in Advanced Robotics the International Journal of The Robotics Society of Japan, Tokyo

A MULTI-AGENT COOPERATIVE SYSTEM FOR AN ASSEMBLY ROBOTICS CELL

Eugénio Oliveira
Carlos Ramos
Faculdade de Engenharia Universidade do Porto
Rua dos Bragas, 4099 PORTO codex
PORTUGAL
email: eco@fe.up.pt
Fax: +351 2 319 280

ABSTRACT

A Multi-Agent System Architecture is here justified and described for the sake of its application to an Assembly Robotics Testbed. Cooperative policies between distributed Agents, leading to either Negotiation or Client/Server protocols, as well as appropriate knowledge have been studied, classified and implemented. This specific architecture as well as knowledge are then responsible for sophisticated behavior both in terms of inter-Agent Cooperation and Agent local control. Several scenaria where different cooperative and conflict situations arise, are presented in the framework of a real Testbed, working under real-time constraints, which has been already implemented and tested at our Laboratory.

We may summarize our contribution , stressed in this paper, as the description of a real system, including Distributed and Cooperative Agents devoted to an Assembly task under real-time constraints.

1 - INTRODUCTION

1.1 MOTIVATION

Intelligent Robotics is one of those domains where hard problems have to be tackled by means of both symbolic knowledge and complex computations, hence making the Distributed Artificial Intelligence approach attractive, if not mandatory.

Multi-Agent Systems (MAS) architecture - upon which ARCHON[1] philosophy has been based - may be seen as providing Open System characteristics, mainly in which flexibility as well as reliability concerns, therefore enabling continuity of operation while the environment is dynamically changing. A good discussion on Distributed Artificial Intelligence foundations and its relationship with Open Information Systems can be found in [1].

Modeling real world, sophisticated application domains through Multi-Agent Systems, presents some other important advantages like:

* providing a more natural representation of the real world domain expertise, which is often itself distributed;
* Breaking down both the problem and the problem-solving strategy into smaller units which makes the search for the solution process more simple and understandable.

A society of Agents avoiding centralized control has also to solve the problem of the relationships between, on one hand, each Agent's behavior and goals and, on the other hand, those of the global System or Multi-Agent community. In order to make the all activity to converge to the global system satisfaction, despite of the Agents degree of autonomy, special policies are to be designed for:

* Agents actions **Coordination**
* Agents **Communication**
* Agents **Cooperation**

Coordination can be dealt with either through (more or less distributed or centralized) planning algorithms, or through the designing of flexible policies and protocols to dynamically direct all the needed agents interactions.

Communication is a technical problem depending on the available operating mechanisms for agents processes interaction. Following the Archon approach our Agents are provided with a Session Layer which makes Unix message passing useful for different kinds of possible Agents interaction policies. Nevertheless, in our opinion, MAS technology and formalisms still lack of a complete framework for information, as well as knowledge and intentions, communication between different, and heterogeneous, entities (Agents, objects, processes) which take into account correctness, efficiency as well as reliability.

Cooperation is described by a set of actions and intentions which deals with the social and cognitive aspects of benevolent Agent interactions. Nevertheless, even discarding non-benevolent Agents, one has to take into consideration the all set of possibilities agents may

---

[1] ESPRIT Project "Architecture for Cooperative on-line Systems". Partners: Atlas Elektronik, Ambar, CERN, CNRG/Athens, EATL, Framentec-Cognitec, Iberdrola, Labein, Queen Mary W. College, Univ. Amsterdam, Univ. Porto, Volmac

encounter during their joint activity, when looking for the satisfaction of the system global goal. Dynamic dependency, negotiation, resolution of conflictual situations at different levels (on task assignment, on resources utilization and on different perspectives upon partial solutions) have to be modeled and represented in order to give the kind of social behavior they need in order to participate on complex team problem solving.

Work we have done in the framework of the Archon project, have contributed to design methodologies as well as strategies responsible for Multi-Agent Systems activity in cooperative domains. We here illustrate the Cooperative intelligent Systems approach in the special domain of Robotics for parts assembly and manipulation. We also would like to emphasize that, the development and use of a priority management mechanism for the robotics testbed Agents, can be seen as a contribution for the sophistication of MAS architecture and interaction policies and, on the other hand, as a breakthrough in the direction of its application to a broader scope of different real-time domains.

1.2 STRUCTURE OF THE DOCUMENT

This document reports the designing of a Multi-Agent System, including advanced features like cooperation and conflict resolution, controlling a flexible robotics cell, we have set up at the University of Porto as a testbed for Archon methodologies (UPART - University of Porto Assembly Robotics Testbed).

The first chapter introduces the motivation for our research in the framework of Archon, as well as this section on the document structure. The second chapter tries to explain what are the characteristics of the problem we have chosen, and why can it be seen as representative of a class of good examples for MAS architecture design. Moreover, the expertise involved in solving parts assembly and manipulation by a flexible robotics cell is also presented.

Chapter three introduces our approach to the MAS architecture as well as the Knowledge for the Agents themselves. We also emphasize the use of the Blackboard for intra (and not inter) agent communication, the enhancement of cooperative policies by the use of real priority management and very simple conflict resolution strategies.

UPART was a good motivation for triggering our approach to Agents conflict resolution as well as negotiation protocols including priority management mechanism as it is explained in chapter 4.

The practical example we have implemented at our Laboratory, is fully presented in Chapter 5. Descriptions of the involved Agents, as well as different scenaria where both cooperation and conflict resolution are dealt with, are described and illustrated by means of several "screen dumpings".

The last chapter is a short conclusion about the relevance of the research as well as experiments that have been done using the robotics testbed UPART. The document concludes with a section on references which have been considered as important for our work.

2 - THE PROBLEM

2.1 CHARACTERIZATION AND DESCRIPTION

Flexibility is the key word in Intelligent Robotics. Limitations on the capabilities of current industrial robots need to be overcome in order to increase the number of application domains, mainly in which parts manipulation and assembly concerns.

Three main points are to be stressed here in order to emphasize the real benefits of the introduction of AI concepts, methodologies and techniques in the robotics field:

* Symbolic Planning is an important feature responsible for any robotics system to exhibit an intelligent behavior whenever dealing with non completely structured environments;

* Differentiated knowledge expertise and skills of very different nature are required in order to solve non trivial robotics tasks;

* Sensors feedback interpretation and fusion is mandatory for real (not simulated) environments.

Therefore, three main consequences come out of the needs pointed out above:

A- Symbolic Knowledge Representation and Problem Solving Reasoning Mechanisms are needed to control a robotics cell.

B- Time tagged information and actions, ought to be manipulated in real environments by appropriated time sensitive algorithms which include processes priority management.

C- Cooperation between different skills will increase the overall system capabilities to solve complex tasks.

The three main principles A, B and C pointed out on section 2.1 are the basis for the general methodology guide-lines we have followed in the implementation of a testbed for Intelligent control of assembly robotics.

2.1.1 General Description

UPART, the Robotics Testbed that has been developed at the University of Porto aims to be suited for assembly of units made up of different parts where millimetric precision is not required. Its main characteristic is to be flexible, therefore being able to perform several different assembly tasks with the minimum amount of reprogramming.

The layout for our testbed is composed of a simple table, 50 cm high, on the top of which some objects have been placed in order to be manipulated by a 5 and 1/2 degrees of freedom robot arm.

The robot arm, equipped with proximity sensors, is being controlled by a PC/AT running a robot level programming language called LM. Compiled programs are then downloaded into the original control box, equipped with a 8086 microprocessor which is responsible for the low-level control of the robot motors.

Moreover, a Camera+Laser image acquisition process was developed in order to gather numeric and geometric data from each current layout.

Although all the computers are linked through ethernet, it is worth to notice that Computer Vision as well as Action control are based upon PC/ATs while all other programs run under UNIX on SUN Workstations.

2.1.2 Relevance of the Robotics Testbed

What we claim this testbed has been suited for, is the useful illustration of a Multi-Agent System architecture controlling a robotics cell providing facilities concurring for the solution of assembly robotics problems involving the following aspects:

- Flexible, not completely anticipated, interaction (possibly including both cooperation and conflict resolution) between different sub-systems (semi-autonomous heterogeneous agents) of a Multi-Agent Distributed System.
- Efficient execution of different, but simple, manipulation tasks described by means of a friendly command language.
- automatic planning, execution and monitoring of each one of several different tasks by the same global system

Fig.1 UP Robotics Testbed

Task Level planning and execution, as well as cooperation, are being illustrated through the robotics testbed for each one of the following classes of tasks:

T1: - to locate different parts and to assemble them.

T2: - to identify different objects as well as their locations and, as a consequence, to update current world description

T3: - to automatically build up a descriptive model of an unknown object.

T1 e T2 are our main concern while T3 will be an optional robotics cell capability.

This Testbed also poses two more important problems:

- the hard problem of real-time task execution, leading to the need of dealing with time constraints, deadlines and task priorities management.
- the truly heterogeneous nature of several Intelligent Systems regarding:
  a) their different capabilities
  b) their different implementation platforms, ranging from PCs to Unix workstations.

2.2 THE EXPERTISE

Selecting a MAS architecture to deal with the problem described implies the identification of the needed expertise for solving the proper goals. Which Knowledge has to be available to the system to enable the joint activity of agents towards the completely automatic assembly and manipulation of different objects? We can identify, at least, the following types:

- Knowledge for **interpretation**
- Knowledge for **decision**

- Knowledge for **action**

Let us postpone description of another kind of Knowledge - Knowledge for coordination to be included in chapter 3.

Scene interpretation asks for Knowledge on sensorial information acquisition, representation and classification. Other functional characteristics of objects related with their possible manipulation, like stable states and grasping positions are also mandatory whenever the task involves physical changes in the world.

Decision making usually follows the interpretation step. In the selected framework, the decision capability is neither a diagnostic nor a simple order but on the contrary, is a more or less complex plan that has to be followed in order to coherently, and efficiently, change the world.

The knowledge involved in this process includes world representations at each step of action and including both initial and final situation. At the core of this knowledge for action are also planning algorithms which take into account all kinds of possible operations for the applications, motion interactions and restrictions.

Our robotics testbed is a good example of the effects of the MAS architecture on the action upon a restricted world. Actions much more efficiently performed, in a variety of situations, are the visible result of a set of primary decisions on Systems organization and architecture.

We summarize this section reminding that in order to be able to solve assembly and manipulation tasks, a robotics testbed has to include at least the following types of expertise: Visual and distance information interpretation, World and Objects symbolic representation, Planning capabilities, at different levels, for action and, finally, Knowledge on Agents Coordination.

3 - A MULTI-AGENT SYSTEM ARCHITECTURE

3.1 AGENT ARCHITECTURE

The following Figure shows our Agents Architecture. The domain related part of the Agent, called the Intelligent System (IS), may either be an Expert System, a Planner or even a more procedural program. What makes it to be seen as an Agent is the layer on top of it which is responsible for cooperation and, therefore can be called the Cooperation Layer. A more detailed discussion about Agent architecture for Archon has been discussed in [2], [3].

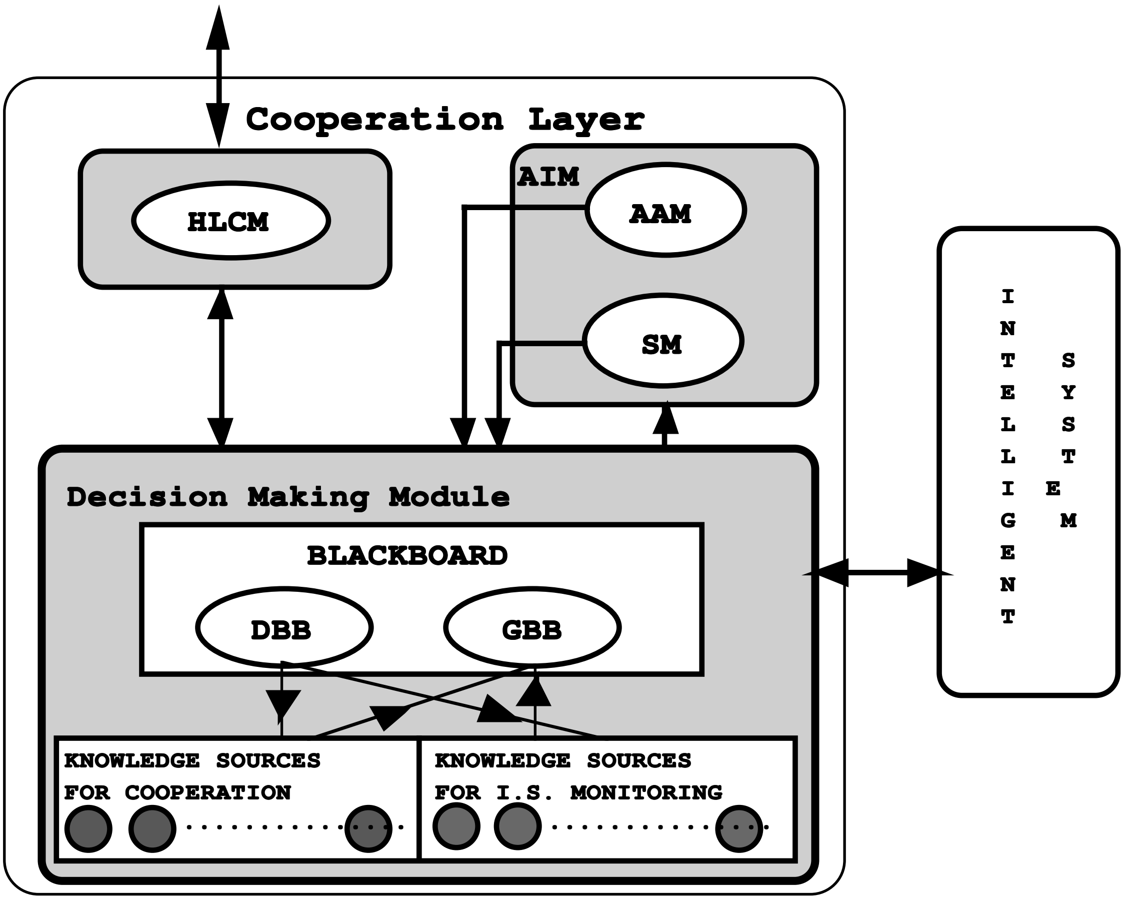

Fig.2 ARCHON Agent Blackboard implementation

In the Cooperation Layer we can distinguish several modules:

- High Level **Communication** Module (HLCM) responsible for the communication through Unix message passing and capable of sending (broadcasting, multi-casting, point to point) as well as receiving messages from other Agents;

- Agent **Acquaintances** Module (AAM), where all needed information about other Agents which are of interest to this particular one, resides;

- **Self** Model (SM), embedding an abstraction of the own Agent capabilities, its skills and interests;

- In the framework of this implementation, the **Decision Making** Module, which can be considered as the heart of the Agent, is blackboard based, more precisely, it uses the GoalBlackboard/DataBlackboard approach for intra (and not inter) Agent communication. Knowledge that Decision Making Module uses either for Cooperation and IS Monitoring, is represented by means of Knowledge Sources (KSs), set of rules that have been described elsewhere [4].

In this section 3.1 we refer features of the Cooperation Layer which play a major role in our testbed implementation. Cooperation Layer is now providing these facilities in order to make cooperation possible in which the Assembly Robotics environment is concerned.

When and how to cooperate is an issue of great importance for an Archon agent. Each cooperative agent should be able to know when it is necessary to cooperate with the others, and this rises the problem of deciding **when to cooperate**. **How to cooperate**, i.e., what cooperation strategy to use in each particular situation, is another problem at a different level. In this paper, we try to discuss these problems by distinguishing the two different roles agents may play in cooperation. Starting from this point, we then construct a framework, based on the Blackboard paradigm, to support the agents on detecting the needs for cooperation and dynamically choosing the cooperation form.

3.1.1 Decision Making Module Structure

UPART Archon-like Agents incorporate a Blackboard-based Decision Making Module both for Cooperation and local control. The rational behind this approach, the use of one uniform mechanism to support these two kinds of reasoning-- cooperation decision and local task scheduling -- is supported by the fact that they are indeed strongly related.

The structure of the decision maker, is based on a GBB/DBB mechanism where the Data Blackboard (Data-BB) is the place to put the messages coming from both the other agents through HLCM and the IS through an interface called the Convergence Layer (CL).

The role the Decision Maker plays in our framework determines the contents of the Goal Blackboard. Two kinds of control decisions have to be made: decisions on interacting with the others (e.g.. how to decompose the task, etc.) and the ones for local task scheduling (e.g.. what task the IS should do now, etc.). Therefore, Goal-BB encompasses goals to make these two kinds of decisions. In order to choose the appropriate goal, all the goals on the Goal-BB are attached with a state flag and dynamic changing priority. A goal in "wait" state can not be selected for execution since all conditions for that goal are not yet satisfied. Only the goal in "ready" state with the highest priority will be chosen and, in this case, the state is changed to "executing".

3.1.2 Knowledge for BB Decision Maker

As a reasoning module, the decision maker has a relatively simple control mechanism: the three-step loop: Generate Goal, Select Goal and Execute Goal. It is widely recognized that control maybe a kind of planning. Agents in the multi-agent framework cannot fully know and anticipate others Agents behavior and, therefore, cannot completely determine its own "long-term" plan. Under such circumstances, planning for control must be engaged in a cycle of plan generation, execution, monitoring and replanning. Clearly, execution of a plan is achieved at the Execute Goal step. Whereas the other three actions -- plan generation, monitoring and replanning -- are all implemented through the mapping from Data-BB to Goal-BB at Generate Goal step.

The key problem then turns to be how to map the incoming messages into the changes on the GoalBB. Intuitively, these changes are made by the set of Knowledge Sources (KSs) linking the two blackboards at the Generate Goal step. Triggering one of them may cause changes on the GoalBB like: the generation of a new goal, or the state change of a goal, or the change of a goal's priority.

At the Select Goal step, the Decision Maker chooses the most appropriate goal in terms of goal's priority and state. This chosen goal will then, at the Execute Goal step, trigger the corresponding KS to execute it. A notable point is that to execute goal, here, refers to control HLCM or underlying IS to do something, since the Decision Maker is only a control mechanism. These three steps obviously represent three different kinds of actions related with decision making: intentions generation, situation assessment and execution of intention.

Other kinds of Knowledge, needed for the Agents to be committed to an effective social behavior, reside in both the SM and AAM.

<u>Agent's Self-Model</u>

The following information in the SM are of great importance for our application :

- Goals and Tasks an IS is able to execute
- Lists with needed inputs for each one of the above referred goals
- Plans to achieve each one of the IS goals.
- Name of the Intelligent System itself
- Identification of its HLCM queues
- Dynamic information about the presence or absence of needed inputs
- Dynamic information about its own current activity.

Acquaintances Model

The following information in AAM are also important for our application:

- Links between needed inputs for tasks and who is able to provide them
- List of Agents which are interested in results of final and partial goals of the IS
- Dynamic information about other agents current status
- Other Agents Agendas

Types of messages

Finally, for the complete characterization of the agents interaction, we may classify messages exchanged among Agents as being of the following types:

- requests for information ;
- (partial or final) results from the agent problem solving activities (requested or unrequested);
- result acknowledgments;
- protocol messages for cooperation and conflict resolution.

3.2 COOPERATIVE POLICIES

3.2.1Two sets of Knowledge Sources

Decision Maker has two different sets of Knowledge Sources (KSs) corresponding respectively to cooperation-decision-making and local-task-scheduling

Cooperation KSs have three kinds of abilities: **detection** of need for cooperation, **choice** of Cooperation Type and **execution** of chosen cooperation policy. This set of KSs is then the primary responsible for the agent's cooperation behavior. Another set of KSs is intended for the purpose of controlling the underlying IS and, therefore, it determines the agent's local problem solving behavior.

3.2.2 Supporting Cooperation

In the following, we will discuss how the Decision Maker deals with these two kinds of reasoning within the same uniform mechanism.

- Reasoning Based Detection:

Reasoning Based Detection is used by the cooperation organizer to detect the needs for cooperation through two KSs (task-KS and result-KS).

These two KSs help the organizer Agent to detect the need for cooperation and to choose a cooperation form. Whenever they are triggered, there is a need for cooperation and, meanwhile, the cooperation type is determined by the activation of the KS.

- Direct Triggering

From the respondent's point of view, the mechanism it uses to detect the cooperation needs is Direct Triggering. We have, until now, considered two different possibilities of establishing co-operation protocols between Agents : Either by negotiation (**Contract net**) or by a more tight relationship (**Client/Server**).

In order to use contract net, there is a *task-announcement KS* which is triggered by the task-announcement message sent by other Agents. Task-KS is also a KS that can be used by the respondent Agent to detect the need of cooperation and assigning a task to another using Client/Server relationship.

Knowledge Sources that are activated either by Task-announcements, requests or incoming data constitute the Direct Triggering mechanism which is used by the Agent playing the respondent role whenever detecting the need for co-operation.

- Choosing co-operation protocol

It is up to the organizer to decide how to choose possible respondents. Two KSs are needed: "for-task-sharing" KS and "for-result-sharing" KS.

"For-task-sharing" KS, triggered by the flag task-sharing(task), will consult AAM to see who can execute that specific task. If there is only one Agent who has this capability, then it generates a goal to initiate a Client/Server relationship. In the case that more than one is able to execute the task, then it generates a goal to initiate Contract Net.

The function of "for-result-sharing" KS is similar to the "for-task-sharing" KS, the only difference being that this KS will consult AAM to see who may be interested in the data rather than to see who can execute a task.

- Choosing organizer

To be able to choose between possible cooperation demands which one to answer, the respondent also needs some specific knowledge. In order to execute the chosen cooperation protocol and eventually establish the cooperation relation, each cooperation protocol must be complete. Therefore, in addition to the task-announcement KS, the following KSs are needed to establish a contract net:

• bid-KS triggered by a bid message, it evaluates bids and sends the appropriate award.
• award-KS triggered by award message, it is responsible by task decomposition and to send task announcement message.
• termination-KS triggered by termination message, terminates execution of the contract and all related subcontracts.

Also, in order to enhance system global coherence, information about other Agents agenda as well as other Agents current dependencies are represented in each Agent Acquaintance Models contributing to improve agents actions convergence to the overall goal.

3.2.3 Local Task Scheduling

We have identified four kinds of possibilities for task scheduling of the Intelligent System, other than normal task initialization:

• task swapping
• restart task
• redo task
• multi-task execution

Redoing an executed task or restarting a task under execution with more reliable data can guarantee that each agent's contribution to the global task is up-to-date. Task swapping gives the agent the capability to process a new task in due time. Multi-task execution can surely reduce the total amount of time the whole system need to finish the global task.

3.3 CONFLICT RESOLUTION

We will here briefly refer the purpose of some important Agents functionalities which are already available for the UP Robotics Testbed.

Negotiation for Cooperation [2] is a very important capability Agents shall exhibit, but Conflicts may appear exactly at the very core of the Cooperation process. Whenever an agent wants to formulate a request, in order to get an answer from other agents, it needs to establish a protocol with the possible respondents for that specific request. This protocol will enable the exchange of information between the agent interested in getting the answer and the possible respondents (which may have to execute an IS task to provide the answer to that request). During this protocol for message exchanging, the Organizer informs other agents about the constraints related to that request (mainly a deadline). The potential contracted agents will respond informing about the expected time needed for having that task executed and, probably, the expected quality of the answer (bid message). The Organizer will face then a Positive Conflict to be resolved. A selection of the appropriate agent has to be made.

Therefore, *Conflicts* are always possible to happen during a Cooperative Multi-Agent Community activity. Strategies for Conflict resolution have been studied by several other authors [5], [6]. A broader classification of Cooperative and Conflict situations is presented in [7]. Strictly speaking, conflicts may appear either during Task Sharing or Result Sharing operations and may also be classified either as positive or negative. Therefore we may recognize the following types of conflicts:

- <u>positive conflict in task sharing</u>: whenever several agents are able to execute a task required by the same organizer. This situation has to be tackled down by means of negotiation functionalities

- <u>negative conflict in task sharing</u>: whenever there are no agents willing (or able) to execute a task required by the organizer. For real time system we can extend the concept to the impossibility of giving the desired results in due time. This situation requires a new plan (if available) to be issued by the Organizer's Monitor in order to find out a new and feasible plan for that specific task

- <u>positive conflict in result sharing</u>: whenever several agents, which are trying to execute the same task, produce different but complementary results for the same request (or the same result with different credibility messages associated)

- <u>negative conflict in result sharing</u>: when several agents, who are performing the same task, produce either inconsistent or antagonistic results. Here, we are referring, in the later case, to different alternative answers to the same request (they can also be seen as different values for the same receiver's task input), or, in the former, to inconsistencies detected between different (but somewhat related) information sets which are inputs to that task. The capability to deal with these situations and resolving the conflicts rely mostly on the availability of a functionality to measure information quality.

3.3.1 Description

Whenever no Agent is able to satisfactorily respond to a request from another one, the latter must either accept the unsatisfactory answer (e.g. later than desired) or change plans for the Task it has in hands. This policy does not require too much complication. On the other hand, real conflicts are those which happen during Result Sharing. When several results for the same request are available there is a need for domain dependent knowledge specifically appropriated for the resolution of the conflict. This fact had led us to propose the assignment, for each type of conflictual information (results of Task T_j, a specific agent $\mathbf{Ag_{crj}}$ in charge of the resolution of that kind of Conflict. Therefore, whenever an Agent Ag_i has an information (possibly a result of one of its own tasks) for another one, and it knows there is a third one (possibly more) trying to answer the same request, it feels obliged to send its results to the agent $Ag_{crj.}$ This Ag_{crj} is specified in the AAM of Ag_i as an expert on resolving conflicts regarding that information Ag_i is about to produce.

It is up to agent Ag_{crj} to resolve the conflict and send the result to the Agent which had made the primitive request. Procedures for resolving specific conflicts have to be available at each agent Ag_{crj} level. This means that we recognize the need for two different levels of knowledge involved with Conflict Resolution: Generic "diplomatic" Knowledge and specific domain-dependent knowledge for each conflict in hands.

Notice that in our approach to conflict resolution, we are avoiding two main and important drawbacks. First, we are not centralizing conflict resolution tasks on just one super-expert (the "Diplomat") which would contribute for a bottleneck in the negotiation process, and second we do make use of the specific knowledge already available in each expert agent to help solving conflicts related with that particular expertise. Therefore, different conflict types ask for different agents as referees, exactly those who already are experts in the domain.

3.3.2 Rules

As it was referred before, in the Decision Making Module there are sets of rules (Knowledge Sources) which are activated when Conflicts between Agents have to be resolved. These Rules include generic Knowledge, hence are not application dependent.

There are Rules for sensing the conflict, others for communicating during the conflict resolution and still others for the resolver, Agent Ag_{cri}, to find a solution.

Reasoning about Conflict Resolution by means of Rules implies, as it was the case for Negotiation, the use of some lower level enabling functions. We just mention here two of them:

* **Conflict Resolution** for Result Sharing:

For each kind of information (mainly a result of a Task), there is one Agent, Ag_{cr}, expert on resolving conflicts for that specific Task, who is the one in charge of sending the final agreed result to the organizer (the Agent which had made the request).

Agents that are simultaneously working to answer the same request (either because they have been contracted or they are voluntarily willing to help) should send their results to Ag_{cr} using the message : answer(Ag_i, Req_j, $Answer_i$, AgDest, $MissingTime_j$).

On receiving several messages of this type, Agent Ag_{crj} will consult in its own Self Model the appropriate slot with the procedure to deal with that kind of conflict, which is based in domain dependent knowledge. Agent Ag_{crj} also knows that it has time M_j (Missing Time for the Deadline) to send final results to Destination (AgDest): answer(AgSender, Req, Answer, AgDest, MissingTime). Procedures to deal with different conflict types include :

calculating new credibility for the final result , merging several results and choosing the best alternative among several ones.

The final message has the following syntax: answer(AgSender, Req, FinalResult, AgDest, MissingTime), where FinalResult is expected to conciliate disparate answers for the same requested information.

* **Conflict Resolution on Different Inputs** for a specific Task:
This functionality has to be activated whenever the Organizer gets different inputs to the Task it wants to execute, and these different inputs (which are all needed) are inconsistent. This inconsistency may be detected when interrelated inputs do not belong to the same context (see example). In more complex situations the conflict detection should be of the Intelligent System responsibility.

It is through these functionalities as well as specific Knowledge Sources that UP Robotics Testbed Agents are able to resolve their own specific conflicts.

3.4 MANAGEMENT OF PRIORITIES

Temporal Deadlines are of great importance for those applications where the produced answers to a certain request are only useful if they are available until a specific point in time (real time constraints). However, deadlines may be used to improve the efficiency of the system performance paying more or less attention to specific tasks, varying their task priorities according to their relative importance or urgency. Therefore, to guarantee the accomplishment of a specific task according to a deadline, an Agent shall compute *Priorities* taking also as much as possible into account information about the current system situation. Although requests are formulated with specific temporal deadlines attached, and not priorities, it is the receiving agent job to compute the more appropriate priority value for task scheduling purposes.

3.4.1 Deadlines and Priorities

Up to now, the same attention has been done to either urgent or non-urgent tasks. This is not an intelligent decision namely because of the possibility we have to change process priorities under UNIX. Note that varying priorities, one may be able to force CPU to give more attention to more important (urgent) requests.

One may use UNIX capabilities to change task priority having in mind hard real-time systems. In hard real-time systems, a task fails if it cannot commit to its deadline. However, we will not take this radical perspective, since a task failure may imply the overall failure in the UP Robotics Testbed. If it is not expected that a deadline will be accomplished, the CPU will still give more attention to that specific task trying to execute it as fast as possible.

Typically the urgency of a request is measured taking into account its deadline and the current computer load which influences decisively the priority of the corresponding task.

<u>Operating System Priorities</u>

ARCHON agents run under UNIX. Therefore we will concentrate our attention on this specific Operating System. UNIX processes have attached priorities. Priorities have an integer value varying in the interval [-20,20]. Priority -20 corresponds to the highest attention of the CPU (largest time slice). On the other hand priority 20 corresponds to the lowest performance (shortest time slice). By default, process priority is equal to 0. In our case we will consider the interval [-20,0] for task priority variation. UPART runs as superuser to manage all priorities at operating system level.

Priorities as a function of Deadlines

In order to take into account deadlines, tasks must be requested with an associated deadline. Let us consider (RT,td) for representing a required task (RT) with a deadline (td). The deadline is an absolute time. The main question is "How to compute the priority as a function of both the task and the deadline?" (p = f(RT,td)) .

Let us consider the following time variables:

t_{wait} - waiting time for inputs (relative time)
t_{exec} - execution time (relative time)
t_{cur} - current time (absolute time)
t_d - deadline of a task (absolute time).

When a request is received, we want to guarantee that: $t_{cur} + t_{wait} + t_{exec} < t_d$

We know t_{cur} and t_d and our problem is to estimate t_{wait} and t_{exec} (note that the computation of the correct values of t_{wait} and t_{exec} are not possible at the time the request is received).

Fortunately, the Robotics Testbed allows some simplifications since we are able to use some values by default. In fact the task load usually depends on the number of parts to be assembled. Therefore, we have considered as known, minimum and maximum execution and waiting times per object.

Minimum and Maximum waiting and execution times for tasks, are easily computed multiplying those assumed values by the number of objects to be manipulated in the current task. We will obtain the following values: $t_{exec_{min}}$, $t_{exec_{max}}$, $t_{wait_{min}}$ and $t_{wait_{max}}$ which we have assumed to correspond to task priority -20 ($t_{exec_{min}}$ and $t_{wait_{min}}$) and task priority 0 ($t_{exec_{max}}$ and $t_{wait_{max}}$):

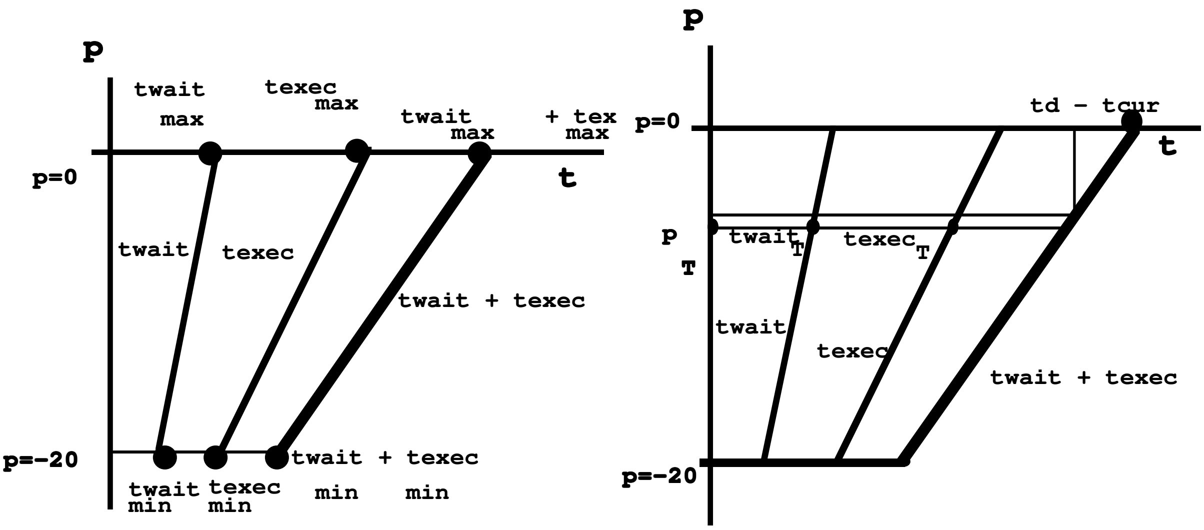

Fig.3 a) Task Priority as a function of Waiting and Execution time b) Calculated Priority

Figure 3a) above illustrates priority as a function of time (p=f(t)) considering a linear approach for waiting time t_{wait} and for execution time t_{exec}. It is also shown the sum t_{wait} +

t_{exec}. In this last line we consider that priority is varying from -20, for $t = t_{wait_{min}} + t_{exec_{min}}$, till 0 for $t = t_{wait_{max}} + t_{exec_{max}}$. The relation we want to achieve is:

$t_{cur} + t_{wait} + t_{exec} < t_d$ or, which is equivalent: $t_{wait} + t_{exec} < t_d - t_{cur}$.

We know, exactly, the right hand part of this relation and that the sum $t_{wait} + t_{exec}$ is the function represented on figure above. Thus, we are able to have an idea of how to compute priorities as a function of the deadline (td) and load (implicit in t_{wait} and t_{exec}).

Figure 3b) illustrates final task priority as a function of time. Looking at this last figure make us understand how priority is derived. The difference between the deadline and the current time is computed ($t_d - t_{cur}$). From the graphic we may extract $t_{wait} + t_{exec}$ such as: $t_{wait} + t_{exec} = t_d - t_{cur}$.

This calculation gives a real value for the priority. However priority must be an integer and the task is labeled with the nearest integer value, in the negative direction, of the calculated priority. Let us refer this value as p_T. Then it is possible to look again to the graphic and to get both values of t_{wait_T} and t_{exec_T}. For the lower side of the fgraphic, p_T is -20 and t_{exec_T} is $t_{exec_{min}}$ while in the upper side p_T is 0 and t_{exec_T} is $t_{exec_{max}}$. Therefore two important values are derived: p_T and $t_d - t_{exec_T}$

Priority P_T is the priority for which the task is pre-scheduled. The value of $t_d - t_{exec_T}$ is important since this is the deadline that the agent must impose to other agents which are in charge of supplying missing inputs for that task.

In conclusion we can say that the agent has received a request (RT,t_d) for a task with a deadline associated, has computed a previous value of the priority p_T for this task and, finally, has requested other agents for missing inputs ($RTmi,t_d - t_{exec_T}$) specifying appropriate deadlines.

These priorities may still be adjusted if waiting time becomes higher than expected. In Section 4.3 the use of these facilities will be illustrated in the framework of the UP Robotics Testbed.

The real importance of a task, its urgency, is related with the associated deadline. On the other hand, Priorities have to vary between the two limits ($t_{wait} + t_{exec}$) min and max. Through the graphics above we have derived the minimum priority (which means the less demanding for the computer) that still enables the system to meet the established deadline.

4 - ROBOTICS TESTBED

4.1 AGENTS DESCRIPTION

A more detailed description of the Agents involved in the UP Robotics Testbed has been made elsewhere [8], [9], [10]. In this section we will only briefly describe the main purpose of the 6 Intelligent sub-systems integrating the Testbed. The Agents are called (from their Intelligent System functionality):

TLP - Task Level Planner - a high level plan generator
ELP&TE - Execution Level Planner & Task Executor - controlling plan execution.
VISION - Object Recognition System based on Computer Vision
LASER - 3D Object Recognition System based on Range Finding Techniques

MODELS - Data Base representing object features
WD - World Descriptor - for extracting objects relationships and geometric constraints

A.1 - TLP

This Intelligent System is responsible for high level plan generation. TLP is an hierarchical and non-linear planner [9]. There are two levels of detail during plan formulation: goals level and operations level. Two plan generation policies are available: a fast generation of a plan and the generation of the best plan.

Plan generation is performed for the following task: *generate_high_level_plan*

A.2- ELP&TE

This Intelligent System is distributed by two machines. The high level part of ELP&TE, responsible for executing the plan and table space management, resides on a Workstation, while the other part of the Intelligent System, written in LM (a robot-level language) and responsible for assembly task execution, resides on a PC controlling the robot arm (a Renault-APRA) [10].

Execution Planning Module translates symbolically described operations, from the plan given by TLP, into a sequence of routine calls. This translation, which is situation dependent, generates code and a set of parameters that are sent out to the low level part of ELP&TE (Executor) [11]. Proximity sensors are also present making some exception handling possible.

The Intelligent System task for the assembly execution is: *execute_assembly_task*

A.3- VISION

VISION is an Intelligent System for object recognition in the world initial state. Some algorithms such as Blob Scan and Hough Transform are used to identify and locate objects of the scene. The results have the following format: position(ObjectName, StableState, X, Y, H, Orientation).

Usually the height, H, given by VISION is 0 since it is a two-dimensional recognition system. However, results sharing together with LASER will enable VISION to obtain a 3D based object recognition.

The task for object identification and location is the following: *get_object_locations*

A.4- LASER

LASER is another Intelligent System for object recognition at the initial state. Range Finding techniques are used for getting heights of objects. Three-dimensional object recognition from the combination of image and distance information's is another capability of this IS. Format of the results is the same announced before for the VISION Agent.

The main task LASER is able to execute, for object identification and location, is the following: *get_object_locations*

A.5- MODELS

MODELS is the simplest Intelligent System of all testbed IS's. It represents object features for different purposes. Objects need to be modeled for three main purposes:

- for more efficient object recognition (VISION, LASER)
- for automatic acquisition of symbolic relationships between objects as well as their geometric constraints (WD)
- for object manipulation (ELP&TE).

MODELS is able to perform two different tasks:
- *get_geometric_models* delivering object models to be used for the acquisition of symbolic relationships and geometric constraints
- *get_manipulation_models* delivering object models to be used for object manipulation and robot working area management.

A.6- WD

We have already seen that the output given by an object recognition system has the following format: position(Object,Stable_State,X_Coord, Y_Coord, H_Coord, Orientation).

It is easy to see that this predicate essentially contains numerical information. On the other hand Planning Systems are more interested in symbolic descriptions of the "world". The relationship "on(A,B)" is more important for Planning than the real coordinates of objects A and B. The capability of describing symbolic relationships among objects without any users help is the main reason for the inclusion of this Intelligent System [8]. Moreover, in order to avoid collisions between either the arm or the object being manipulated with other objects, WD can generate both symbolic and numeric geometric constraints to be taken into account by TLP and ELP&TE.

All the features of WD Intelligent System will be relevant for the execution of the following task: *get_symb_rel_geo_cons*

4.2 COOPERATIVE SITUATIONS

4.2.1 An Illustrative Scenario

This UPART scenario involves the six Agents referred in the previous section. Each Intelligent System has been enhanced with a Cooperation Layer, following the architecture presented in section 3.1 and, therefore, is considered as an Agent. From the Computer System point of view each Cooperation Layer is an independent process and, for each IS task, another different process is launched.

Figure below illustrates the flow of requests for a cooperative situation inside UPART. The system activity is started up by the USER requiring the execution of an objects assembly Task. Through the USER interface window, the operator specifies the final situation to be reached. USER Agent volunteers the final locations of objects to all interested agents (by consulting its AAM). USER's AAM also informs that the final task execution is of ELP&TE responsibility. Then, request1 (*execute_assembly_task*) is sent to ELP&TE.

ELP&TE receives the request and consults the Self-Model in order to know the needed inputs for "*execute_assembly_task*" which are: final object locations, initial object locations, object manipulation models, geometric constraints and the high level plan.

Only the former is already available since it has been volunteered by the USER. Requests need to be formulated for all other inputs. AAM is consulted in order to know which agents are able to give these inputs. Therefore the following requests are issued:

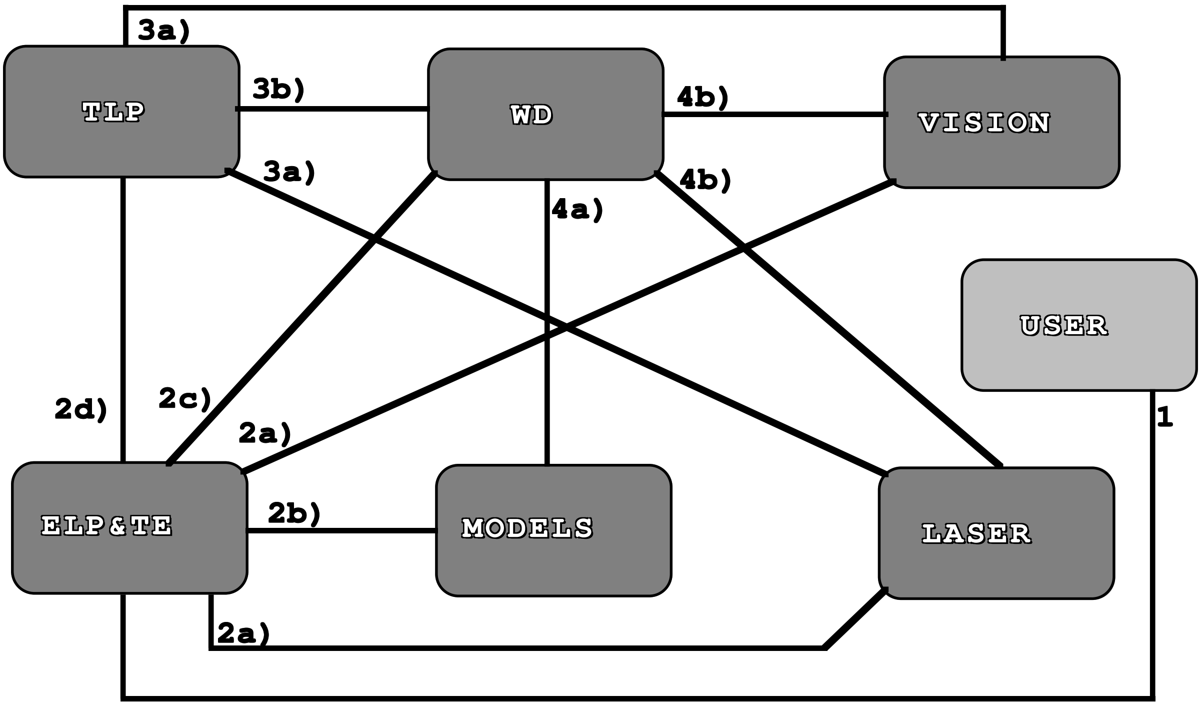

Fig.4 Beginning of Agents interaction: 1- execute_assembly_task; 2a)-get_object_locations; 2b)-get_manipulation_models; 2c)-get_symb_rel_geo_cons; 2d)-generate_high_level_plan; 3a)-get_object_locations; 3b)-get_symb_rel_geo_cons; 4a)-get_geometric_models; 4b)-get_object_locations.

- request 2a *get_object_locations* to VISION and LASER
- request 2b *get_manipulation_models* to MODELS
- request 2c *get_simb_rel_geo_cons* to WD
- request 2d *generate_high_level_plan* to TLP.

Therefore, all other agents of the Community got requests and start executing useful tasks. TLP receives the request to generate a plan and consults the Self-Model concluding that the following inputs are needed: "final object locations", "initial object locations", "symbolic relationships plus geometric constraints".

The former has been volunteered by the USER. AAM is then consulted to know who is able to produce the other two inputs. Therefore the following requests are sent:

- request 3a *get_object_locations* for VISION and LASER
- request 3b *get_simb_rel_geo_cons* for WD.

WD also needs to get objects symbolic relationships and geometric constraints. Consulting its Self-Model, WD concludes that the following inputs are needed: "final object locations", "initial object locations", "geometric models".

Again the former has been volunteered by the USER. Acquaintance Model is now consulted to know who is able to give the other two inputs. As a consequence, the following requests are sent:

- request 4a *get_geometric_models* for MODELS
- request 4b *get_object_locations* for VISION and LASER

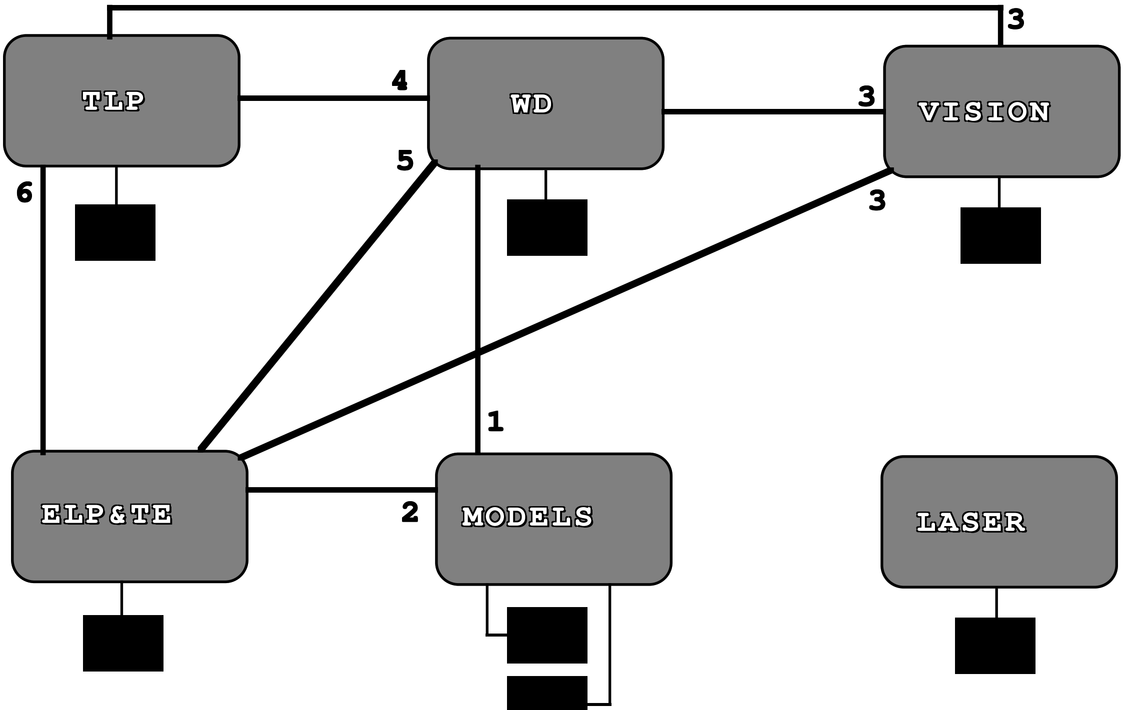

Fig.5 Agents controlling their own "Intelligent Systems": 1-geometric_models; 2-manipulation_models; 3-initial_object_locations; 4-symb_relations_geometric_constraints; 5-geometric_constraints; 6-high_level_plan

In this Figure, black boxes mean tasks launched by Agents. The only needed inputs for the tasks required to MODELS, VISION and LASER are final object positions which have been given by the USER. This means that the following four tasks can immediately be launched:

- *get_object_locations* by VISION
- *get_object_locations* by LASER
- *get_manipulation_models* by MODELS
- *get_geometric_models* by MODELS

These four tasks are simultaneously executed (note that MODELS supports Multi-tasking operation). Also the results of MODELS are sent out to interested Agents as follows:

- result1 *geometric_models* are sent to WD
- result2 *manipulation_models* are sent to ELP&TE

Conflicts may happen since both, VISION and LASER, are able to get object locations. Later, we will describe how this kind of conflict is dealt with in UPART. By now it is only important to know that LASER will send its results to VISION, despite having been requested by other agents, and VISION will solve potential conflicts between its own results and those of LASER. VISION is then in charge of sending out the following result:

- result3 *initial_object_locations* to WD, TLP and ELP&TE

WD has now all necessary inputs for launching its own task. After the end of this task WD will send the following results:

- result4 *symbolic_relationships_geometric_constraints* to TLP
- result5 *geometric_constraints* to ELP&TE

Now, TLP is able to start planning and a high level symbolic plan is generated. As a consequence it sends out:

- result 6 *high_level_plan* to ELP&TE

Finally, ELP&TE is able to translate the symbolic plan in a lower level set of actions, the result of this translation is then down loaded to the PC controlling the robot executing the corresponding task.

4.2.2 Cooperative Situations and corresponding actions in the UP Robotics Testbed

This sub-section will point out, systematically, situations where cooperation takes place in the framework of our Robotics Testbed and the respective actions carried out by the Agents Cooperation Layer. Here, {CSi} notation will represent the set of cooperative situations.

CS I:

Task Sharing - An agent requests other Agents in order to satisfy the execution of different sub-tasks (e.g. when LLE asks for the high level plan, initial object locations and grasping models).

CS II:

Client/Server Protocol - This policy is established whenever only one agent is able to execute the desired task (e.g. ELP establishes a client/server relation with TLP since only TLP is able to generate the high level plan).

CS III:

Contract Net Protocol - This policy is established whenever more than one agent is able to answer to a specific request (e.g. considering VISION and LASER agents, both are able to identify the objects locations required either by ELP, TLP or WD)

CS IV:

Result Sharing - It may take place between two or more agents trying to accomplish basically the same task but with different perspectives (e.g. during object identification task, VISION and LASER may exchange partial results in order to improve the final identification results accepted by both)

CS V:

Partial Results - Intelligent Systems are able to send partial results to the Cooperation Layer. The Agent then decides whether these partial results are to be sent to other agents or not. (e.g. TLP Intelligent System is able to give a partial plan whenever the total plan cannot be generated). Partial plans can be correct, incorrect or incomplete according the specific situation.

CS VI:

Reformulating of Previous Results - Intelligent Systems are able to reformulate, after some amount of processing, a previously sent result (e.g. VISION IS sends a partial result, but, after the application of a time consuming process, new, and better, results can be sent to VISION). It will be up to VISION Agent to decide which one of the given results to use.

CS VII:

Uncertainties and Merging of Results - A certainty factor is attached to each result. Whenever an Agent Cooperation Layer receives two different results, coming or not from the same agent, it must either to combine them, or to choose between them (e.g. VISION and LASER giving different results to be handled by a third Agent). Special attention should be given to the possible existence of conflicts between results.

CS VIII:

Error Dissemination - An Agent Cooperation Layer has to be able to detect whether an incorrect result has been sent, and probably used by the other agents.

CS IX:

Restart a task - An agent receives new inputs for an ongoing task. It must be able to restart the execution of the task whenever the specific situation leads to this need (e.g. TLP may receive new symbolic relationships describing the current world situation which implies that the plan generation task has to be restarted).

CS X:

Redo a Task - This case is similar to the previous one with the only difference that the specific task has already finished. In this case, the Archon Layer, should keep trace of the fact that results produced earlier by the finished task have already been sent to other agents and proceed accordingly.

CS XI:

Abort a Task - If the task is considered as being useless (e.g. the conversion of numerical objects locations data to symbolic objects relationships, whenever the objects locations have been changed).

CS XII:

Multi-tasking - Whenever more than one task is requested to an agent during a short interval of time (e.g. MODELS Agent is requested by ELP, WD and VISION agents in order to simultaneously perform different tasks. Therefore, Models Agent IS will launch several "simultaneous" processes.

CS XIII:

Priority Modification - Priorities of "ready" state tasks (for the scheduling process) are changed according to the dynamic situation (Agents task urgencies). Tasks in "running" state (when multi-tasking is possible) may also have their priorities changed for execution (e.g when MODELS Agent starts to execute a task which has been asked for ELP with low priority, VISION may also ask for the same task with a higher priority. This situation will imply an increase of the ongoing task priority).

CS XIV:

Temporal Deadlines - An Agent may be requested to accomplish a task during a specific time interval. Depending on IS capabilities, the Cooperation Layer may be responsible for the choice of the best task plan to be followed in order to meet the time constraints. This is what happens when TLP agent has a specific time limit to generate the plan, therefore it generates either the fastest plan or the better plan according to time availability).

Fig.6 Screen Dump of the Multi-Agent System

4.3 DEALING WITH PRIORITIES AND CONFLICT SITUATIONS

4.3.1 An example in the Robotics Testbed

Previous screen dump (Fig. 6) shows Cooperation Layer activities after completion of a cooperative situation similar to the one referred in section 4.2.1. Let us then look at that figure in order to understand what has really happened.

USER has asked the Multi-Agent System to "execute_assembly_task" until deadline t_d= 11:45:54 (see REQ field in the ELP&TE agent window). This request has been sent out at 11:42:24 (210 seconds for the overall community activity). The computation of the priority and expected execution time has followed the method described in the respective section. Therefore, this task has been scheduled with priority 0 (p_T=0 in the same window and field) and the expected execution time is 49 seconds for that priority (ED=t_{execT}=49).

As a consequence of this, deadlines for the arriving of missing inputs are computed and the appropriate requests are made to the other Agents.

Constraint on the size of this article forced us to abbreviate this scenario analysis, therefore we just draw the readers attention to the priority p_T=-7 derived at WD Agent, due to the fact that two other Agents [tlp, elpte], have sent the same request ("get_symb_rel_geo") shortening t from 11:45:5 to 11:44:23.

Also to notice is that Models got two different requests and it was able to assign two different priorities (0, -13) to them according to calculations based on deadlines.

To be brief, after "plan generation" has been executed by TLP Agent (p_T=-10), the Agent responsible for controlling the manipulator (ELP&TE) was also able to derive the final sequence of robot commands (task with p_T=0).

Figure 6 is then the practical illustration of how the overall system acts and reacts according to the dynamically changing situation. It shows how it computes priorities from deadlines, new deadlines for missing inputs as well as how it re-calculates priorities according to dynamically changing requirements.

4.3.2 Conflict Resolution

Another additional functionality we have already implemented in the UP Robotics Testbed is Conflict Resolution. Problems may happen when an agent is able to give more than one result or if more than one agent are able to answer differently to the same request. In fact, conflicts may happen involving results coming from different sources as it is the case with VISION and LASER Agents in our testbed.

In this example, conflict resolution means trying to deduce a common result from both VISION and LASER answers to the same request.

Usually, an agent sends out results as soon as they are made available by its IS. Meanwhile, Cooperation layer has been changed, since sending all results to the interested agents is dangerous whenever dealing with conflictual results. If this is the case, VISION will wait till the LASER's result are made available or till a deadline is reached. Note that LASER will deliver its output to VISION and not to all the agents interested in this result. Whenever both results are available, Conflict Resolution Knowledge Source is activated by VISION Decision Making Module trying to find out a common result which is then sent to other interested agents (WD, TLP and ELP&TE).

For this purpose we have included new knowledge in the following (self-explanatory) slots of Self-Models and Acquaintances of VISION and LASER Agents:

• i_solve_conflicts(get_object_locations, object_locations) - VISION Self-Model.
• conflicts_with_me(laser,get_object_locations) - VISION Acquaintances.
• may_have_conflicts(get_object_locations) - LASER Self-Model.
• who_solve_conflicts(get_object_locations,vision) - LASER Acquaintances.

Let us follow what happens. Whenever a request on "get_object_locations" arrives to VISION and LASER agents, their respective ISs try to get the appropriate answer to be send out to the interested agents (ELP&TE, TLP and WD). But, because VISION has been pointed out as the expert on resolving conflicts on "object_locations" it becomes responsible for issuing the final coherent result.

Fig. 7 Results from VISION and LASER Agents

Figure 7 illustrates the results of VISION and LASER to the same request, in the Conflict Resolution window. Each item (object position) has 6 arguments referring the name of the object, stable state, coordinates (X,Y,H) and orientation. The number at the right hand side of each item refers to a credibility measure that IS's gave to each result item.

To resolve the conflict, in this case, means to merge results from different sources. If they are similar, as it is the case for objects "block" and "ob1" (see figure above) the credibility is increased. If an object is identified by only one of the agents (e.g. "toothpick_case" identified by VISION and "ob2" identified by LASER) it will be also considered.

The worst case is that one in which the locations for an object are different. Usually this problem appears due to different accuracy of VISION and LASER in the Robotics Testbed. This is what happens for objects "base", "pepper_box" and "salt_box" (note that the results are not quite different, only small variations are found in coordinates or orientation). The solution for this kind of problem is to select results with higher credibility associated. Therefore, the location of "base" and "salt_box" are taken from VISION while the location of "pepper_box" is taken from LASER (see the same figure again).

Finally, figure 8 illustrates the final result in the Conflict Resolution window of VISION Agent which will be sent out to WD, TLP and ELP&TE for further processing.

Fig. 8 Conflict Resolution

As a final remark we want to stress that the involved procedures were independent of the specific domain of Robotics. Also what we had in mind with the example was to briefly illustrate all the protocol that takes place in the Multi-Agent community to detect a conflict and activate a resolution procedure.

5 - CONCLUSIONS

UPART, a Robotics Testbed we have implemented at University of Porto, gave us the possibility for experimenting "in situ" most of the ideas related with the architecture as well as cooperation strategies Archon consortium has developed.

The Multi-Agent structure, the granularity and degree of autonomy of Agents, their knowledge organization have proved to be suitable for the kind of problems we wanted to look at in the field of assembly robotics.

On the other hand, the Testbed itself, have induced some new problems derived from the fact that time was an important factor for the robot action. Inclusion of deadlines and priority management was also a contribution the experimentation in the framework of the robotics field gave to the coordination policies that Multi-Agent Systems have to take into account.

All the Agents have been implemented in our Laboratory and every functionality we described here can be demonstrated. Informal measurements concerning execution time to perform several tasks both with the Multi-Agent approach and the hierarchical one has been made leading us to the obvious conclusion that the Decentralized Multi-Agent approach is not only more natural but also more efficient than the traditional one.

ACKNOWLEDGMENTS

Acknowledgments are due to several members of the AI&Robotics Group of our Department for their help on getting the testbed implemented and many discussions that have contributed to the success of this work. Fernando Mouta, Paulo Lopes and Ana Paula Rocha should be especially mentioned. We also thank all partners of Archon, specially Thies Wittig and Nick Jennings, who have also contributed to the development of many ideas on cooperation as well as Agents architecture.

6 REFERENCES

[1] Les Gasser, "Social conceptions of knowledge and action: DAI foundations and open systems semantics", Artificial Intelligence 47, 1991

[2] Smith R., Davis R., "Framework for Cooperation in Distributed Problem Solving in Readings in Distributed Artificial Intelligence", Ed. A.Bond, Les Gasser, Morgan Kaufmann, 1988

[3] Lander S., Lesser V., Connell M., "Conflict Resolution strategies for Cooperating Expert Agents", in Cooperating Knowledge Based Systems 1990, Edited by S.M.Deen, Springer-Verlag

[4] Klein M., Baskin A., "A Computational Model for Conflict Resolution in Cooperative Design Systems", in Cooperating Knowledge Based Systems 1990, Edited by S.M.Deen, Springer-Verlag

[5] Jennings N., Oliveira E., Mouta F., Rocha A., F. Tuijnman,H. Affarmanesh,G. Wiedijk, "Design guidelines for full AL functionalities" Deliverable 410 of ARCHON Project, 1993

[6] Wittig T. (Editor), "Archon: Architecture for Cooperative Multi-Agent Systems", Ellis Horwood 1992

[7] Oliveira E., Mouta F., Ramos C., Rocha A., "Conflicts and Cooperation Analysis within", ARCHON UP/Tech. Note 14 of ARCHON Project, 1992

[8] Oliveira E., Mouta F., Rocha A., "Negotiation and Conflict resolution within a Community of Cooperative Agents", Int. Symp. on Autonomous Decentralized Agents, Kawasaki, 1993

[9] Ramos C., Oliveira E., "Planning, Execution and Sensor based reaction for Assembly Robotic Tasks", ECAI-92 - Wien (Austria)

[10] Ramos C., Oliveira E., "Closing the loop of task planning, action and sensing", IEEE/RSJ-Int. Conf. on Intelligent Robots and Systems, Raleigh (USA), 1992

[11] Ramos C., Oliveira E., "Sensor Based Reactive Planning and Execution for Assembly Tasks", IEEE- Int. Conf. on Systems Engineering, Kobe (Japan), 1992