Transparent Distributed Problem Resolution
in the MAKILA Multi-Agent System

Karmelo Urzelai

Report LSI-96-43-R

22 JUL. 1996
Transparent Distributed Problem Resolution in the MAKILA Multi-Agent System

Karmelo Urzelai

Univ. Politècnica de Catalunya
Dpt. LSI, Pau Gargallo 5, FIB
08071 Barcelona - SPAIN
email: karmelo@lsi.upc.es

Topics: Distributed Problem Solving Tool, Transparent Resolution

Abstract

MAKILA is a model for problem solving in a society composed of cooperative agents. Its main objective is to offer an environment where the application implementer could focus on the domain knowledge, remaining transparent all the internal processes, as the goal resolution, and the communication and negotiation processes.

Problem solving is goal driven, both, internally to the agent, and at the social level. The contracting mechanism is used to solve problems that the agent cannot solve with its own domain knowledge.

This paper describes a flexible high-level tool for building Multi-Agent Systems for the MAKILA model. It focuses on the issue of transparency in knowledge definition, due to the separation of the declarative domain knowledge, from internal procedural knowledge about problem solving and social interactions. The main user interface's functions, with the information they require, are described.

In closing, a working example shows the tool use and the system behavior.
1. INTRODUCTION

Most Distributed AI systems that use negotiation [Davis & Smith 83] to cooperate are constructed from scratch. They usually have an ad hoc model, designed for the problem to be solved, which cannot be exported to other problems. This means that the knowledge defined to solve a subproblem cannot take part in the resolution of a different problem.

Despite this inconvenient, the cooperation paradigm based on negotiation is increasingly used in many applications.

Some tools have been developed to help the engineer on the definition and implementation of cooperative environments [Shekhar & Ramamoorthy 89]. Usually, these tools offer an environment where to develop the applications, they have facilities for the communications (i.e., predefined messages and a communication medium) [Finin 93], they allow parallel execution, ... But the knowledge engineer must specify where, when and what must be communicated, he manages the knowledge about the society, keeps track of the negotiations, and controls the resolution process. Also the experts of the domain areas must be aware of the distributed environment and take care of the messages to be interchanged, during the knowledge specification phase.

This is a hard handicap because the domain experts have the knowledge about their specialization area, but they do not know anything about D.A.I., cooperative systems, communication and negotiation.

The MAKILA model [Urzelai & Garijo 94] wants to go a further step, supplying the engineer an environment that allows him to focus in the definition of the domain knowledge, without being concerned about the internal agent functioning.

To achieve this objective the model of MAKILA has been defined from a different point of view. Commonly, the Cooperative Distributed Problem Solvers have been developed outside-inside; i.e., beginning with the definition of the requirements for the communications; followed by the interactions at the social level; and at last, the agent's task manger was constructed, with the restrictions generated in the previous stages.

In MAKILA, the model has been defined inside-outside. We started defining a generic resolution model. The model had to be enough general, flexible and independent, to be easily extended. Afterwards an internal agent's architecture was constructed, following the defined model, to solve the internal problems. Following the same general problem solving model, the social resolution strategy was defined, as an extension of the agent's internal inference mechanism. At last, the communication protocols were defined to fit the wished social behavior.

As a consequence, the resulting system is globally (in the local and social levels) more homogeneous, than other tools and environments, as ARCHON [Wittig 89], allowing to achieve what is the key and main goal of MAKILA: the transparency.

This transparency is achieved at diverse levels:
1.- Agent's internal resolution: the agents are independent generic goal resolution systems, thus the engineer does not need to define or control the agent's resolution engine.

2.- Distributed resolution: the engineer is not concerned about the assignation of subproblems to agents. The agent's communication manager, selects whom offer a contract; and the receiver's communication manager decides whether it can, and it wants to accept the contract or not.

3.- Knowledge definition: the experts only focus on the knowledge for problem resolution, independently of the distributed environment. There is not need of special coordination among the experts that define the knowledge of the different domains. Actually, the experts may not be aware that the knowledge, that they are defining, will be integrated in a cooperative distributed problem solver; they just work on generic problem classes.

4.- Communications: They are fully transparent. The communication manager of each agent decides when to send a message, whom, and what to do when a message arrives. The user doesn't have access to the communications, i.e., he cannot send a message or read a received one.

5.- Social control: A set of internal predefined agents have been defined, to make social control tasks. These agents remain hidden to the rest of the society.

6.- Architecture: An open architecture supports the model. A blackboard system, which remains hidden to the model, has been used for the architecture, to facilitate the implementation and the transportability to other systems.

7.- Implementation: The implementation is absolutely transparent, such that the engineer can not know any detail about it.

The transparency allows:

- Abstraction: The expert is not concerned with the negotiation process, the communication protocols, the problem resolution and the system control.

- Reutilisation: The knowledge has been defined independently from the problem and the environment; therefore, it can be used for the resolution of different problems.

- Easier development: The engineer does not need to develop and control internal processes. The whole social environment has been previously defined and when an agent is created, all its architecture and capabilities are automatically acquired.

Thus, the model is not oriented to users who want to control the internal processes and communications. It is suitable for non MAS experts, with applications that do not require complex information interchanges, facilitating them the development of distributed problem solvers.
2. MAKILA'S COOPERATION AND RESOLUTION MODEL

MAKILA offers to the engineer the background to develop MAS applications. The societies in MAKILA are composed of a set of autonomous problem solvers, called agents. The agents work asynchronously communicating by means of messages.

There are two kinds of agents:

- **Predefined internal agents**: They control the correct performance of the society. They are created together with the society, and the user cannot accede to them. The developed tasks are application and domain independents.

These agents are different to the ones defined by the user; they neither offer nor accept contracts to solve goals, they only execute the task for which they were created. Then their behavior is heterogeneous, because they were designed to take maximum advantage of their features. The existence of these agents allows the engineer to use the social infrastructure, focusing his attention only on the agents definition.

For example there has been defined a Loops Avoidance Agent, which controls that there are not loops in a sequence of consecutive contracts for the same goal. It traces the contract negotiations and breaks those that can close a loop. Other control tasks that the predefined agents can execute are: communication trace, performance statistics, message translations.

- **Domain agents**: They are the agents defined by the user to solve the application problems. They cooperate with the society solving other agent's goals. When an agent has a goal that it cannot solve with its own knowledge, it tries to contract other agents for its resolution.

The user defined agents are KBS that work with two knowledge categories:

- **Domain knowledge**: It is composed of a set of Resolution Methods (RM) defined by the application implementer. Each resolution method has an executable function to solve a set of goal classes.

- **Social knowledge**: It is composed of the cooperation model based on negotiation, and the information about society's agents and the tasks they can solve. This is internally handled, and the user is not concerned with it. The social knowledge can be initially empty, and increase with the agent's experience. The agents acquire information about the existence of other agents and their capabilities, during the contract negotiations.

The agents run two processes asynchronously: the goal resolution and the communication management. The interactions between both processes are internally controlled, not defined by the user, and there are only four possibilities.

- The goal resolution process transfers a goal to the communication manager to be solved in the society context.

- The communication manager integrates a contracted goal in the resolution process.
accepted, the contracted agent integrates the corresponding goal in its resolution system, being the contracted agent’s goal equal to the contractor’s goal.

There are not differences for an agent among the goals created in a task during the execution of a RM, and those accepted as contracts to the society. Both are equally treated by the agent’s resolution engine, and both follow the same resolution steps. This equivalence is a necessary restriction for a real transparency.

Therefore the problem resolution is not executed at two disjunct levels, with two different resolution engines, and the need to specify how an agent can contract a problem resolution to another agent. It could be said that:

- The agents apply methods to solve the goals.
- The society applies agents to solve the goals that an agent couldn't solve internally.

The main difference between both stages is that for the internal resolution there is a central control that selects the goal to be solved, and the method to be executed, while there is not a centralized control in the social resolution. The own agents decide when to offer a goal to be solved and to which agent, and the addressee of the contract offer decides whether it wants to work or not. In other words, the agents have a central control that manages the internal resolution, but the society has the control distributed among the agents.

A consequence of the coherent integration of both resolution models is that the domain knowledge definition is not affected by the extension of a single resolution system towards a distributed one.

2.2 Contract negotiation

The organizational structure of the society, represented by a net of agents linked by contracts (Fig. 2), is dynamically adapted to the problem needs and the agent’s availability. This structure can be improved with the experience of each agent, that learns about the capacities of other agents. Moreover, the system can react to the disappearance of agents, adapting the structure to the new society configuration, and integrate new agents. That is done during the resolution; when an agent does not answer to the messages, a new one is searched to solve the problems.

The agents defined by the user can decompose a problem into smaller subproblems and submit them to other agents. Every agent can play the role of manager or contracted agent, or both together. An agent can offer contracts to solve problems to other agents. The receivers of these contracts, evaluate them and decide whether to accept or not. Once the contracted agent finishes, it returns the results to the manager. If a contract is not answered in a reasonable period of time, a new contract is offered to a different agent.

When an agent cannot solve a goal and doesn't know anyone adequate for it, the agent sends a request to the society's agents asking for someone able to solve the problem. The interested agents answer the request. This communication enriches the agent's social knowledge, and
give chance to new agents to collaborate and be integrated.

Figure 2: Contract links example

Therefore, the cooperation among the different agents that belong to the society is based upon their mutual interest in solving goals and acting as managers or as contractors.

3. DEFINING THE AGENTS AND THEIR KNOWLEDGE

The definition of the society in an application starts with the agents declaration. Once an agent has been created, its domain knowledge can be specified in a set of Resolution Methods. When the agents and the RM have been defined, the execution can begin, with the creation of a goal in an agent. The goal represents the problem to be solved. During the resolution new subgoals can be generated, starting new resolution cycles, that can give rise to offer contracts to other agents.

A tool to support the MAKILA model has been developed, with a LISP environment. Next are the main function definitions necessary to build an application. They show the transparency of the system, and demonstrate that there is not need to define anything but the set of agents that form the society and the domain knowledge for general problem resolution.

The examples are taken from an Urgent Medical Assistance problem, which is explained deeper in the next section about the working example.

The user creates an agent with the instruction:

(\texttt{define-agent \textit{name}})

where \textit{name} identifies the agent in the society. No more instructions or information is required. When the user defines an agent, the architecture that supports it [Urzelai & Garijo 94] is automatically created, fully operative (Figure 3):
- The conceptual Panel, formed by the modules that contain the knowledge that the agent uses to solve problems.

- The application panel, with temporal data and information that the agent uses during a particular problem resolution process.

Initially the domain and social knowledge are empty. The former must be specified by the engineer in the methods, and the latter are acquired during the cooperation processes. Optionally, the engineer can define part of the social knowledge, and afterwards it will be automatically extended and improved. The user can adapt the behavior of the agent to his wishes modifying the agent’s control parameters.

The RM's are schemes with the domain knowledge defined by the expert. An agent can have several methods capable to solve the same goal class. These methods can follow different methodologies, so that when one of them fails, the agent tries to solve the goal applying another one.

The engineer creates the RM's with:

```
(define-method name agent goal-list function )
```

The name is an identifier associated to the method, and therefore it must be unique for each agent. The agent parameter specifies the name of the agent to which the RM belongs. The goal-list specify the problems that the method can solve, i.e., the goal classes for which the method may be executed. At last, function must be an executable LISP function. This function can be from a simple operation to a complex system, so that an E.S. constructed to work
alone, can be integrated in a cooperative environment, defining it as the function of a method, and putting as \textit{goal-list} the goal classes it solves.

The method's function receives a parameter when it is executed. The value of the parameter is associated to the goal instance for which the method is triggered (see below the \textit{create-goal} function definition). Its finality is to transfer to the solver information associated to the goal instance, that can be necessary for the resolution. The value returned by the function is assigned as the goal's solution.

In a RM's function, the expert can generate new subgoals; there can be methods that just decompose a goal into subgoals. The function \textit{create-goal} creates a new goal instance:

\begin{verbatim}
(create-goal goal-class &KEY :parameters parameter-list :weight weight-value :expectation expectation-function :on-solved-goal solved-function :on-failed-goal failed-function)
\end{verbatim}

The most important value, and the only one indispensable, is \textit{goal-class}, that specifies the problem to be solved. This value restricts the set of methods and agents that can be selected for the resolution of the goal. It is sent too in the contract offers, so that the receivers may evaluate whether to accept the contract or not. The \textit{parameter-list} allows the user to pass the necessary data for the resolution, to the method or agent selected to solve the goal instance. This value corresponds to the parameter that the method's function receives.

The \textit{weight-value} express the priority of the goal instance to be solved. By omission, it is the lowest possible. The resolution engine sorts out the unsolved goal list according to an evaluation function. This function corresponds to the weight value by default. The application implementer can change this evaluation function.

The \textit{expectation} must be a LISP function, that specifies when a goal instance is considered solved; it is associated to the goal instance more than to a goal class. The function is executed whenever a goal resolution attempt has finished, i.e., when a contracted agent has returned a solution, or a RM, and the subgoals created into it, have finished. The function receives as parameter the goal's solution. If it satisfies the expectation, the goal has been solved and the solution is propagated back, to the task that created the goal. If not, the goal remains unsolved, and will be candidate for resolution again, with a different method or agent.

The functions \textit{solved-function} and \textit{failed-function} are executed depending on if the goal ends solved or failed. Both receive the goal's solution as parameter. The \textit{solved-function} allows the use of the goal's solution to follow with the resolution of the problem that created the goal. The \textit{failed-function} allows to take some actions (e.g., to create an alternative goal), preventing from failing the task where the goal was created, only because of the fail of a subgoal. The result of
both functions does not affect to the subgoal to which they are associated. They are executed in
the context of the task where the goals were created.

Therefore, the application implementer do not specify anything about who, how or when the
goals should be processed. The selection of a method or an agent for a goal is internally done
by the agent, freeing the expert from any decision. Once a goal has been solved, the solution is
returned back, and there is not distinction whether it was solved by the own agent with its
knowledge (nor which method did), or it was solved by the society (nor which agent did).
This means that the domain knowledge may be defined by different experts that do not know
anything about cooperation and negotiation paradigms, and that there is not need of
coordination among them, to make all the knowledge work together.

Another remarkable feature, is that no negotiation or communication instructions are needed.
The engineer, optionally, can define the existence of an agent to the social knowledge of other
agent. This is done using the function:

```
(define-known-agent identifier goal-list agent )
```

The parameter agent specifies which is the agent to which the information will be included.
The information is that the agent identifier solves goals of the classes included in goal-list.
To start the resolution of a problem, a goal class representing the problem is created using the
following function:

```
(run-system goal-class agent & KEY :parameters parameter-list
 :expectation expectation-function
```

The attributes are equivalent to those explained for the create-goal function. The goal-class
defines the general problem to be solved. For its resolution the explained processes of
decomposition, subgoals generation, contract negotiation and problem solving are used.

There are other functions too, to initialize the tool, ask for information about the society, the
agents and their knowledge, and trace the execution. Some internal operators are
parametrizable too, being possible to modify some control functions, as for example the
selection functions for the goals, methods and agents to be contracted. The default functions
for conflict resolution in the method/agent selection, choose the one who has the best relation
between the number of times it solved the assigned goals with respect to the times it failed.
4. WORKING EXAMPLE

The example is based in an application that implements an Urgent Medical Assistance problem [Garijo 87]. This section does not describe the whole application; it only describes some functions, to show the use of the tool and to demonstrate that the application implementer defines the domain knowledge, without any reference about how the agents should cooperate.

The problem consists of an operator receiving phone calls informing about incidents. The operator has to identify the incident and send an ambulance to assist the injured people.

Five agents are used in the example (Figure 4):

- **Operator**: Makes the primary knowledge acquisition, and generates two subgoals for the incident location and the ambulance selection.

- **Locator**: Locates the incident place.

- **Ambulance Assigner**: Selects an available ambulance, according to the damages. It needs to know the damage evaluation to make a selection.

- **Damage Evaluator 1** and **Damage Evaluator 2**: evaluate the type of the incident and its graveness.

![Figure 4: Example framework](image)

The difference between the two **Damage Evaluators** is that the **Ambulance Assigner** will be informed about the existence of the former. The **Damage Evaluator 1** fails solving the goals, thus the **Ambulance Assigner** will have to search for another agent able to evaluate the damages: the **Damage Evaluator 2**. The objective of this redundant definition is to demonstrate that when agents cannot solve the goals with its own knowledge, they offer contracts to the ones included in their social knowledge adequate for the goal. If they fail, a general request is send to the society, looking for adequate agents. On later problems, when the **Ambulance Assigner** needs an agent for a damage evaluation, it will prefer the **Damage
Evaluator 2 before the Damage Evaluator 1

The agents are created by means of the define-agent function. The Operator is created as:

```
(define-agent 'operator)
```

This instruction creates the agent's structure and capabilities, integrating it into the society. The rest of the society's agents are defined similarly.

Once an agent has been created, its knowledge can be defined. With the function define-method the agent's domain knowledge is defined. The Operator has a method to solve goals of class Operation. This method calls to the create-goal function to create two subgoals: Location and Ambulance Assignation.

```
(define-method
  'operator-method ;method's name
  'operator ;agent, owner of the method
  '{Operation} ;list of goals classes that the method solves
  ; executable LISP function that receives the parameters defined for the goal
  #'(LAMBDA (goal's-parameters)
    ...
    ;standard LISP code
    (create-goal ;subgoal to be solved
      'Ambulance_Assignation ;goal class
      :WEIGHT 50 ;goal's priority
      :PARAMETERS patient ;parameter that the solver will receive
      :EXPECTATION
        ;Function that specifies when the goal will be considered solved
        #'(LAMBDA (goal's-solution)
          (available-ambulance-P goal's-solution))
      :ON-SOLVED-GOAL ;actions to take when the goal is solved
        #'(LAMBDA (solution) ;the solution is an ambulance
          (reserve solution))
      :ON-FAILED-GOAL ;actions to take when the goal fails
        #'(LAMBDA (result)
          (DECLARE {IGNORE result}))
      (CREATE-GOAL taxi-search ...)))
    ;standard LISP code
    (create-goal 'Location ...)
    ...)) ;standard LISP code
```

The methods of the rest of the agents are defined too.

Part of the social knowledge can be defined. The only definition in the example is to inform the Ambulance assigner that there exists an agent called Damage Evaluator 1, that solves goals of Damage-Evaluation class.

```
(define-known-agent
  'Damage-Evaluator-1 ;agent that is being described
  '{Damage-Evaluation) ;goal classes that it solves
  'Ambulance-Assigner) ;informed agent
```
4.1 Execution

The example execution starts with the creation of the initial goal Operation into the Operator agent.

\texttt{(run-system 'operator 'Operation)}

The Operator executes its method to solve the goal. During the method execution, the goals Location and Ambulance Assignment are created; when the method execution finishes the task stays pending until the subgoals are solved or fail.

The Operator has neither methods nor social knowledge for the new goals, so the agent sends a message to all the society looking for someone able to solve them. The Locator and the Ambulance Assigner answer. The information about both agents is included in the Operator's Social Knowledge. Two contract offers are sent, and accepted.

The Locator solves the Location goal, and returns a solution message to the Operator.

While the Locator solves its goal, the Ambulance Assigner works on the Ambulance Assignment goal. During its resolution a Damage Evaluation goal is created. It has an on-solved-goal function, to use the damage evaluation solution to select the most accurate ambulance.

The Ambulance Assigner cannot solve the new subgoal, but it knows to the Damage Evaluator 1 that solve goals of its class. A contract offer is sent, and accepted. Later on the contracted agent returns a message informing that it has failed.

The Ambulance Assigner doesn't know to anyone else, therefore, it sends a request to the society asking for available agents for the goal class. Both Damage Evaluator answer; the Damage Evaluator 2 is included into de social knowledge and contracted.

The Damage Evaluator 2 solves the goal and returns a solution, which is used by the Ambulance Assigner to select an ambulance.

The selected ambulance is returned to the Operator that checks the expectation. It is satisfied; therefore, the on-solved-function is executed to make the reservation.

As the subgoals Location and Ambulance Assignment are solved, the main goal Operation ends solved too.

The message flow-chart is shown in Figure 5.
4.2 Results

When the application execution finishes, the agent's conceptual level has been modified, such that the knowledge acquired during the resolution, can be used in later executions to improve the application performance. The enrichment is done at two levels:

- **Social Knowledge:** The Operator, that had an empty social knowledge at the beginning, has information about the Locator, that solves goals of the class Location, and the Ambulance Assigner that solves goals of the class Ambulance Assignment. Both have solved the assigned goals, therefore, they have a good trust value. So, in the future, when the Operator needs an agent for these goal classes, it will not need to send messages looking for available agents.

The Ambulance Assigner, that knew to the Damage Evaluator-1, has information about the Damage Evaluator-2 too. Both solve the same goals class, but the first one has a negative evaluation. From now, next time the Ambulance Assigner needs an
agent, will prefer the *Damage Evaluator-2*.

- **Resolution Methods**: As with the social knowledge, the methods have a trust value too, according to whether they solved the assigned goals or not. Therefore, in the next method selection, it will be chosen first, the one with the best rate of solved goals.

5. **CONCLUSION**

The tools to develop DAI applications usually offer the engineer some assistance to define the agents, their knowledge, the communications and to control the problem. The paper has presented how the MAKILA model frees the expert from every internal process. He has to define only the domain knowledge, independently from being in a distributed system or a centralized one, being transparent every resolution, control, communication and negotiation process.

A tool that supports the MAKILA model is described. The tool offers the user an environment to define a society made up of independent agents. The system also supplies some predefined agents to perform control tasks and to trace the global cooperation processes.

The article explains the available functions to develop an application, showing the independence of the domain knowledge with respect to the selected implementation and the internal processes, that remain all them hidden.

The system is now running in an Explorer II. Its implementation has been developed in LISP, using the GBB blackboard system [Gallagher et al 88], and has been tested on an Urgent Medical Assistance problem. In the current implementation, the agents reside in the same LISP shell, and cannot be distributed among different platforms.

**REFERENCES**


in a Distributed Problem Solver". IEEE Transactions on Computers, Vol. C-29 no 12 December 1980, pags 1104-1114


LSI-96-1-R “(Pure) Logic out of Probability”, Ton Sales.


LSI-96-3-R “A Frame-Dependent Oracle for Linear Hierarchical Radiosity: A Step towards Frame-to-Frame Coherent Radiosity”. Ignacio Martín, Dani Tost, and Xavier Pueyo.

LSI-96-4-R “Skip-Trees, an Alternative Data Structure to Skip-Lists in a Concurrent Approach”, Xavier Meseguer.

LSI-96-5-R “Change of Belief in SKL Model Frames (Automatization Based on Analytic Tableaux)”, Matías Alvarado and Gustavo Núñez.

LSI-96-6-R “Compressibility of Infinite Binary Sequences”, José L. Balcázar, Ricard Gavaldà, and Montserrat Hermo.

LSI-96-7-R “A Proposal for Word Sense Disambiguation using Conceptual Distance”, Eneko Agirre and German Rigau.


LSI-96-10-R “POS Tagging Using Relaxation Labelling”, Lluís Padró.


LSI-96-12-R “Using Bidirectional Chart Parsing for Corpus Analysis”, A. Ageno and H. Rodriguez.


LSI-96-14-R “Logic as General Rationality: A Survey”, Ton Sales.


LSI-96-17-R “Rewriting in Categories of Spans”, Miquel Monserrat, Francesc Rosselló, Joan Torrens, and Gabriel Valiente.


LSI-96-19-R “Learning Causal Networks from Data”, Ramon Sangüesa i Solé.


LSI-96-21-R “Exact Learning of Subclasses of CDNF Formulas with Membership Queries”, Carlos Domingo.

LSI-96-22-R “Modeling the Thermal Behavior of Biosphere 2 in a Non-Controlled Environment Using Bond Graphs”, Angela Nebot, François E. Cellier, and Francisco Mugica.


LSI-96-25-R “Decision Trees Have Approximate Fingerprints”, Víctor Lavín and Vijay Raghavan.


LSI-96-27-R “Massively Parallel and Distributed Dictionaries on AVL and Brother Trees”, Joaquim Gabarró and Xavier Meseguer.


LSI-96-29-R “Bounding the expected length of longest common subsequences and forests”, Ricardo A. Baeza-Yates, Ricard Gavaldà, and Gonzalo Navarro.


LSI-96-31-R “ParaDict, a Data Parallel Library for Dictionaries (Extended Abstract)”, Joaquim Gabarró and Jordi Petit i Silvestre.

LSI-96-32-R “Neural Networks as Pattern Recognition Systems”, Lourdes Calderón.

LSI-96-33-R “Semàntica externa: una variant interessant de la semàntica de comportament” (written in Catalan), Vicent-Ramon Palasí Lallana.


LSI-96-39-R “Reducción de l'equivalència inicial visible a teoremes inductius” (written in Catalan), Vicent Ramon Palasi Lallana.


LSI-96-41-R “Semànica algebraica del llenguatge AL: l'algorisme a” (written in Catalan), V.R. Palasi Lallana.

LSI-96-42-R “Partial Occam’s Razor and its Applications”, Carlos Domingo, Tatsuie Tsujiki, and Osamu Watanabe.

LSI-96-43-R “Transparent Distributed Problem Resolution in the MAKILA Multi-Agent System”, Karmelo Urzelai.

Hardcopies of reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo, 5
08028 Barcelona, Spain
secrelsi@lsi.upc.es

See also the Department WWW pages, http://www-lsi.upc.es/www/