. Lander, Blackboard Technology. IEEE Expert, March-April 1997, p18-26.
6. Coordination in Software Agents Systems. Hyancinth Nwana, Lyndon Lee & Nick Jennings. BT Technology Journal, 14(4), 1996.