COOPERATION IN A MULTI-AGENT COMMUNITY

Eugénio Oliveira, Fernando Mouta, Ana Paula Rocha
{eco, mouta, arocha}@fe.up.pt

Departamento de Engenharia Electrotécnica e de Computadores
Faculdade de Engenharia da Universidade do Porto
Rua dos Bragas, 4099 Porto Codex
Portugal

ABSTRACT

This paper describes a framework for cooperation suitable for being exploited in a society of cooperative distributed agents. We briefly present a tool UPShell - University of Porto Shell, which may be used to generate specific Intelligent Systems, although its main purpose is to build up a community of coarse-grain, semi-autonomous agents out either of those generated Intelligent Systems or of already pre-existent Knowledge Based Systems. After selecting the agents and the machines where each agent is going to run, UPShell sets up an architecture enabling agents to communicate and cooperate. Inside each agent, a blackboard-based architecture supports a flexible control and natural integration of different modules responsible for internal activity (problem solving) as well as external activity (cooperation and communication).

Key words: Distributed Artificial Intelligence, Multi-Agent Systems, Cooperating Expert Systems

Category: Techniques & Methods

1. Introduction

We have developed UPShell (University of Porto Shell), an environment to generate and exploit a community of cooperative Knowledge Based Systems, in the framework of the Esprit II Project (P-2256) ARCHON - Architecture For Cooperative Heterogeneous On-Line Systems. This project aims to construct a general purpose multi-agent system architecture for industrial applications.

The UPShell [1] may be used either in generation mode, by domain experts, to generate specific Intelligent Systems (IS) or, in consultation mode, to run a community of cooperative agents, enabling to deal with complex applications. To set up the community, special modules including communication and cooperation facilities as well as additional knowledge are automatically generated and integrated for the use of the cooperative community. As a consequence ISs are transformed into Cooperative Agents. In the consultation mode, the Shell provides the means for user interaction with either a separated agent or a set of agents (a society of agents) pursuing an overall common goal.

UPShell generates different Agents which run as separated Unix processes (not necessarily in the same machine), using sockets to communicate asynchronous among themselves. UPShell is implemented in Prolog using YAP compiler under Unix and also encompasses capabilities to use Xwindows and Unix interprocess communication facilities.

2. General Architecture

In the framework of Distributed Artificial Intelligence, we are interested in the cooperation of different intelligent systems to perform a complex overall goal. In order to cooperate, these systems are enhanced with special features concerning specific knowledge and reasoning capabilities for cooperation purposes. Each system builds up a model of itself and then based on the generated Self-Models, each system creates models of all its acquaintances (only about systems and items in which it is interested). In this way the systems become Agents of a cooperative community.

The Self-Model (SM) represents the Agent's capabilities and needs through the following structures:

- knows_about(Task) : tasks the agent can perform;
- interested_in(Task, Needed_data) : what are the Needed data for the execution of that task;
- skill_descriptor(Task, NeededInfo) : Task not decomposable in Subtasks;
- task_descriptor(Task, Plan, ExternalSubtaasks, NeededInfo) : the plan of the task, where plan is an ordered list of subtasks.

The Agent Acquaintance Model (AAM) represents explicitly only "interesting" knowledge about other agents through the structures:

- who_knows_about(Task, Agents) : which Agents can do that Task;

• who_interested_in(Data, Agents) : which Agents are interested in Data;
• has_supplied(Data, Agent) : what Data the Agent has already supplied.

The basic architecture of the Agents [2] consists of two conceptually separated subsystems (Fig. 1):

• An upper layer for cooperation and coordination of agent activity, the Cooperation Layer (CL);
• The underlying Intelligent System (IS) that has the domain specific problem solver capabilities.

It is the responsibility of the Cooperation Layer to coordinate inter and intra agent activities. This layer is further divided into:

• a high level communication module for information exchange with other agents (hlcm);
• a coordination and cooperation module for:
  - Coordinating internal activity,
  - Choosing cooperation policies and executing cooperation protocols according to the Agent workload and overall community situation,
  - Resolving Conflicts, and
  - Controlling the communication module;
• a monitor for scheduling IS problem solving activities.

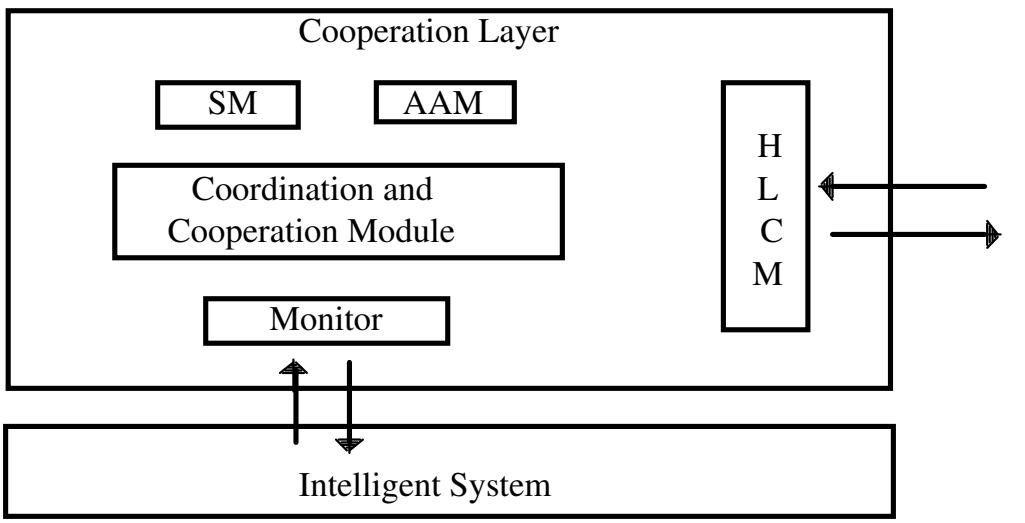

Fig. 1 - Internal Architecture of an Agent

The cooperation layer of each agent uses an internal Blackboard to access all information needed to make appropriate decisions about internal activity (problem solving) as well as external activity (cooperation and communication). Here, the blackboard paradigm is not used as a communication and cooperation framework among different agents, but only for internal purposes.

A special agent is the User Interface Agent (UIA), which integrates the user into the multi-agent system. We use User Models for the UIA to support the interaction between different users and the overall system. The User is seen as an underlying capability and component of the UIA. The User Model is reflected in the Self-Model of UIA and through the self-model in the Acquaintance Models of the other Agents of the community.

Each Agent runs as two separated Unix processes, one for the Cooperation Layer and another for the Intelligent System. They communicate asynchronous each other through sockets.

3. Internal Agent Activity

The Blackboard inside each Cooperation Layer is divided in two distinct parts: the Data-Blackboard (Data-BB) for messages arriving to the Cooperation Layer through "hlcm" and the Goal-Blackboard (Goal-BB) for intentions of goals the agent plan to perform.

Messages coming from the other agents through "hlcm" and from its own IS are put in the Data Blackboard (Data-BB) (except messages about other agent's state (busy/free) that are used to update the AAM).

Messages coming from IS can be
- results of a task (intermediate or final),
- requests for data not internally available.

Messages coming from the other agents can be
- task requests,
- task results,
- volunteered information,
- protocol messages (task_announcement, bid, accepted, rejected),
- abort messages,
- requests for explanations.

There are knowledge sources to deal with each type of message. Each knowledge source is composed of a trigger and a set of condition-action pairs. We will show two examples:

Task Request

For a task request coming from any other agent three condition-action pairs are considered:

1 - The task is under execution, in which case only some structures are updated (deadlines, priority, requesting agents).
2 - It is a new request and can be executed internally.
3 - It is a new request and to be solved need to be decomposed because for some subtasks it is necessary to ask for help outside the agent, or the result of some subtask may be of interest to any other agent in the community.

Volunteered Information

Unrequested data arriving at the Cooperation Layer is used to update a table of supplied information and trigger a knowledge source that looks for Tasks triggerable. All task_descriptors are scanned to see for which tasks the just arrived information may contribute. Then for each task found, it is checked if all information needed to perform that task is already available. Only the task or tasks, if any, that have all needed data present are selected to trigger or restart its execution. If the task is not running it is triggered to start execution, if it is already running, it is stopped and restarted with the new data.

Many of the actions executed by the knowledge sources applicable to the Data-BB, result in structures put in the Goal Blackboard (Goal-BB) expressing intentions of goals the agent plan to perform.

Messages arriving at the Data-BB reporting other agents activity and also its own IS activity are continuously triggering knowledge sources that modify the intentions of goals expressed in the Goal-BB. This is the main control mechanism that enables agents to adapt among themselves.

Goals in the Goal-BB have one of the following states:

- wait: if some data or subtask need to be asked outside.
- ready: if all input data for the task is available and it is possible to execute the task inside the agent.
- running: while the task is being executed.
- swapped: the task has already started execution but has been stopped and swapped by another more urgent task.

Messages sent by the Cooperation Layer to its own IS can be:

- task requests;
- answers to previous IS requests;
- task-swap requests (to stop the task that is running, to launch the new one, and when this has finished, resume the stopped one);
- stop a task (its execution is no more needed).

Messages sent by the Cooperation Layer through "hlcm" to (the Cooperation Layer of) any other agent are of the same type of messages that can be received from any other agent (listed above).

The IS structure is similar to a Rule+Frame based Expert System. Some sophistications were made in the rule interpreter in order to be able to attend the upper layer control directives and maintain several tasks simultaneously by means of indexing scheme for the tasks environments.

4. Knowledge Sources (KSs)

Knowledge Sources are sets of Rules of a partitioned Knowledge Base which are triggered by means of specific pre-conditions. Each Knowledge Source groups a number of Rules which may be selected under that pre-condition and fired according to their own conditional part.

Knowledge involved in them tends to be domain independent. However they make large use of pre-defined functionalities and data structures. They are grouped according to the high level functions they are responsible for:

- To select and implement a cooperation protocol [3]
- To support result sharing
- To help voluntarily another agent
- To monitor the Intelligent System (IS)

4.1. - **To select and implement a cooperation protocol**

4.1.1 - To choose cooperation protocol: according to Agent AAM and the specific information to be requested, the Agent chooses between one of two possible cooperation types.

Trigger: An Agent detects that it needs help on executing a specific task.

Rule 1:

IF more than one acquaintance can execute a sub-task
AND
there is no information on AAM saying other agent is
willing to execute it (agents_agenda(Agent,Task))
THEN
start to establish Contract Net protocol
send task_announcement(Taskj, Agi, Deaj)

Rule 2:

IF only one acquaintance can execute a sub-task
AND
there is no information on AAM saying it is
willing to execute it (agents_agenda(Agent,Task))
THEN
start to establish Client/Server protocol
send slave(Taskj, Ag, Deaj).

4.1.2 - Negotiation protocol: checks its own agenda in the SM and returns a bid message.

Trigger: Task Announcement message.

IF Calculated Priority for Taskj is Prj
AND

Expected Executing time of Taskj is Expj

THEN

send bid_message(Agent, Task, Bid) where Bid=Expj

4.1.3 - Negotiation protocol : to evaluate the returned bid messages and eventually choose the appropriated agents to whom to request the information.

Trigger: Bid message.

R1: IF number of sent task_announcement(Taskj, Agenti, Deaj) =
number of received bid-messages(Agk, Taskj, Bidk)
AND
Best Bid is chosen

THEN

send request (Agn,Taskj, Deaj) to the selected Agent.

4.2. - **To support result sharing**

4.2.1 - Result Sharing: Respondent's Point of View

Trigger: Answer to a request is available to be sent

Rule 1:

IF the Agent which has made the request is still interested in the response
AND
there is no other Agent working in the same task

THEN

send the requested information to the appropriate Receiver.

Rule 2:

IF there are other agents trying to deliver the same kind of results
AND
there is an agent Ag_{cr} which has the capability to resolve potential conflicts among Results of that Task

THEN

send the result to that agent Ag_{cr}.

Rule 3:

IF an agent Ag_{cr} has received (from different respondents) several similar answers for the same requested task
AND
the agent is able to manipulate credibility factors

THEN

compute the new value of the information credibility

Rule 4:

IF an agent Ag_{cr} has received (from other respondents) several different answers for the same requested task
AND
the agent is able to deal with inconsistency and complementarity of data (Taski different results)

THEN
try to find out a common result

Rule 5:

IF a message has returned back from the Organizer saying that there is an inconsistency between this result and another one the Organizer also needs as its own Task inputs

THEN
the Respondent should choose another plan for the same task or give up to produce another result.

4.2.2 - For Result Sharing: organizer's point of view

Trigger: requested answer(or volunteer information) which is an input for this Agents' task has arrived

Rule 1:

IF the input confirms an already available one

THEN
increase the credibility of that information (input)

Note: This implies the existence of comparison and credibility manipulation functions.

Rule 2:

IF different antagonistic results are available for the same required information by the organizer (Agi)

THEN
use the most credible result

Rule 3:

IF the organizer concludes, using its SM, that the received inputs for its own
task are inconsistent
OR
the IS can not perform the task and it detects inconsistency in the received
inputs

THEN

inform the respondents, about the inconsistency and give back those results

4.3. - **To help voluntarily another agent**

4 .3.1 - Voluntarily supply help

Trigger: message busy_free(BF, Agent1, Task)

IF Agent1 is free
AND
Agent1 can supply some data to Agent2
OR execute some relevant Tasks for Agent2
THEN
help Agent2 on doing it.

4.3.2 - Voluntarily supply help

Trigger: Data messages arriving (either volunteered or previously requested).

IF new incoming message together with the existing information form the complete set of inputs to task Tj
THEN
generate goal to execute task Tj

NOTE: This KS may cause the agent to execute a task voluntarily.

4.4. - **To monitor the Intelligent System (IS)**

4.4.1 - To redo a Task

Trigger: New incoming data message.

IF arriving message is an input for an already executed task
AND
its content is different and more credible than the previous one
AND
Task Tj may still be executed before Deadlinej
THEN
inform the Monitor to command IS to redo that task with new inputs.

4.4.2 - To restart a Task

Trigger: New incoming data message.

IF arriving message is an input for a task under execution
AND
its content is different and more credible than the previous one
AND
that Task may still be executed before Deadlinej
THEN
inform the Monitor to command IS to restart the task with the new inputs.

4.4.3 - To stop (or abort) a Task

Trigger: information from another Agent (by looking into its agents_agenda) or from itself (looking into its self_agenda) reporting on the current execution of a Task Tk

IF the outputs of task Tk are important to one of its own current tasks Tj
THEN
stop (or abort) current task Tj and wait for new inputs.

Note : A calculation on Deadlines should be done to prevent for task Tj not to be executed in due time.

5. Example

In this section, we will present examples involving a realistic Scenario conceptually proposed by Iberdrola, a Spanish company which is responsible for the management of large networks for distribution of electrical energy. This scenario relates with a situation involving several different agents:

- AAA - Alarm Analysis Agent
- BRS - Breakers and Relays Supervisor
- CSI - Control System Interface
- BAI - Black-out Area Identifier
- SRA - Service Restoration Agent

To diagnose a disturbance, AAA generates a list of elements possibly at fault (Hypothesis Generation Task) and then analyze one by one each hypothesis for validation (Hypothesis Validation Task). This last task is very time consuming and would benefit if it is possible to shorten the list of hypothesis. Another agent, BAI calculates the initial black out area (Initial Black out Area Task) based on the first set of alarms it receives from the CSI when there is a disturbance. If this information arrives to the AAA, before their Hypothesis Validation Task, this agent may filter the list of hypothesis generated, eliminating any element that does not belong to the initial black out area (it will perform another task - Hypothesis Refinement Task) hence, reducing the length of the hypothesis list. Hypothesis Validation Task will then be faster.

If the result of Initial Black out Area task only arrives when AAA is already executing the task Hypothesis Validation, this last task is stopped, task Hypothesis Refinement is executed and its result taken into account when restarting Hypothesis Validation task. To resume Hypothesis Validation is possible because each hypothesis is validated at a time (see Fig. 2).

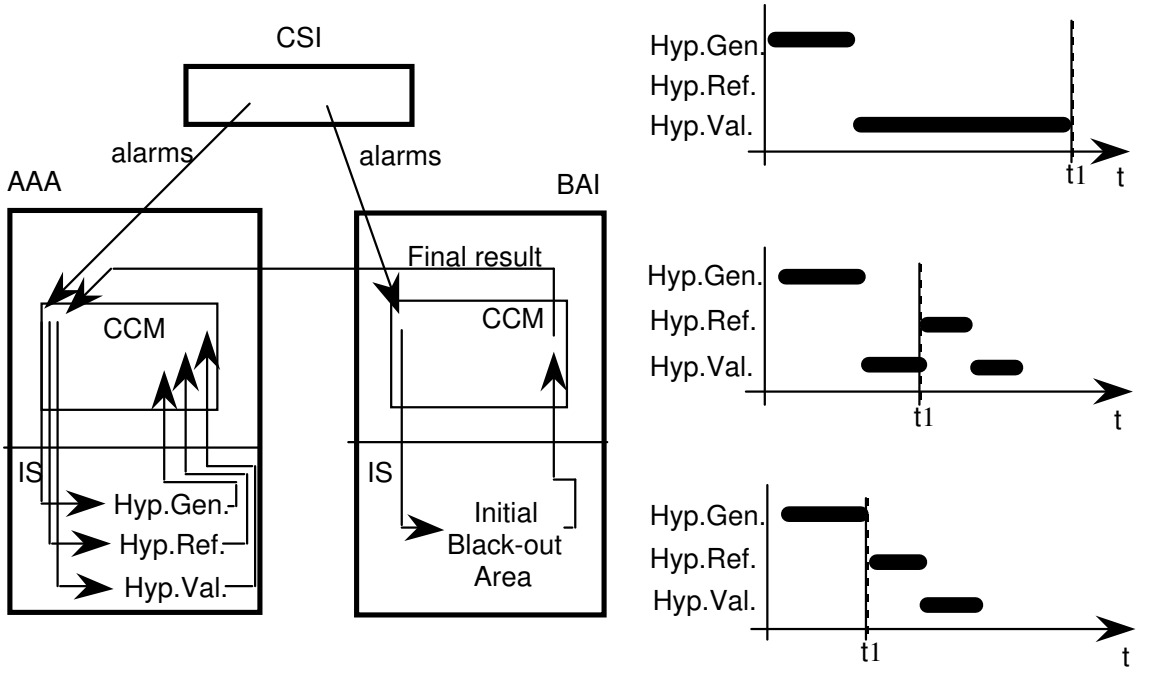

Fig. 2 - Temporal Diagrams of possible Cooperative Situations. At instant t1, the result of the Task "Initial Black out Area" arrives to Agent AAA

This kind of conflict occurs when it is arriving additional information that can improve the reliability of tasks which are either under execution or already executed.

When Hypothesis Refinement starts, mechanisms exist to detect that its result will be important as input to the task Hypothesis Validation and, therefore the latter will be stopped and only restarted when the former makes its results available.

6. Conclusions

An important characteristic of industrial control applications is the level of dynamicity they exhibit. Due to this non-deterministic nature it is difficult to specify how the system should behave in all cases in advance. It is desirable that the Agents apply general techniques at run-time [4].

We present sophisticated cooperation protocols for negotiation such as contract net where an Agent broadcasts a request to a group of acquaintances, the recipients then respond with a bid indicating the earliest time at which they can perform it and when all bids have been received the originator evaluates the most promising agent and places a contract with it.

Acknowledgements

This work is developed under the support of Esprit project P2256: Archon. We give our special thanks to all of the other Archon partners: Atlas Elektronik GMBH, Amber, CERN, CNRG-NTUA Athens, EA Technology, Framentec, FWI University of Amsterdam, Iberdrola, Iridia University Libre Bruxelles, JRC ISPRA, Labein, Volmac, Queen Mary and Westfield College.

We also acknowledge our colleague Rui Camacho Ferreira da Silva for his valuable contribution to the initial development of UPShell.

References

[1] - Eugénio Oliveira, Rui Camacho
"A Tool For Cooperating Expert Systems"
Proceedings of the 1st Expert Systems World Congress
Pergamon Press, 1991

[2] - ARCHON: Architecture for Cooperative Multi-Agent Systems
Editor: Thies Witig
Ellis Horwood 1992

[3] - Readings in Distributed Artificial Intelligence
Edited by Alan H. Bond and Les Gasser
Morgan Kaufmann Publishers, 1988

[4] - N. Jennings & A. Mamdani
"Using Joint Responsibility to Coordinate Collaborative Problem Solving in Dynamic Environments"
AAAI, San Jose, 1992