Extending a Single Resolution System
Towards a Distributed Society

Karmelo Urzelai

Report LSI-93-21-R

BIBCE
Facultat d'Informàtica
de Barcelona - Biblioteca
EXTENDING A SINGLE RESOLUTION SYSTEM TOWARDS A DISTRIBUTED SOCIETY

Karmelo Urzelai
Universitat Politècnica de Catalunya, Pau Gargallo 5, Dpt. LSI, Barcelona
karmelo@lsi.upc.es

Abstract - This article shows how a conventional problem solver is extended to have a set of agents, each of them based on the former problem solver, that cooperate in a social framework. The evolution and differences from the single resolution system towards the distributed one are explained.

A generic tool for the design and implementation of systems using this model has been developed. A remarkable feature of the resulting system is the independency of the knowledge of the problem solving, from the architecture that supports the society.

Problem resolution is goal driven. Goals are solved by agents with their own methods, or by submitting them to other agents. During a method execution new subgoals can be generated, starting new problem resolution cycles.

The architecture that supports the model is presented too. A working example is described, showing the tool use.

1. INTRODUCTION

When the knowledge based systems grow up too much, their complexity increase exponentially, becoming unmanageable. A solution is to divide the problem in subproblems, and to build smaller systems specialized in each subproblem class, in such a way that these systems, working together, can solve big problems.

This solution can be compared with a social model where a team of people, professionals in different areas, cooperate to solve problems that cover diverse areas.

This is one of the foundations of the Distributed Artificial Intelligence. Some of the main contributions of this field are:

- Natural adaptation to the application domains
- Fault tolerance
- System engineering improvement
  - Modularity
  - Knowledge independence
  - Reusability

An interesting approach has been the cooperation based on negotiation. The most representative model of this paradigm is the Contract Net [8]. This model offers a framework to make heterogeneous agents cooperate with a distributed control, local to each agent. The agents solve problems, negotiating to find the best suited node to solve the tasks.
The Contract Net model has been used in many systems [1,10]. For this reason, for the development of systems based on this paradigm, there have been developed tools to help the knowledge engineer in its implementation [7], with predefined messages and facilities to define the agents and send messages, as soon as an environment where to define a problem resolution system.

The experts that defines how to solve concrete problems, usually doesn't have knowledge about DAI, communication protocols, ... Thus it is not a realistic approach, that, where the knowledge to solve the problems must contain the information about protocols, the surrounding society, and the selected implementation.

In this way, there is a lack of general tools to develop systems based on the negotiation paradigm that frees the knowledge engineer from communication protocols and the need to define a problem resolution process.

Thus the main objectives pursued in the MAKILA system [9] are:

- The development of a generic tool that use the Negotiation paradigm [2] in a Distributed environment.
- Knowledge independence: The knowledge can be designed by different experts in each field, without being aware that it is for a cooperative distributed system.
- The design of an architecture oriented to the creation of societies formed by intelligent agents. The system offers the user an environment to define the agents, their knowledge and control and debugging functions.
- Architecture and implementation transparency.
- Internal social control. Some tasks like the detection and elimination of contract loops are done by the system itself, through internal predefined agents.

2. A SINGLE RESOLUTION SYSTEM

Let's start thinking in a problem solver working alone, not attached to any other system. The system should be as general as possible, because afterwards it will be extended to work in a distributed environment, in such a way that the problem resolution knowledge should remain being applicable without any adaptation to the new model where it will be integrated.

The developed resolution system is goal driven. The system disposes of a set of Resolution Methods, defined by the user, that can be applied to the goals in order to solve them. During the resolution phase new goals can be created, which start new resolution cycles for them.

2.1. Single Resolution System Architecture

The architecture of the resolution system is made up of four modules (Figure 1):

Resolution Methods: Which are defined by the user to solve the goals. When a method is defined, the goal classes that it solves and an executable procedure must be declared. The procedure can be from a simple function to a complex Expert System, such that an E.S., designed to work independently, can be integrated into a more complex environment, being included in a Resolution Method (R.M.), and adding it the information of the goal classes that it can solve.

The methods have an internal attribute containing information about how well they solved previous goals, which is automatically updated each time it is executed. This information is used to calculate a trust value [6] of each method during the method selection of the goal resolution phase.

Goal Instances: These are the goals generated during the problem resolution phase. When the user declares, in the procedure of a R.M., to create a new goal, it just specifies the goal class to be solved, the parameters that must be passed to the method that will try to solve it, the expectation, i.e. when the goal will be considered to be solved, and the actions to be executed when the goal is solved or when it fails. It never declares who, when or how the goal should be processed; these decisions are taken by the internal resolution system. Actually, after a goal has been solved, there is no distinction about which was the method that solved it.
Tasks: Associate goal instances with the Resolution Methods selected to solve them. More than one task can try to solve a single goal; some of them can fail, but if someone ends satisfactorily and the expectation of the associated goal is validated, the goal is solved. A method can decompose a problem in subproblems; when that R.M. is executed in a task, new subgoals are created, starting new resolution cycles. These subgoals are linked to the task, which will fail if one them fails.

2.2. Single Reasoning Cycle

The reasoning cycle is executed by the system itself, and remains hidden to the user, i.e., the knowledge engineer doesn't need to design, implement and control a resolution engine. It consists of the following steps:

1.- Selection of the next goal to be solved, from the set of unsolved goals.
2.- Selection of an appropriate method for the goal.
3.- If there is such method, it is instantiated in a task. The task is activated, being started the execution of the method's procedure.
4.- If the task fails, the goal remains unsolved, then it returns back to be candidate for resolution. If the task ends correctly, the goal's expectation is checked:
   - if it is satisfied the goal has been solved, and the results are propagated back, to the task that generated the goal.
   - if the expectation fails, the goal returns to be unsolved.
5.- If there is not a method for the goal, or all of them have failed trying to solve it, the goal fails. This failure is propagated back, being the task where it was created failed.

3. MAKING AGENTS COOPERATE IN A SOCIAL FRAMEWORK

This model has been extended towards a society composed of distributed systems, called agents. The agents work independently communicating by means of messages. Each of them is an expert in some problem classes, having the necessary knowledge to solve them. There is no centralized control; it is local to each agent.

3.1. Agents

There are two kind of agents in the MAKILA system: internal predefined agents, and user defined agents. The former are created together with the society, and the user is not concerned with them. They are used for social control tasks. As an example of an internal agent consider the Loops Avoidance Agent, which controls that there are no loops in a sequence of consecutive contracts for a goal (for example if agent A contracts with agent B for a goal, B contracts with agent C for the same goal, and C tries to contract with A for that goal). Its task is to keep listening to the
communication line and to annotate every contract established between agents. When a contract loop attached to the same goal class is detected, it breaks the last loops' contract. These internal agents remain hidden for the rest of the agents.

The user defined agents are based in the single resolution model, extended to cooperate with other agents in a society: an agent can solve other agents' goals and offer contracts for the goals that it cannot solve by means of its own knowledge.

To define a new agent, the user just declares its name, everything else is created automatically: the agent's architecture, the reasoning system, the communication and negotiation capacity and the default control information. The agent has neither the knowledge to solve problems, which must be defined by the user in the Resolution Methods, nor social knowledge, which will be acquired automatically through the agent's negotiation processes.

The process to create new goals and the definition of the Resolution Methods and their procedures, are equal to the single resolution system, the only difference is that when a R.M. is defined now, the name of the agent to which it belongs must be specified; there is not any other difference. Moreover, after a goal has been solved, there is no distinction whether it was solved by the agent with its own knowledge or by a contracted agent. Therefore, the knowledge about how to solve the problems (included in the R.M.), is independent from being in a centralized system or in a distributed one.

3.2. Agent's Architecture

The single resolution system's architecture has been extended to support the new features and the communications with the society's agents. The following knowledge has been added to the agent, which remains hidden to the user.

Social knowledge: It is the view that each agent has of the society. When an agent cannot solve a goal through its own knowledge, it tries to submit the goal to other agents. If it has information about them, it directly offers the contract, otherwise it sends a message addressed to everybody, asking for someone able to solve the goal class. The information about which agent can solve which goals is the social knowledge. As in the methods, there is an additional attribute associated with each agent with information about how well they solved previous contracts, which is used to calculate the agent's trust during the selection of the best suited agent, from those able to solve the selected goal.

Self knowledge: It is the set of goal classes that the agent is able to solve, i.e. the goals that the agent's resolution methods can solve. When a contract offer is received from another agent, it is verified whether the received goal class belongs to the set of goal classes that the agent can solve, or not.

Communication capability: The agent must be able to send messages and recognize the received ones. When a contract message arrives, the agent decides whether to accept or to refuse it. If the contract message is accepted, the associated goal is integrated into the goal resolution environment.

The knowledge engineer doesn't access the messages, he cannot order to send a message or read a received one; all these decisions and tasks are internally done by the communication manager.

There are two main protocols (Figure 2). The first one is used when an agent has a goal to be offered as contract, but it doesn't know to any other agent able to solve it. In this case, it sends a Goal Announcement message to all the society, looking for someone that solves the goal's class. Those that receive the Goal Announcement and that have adequate methods for that goal, answer with an Agent Available message. The second main protocol is used when an agent has a goal that it cannot solve and it knows another agent which can solve it, a Contract message is sent. The receiver answers whether it accepts the contract or not, with a Contract Acceptance message. If the contract is accepted, once it has been solved or it has failed, the contracted agent sends a Solution message with the results. If the contractor wants to break the contract before the contracted agent has finished, it sends a Contract Cancellation message to the contracted agent.
3.3. Reasoning cycle extension

The single reasoning cycle has been extended too, such that when the agent doesn't have a method to solve a goal, it doesn't fail, the following steps are added:

1.- Selection of an agent from the social knowledge, able to solve the selected goal.
2.- If there is such agent, a task is created with the information, and a contract offer is sent to the agent.
3.- If there is not an adequate known agent
   a.- A request, looking for an agent capable to solve the goal, is sent to all the society, if it hasn't been sent yet. The information about the agents that answer affirmatively to the message, is added to the agent's social knowledge, and will be used later to solve the goal.
   b.- If the request was sent already, the goal fails. This result is propagated back to the one which created the goal.
4.- If the contract offer is not accepted or the contracted agent doesn't solve the goal, the goal comes back to be unsolved.
5.- If the contracted agent returns a message with a solution, the goal is solved or unsolved depending on the goal's expectation.

While the agent waits that a contracted agent solves a submitted goal, or while it is expecting an answer to a message, it can work on other goals.

3.4. Society's Architecture

The society is the environment where the agents are located. When the user defines the RM with the knowledge to solve the goals, it doesn't need to be aware of the cooperative environment; the social definition and maintenance is done internally by the tool. The architecture is divided into four areas (Figure 3):

- The agent area: where the user defined agents and the internal agents are located. They are connected through the communication area.
- The communication area: It works as a communication line, storing the agent's messages, until the addressees read them. There is neither a manager nor a message processing.
The control area: composed of the internal control data, used by the society's execution functions, and the functional parameters, defined by the user to adapt general behaviour to its wishes. These data are read and modified by the internal processing functions.

The user interface: There is a set of functions to create agents, add methods to them, define the user's parameters, or ask for information about the society. There are some internal functions too, they trace the execution and display informative messages.

4. WORKING EXAMPLE

The working example implements an Urgent Medical Assistance problem [5], that consists of an operator receiving phone calls informing about incidents. The operator must develop the following tasks.

1.- Identify the problem.
2.- Locate the incident.
3.- Assign an ambulance according to the damages.

In our example the tasks have been assigned to diverse agents (Figure 4). These are: the Operator, the Locator, the Ambulance Assigner and the two Damage Evaluators. The agents can answer requests from any agent.

As example of use of the tool, the definition of the Operator agent, one of its methods and how the goals are created in the method, follows:

The operator agent is defined with the following function:

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

It has a method that solves goals of class "Operation", and which actions are:

1.- Identifies the incident class
2.- Generates the goals "Location" and "Ambulance Assignation"
The method is defined as:

\[
\text{(define-method 'operator-method 'operator 'Operation) \\
\quad \#'(LAMBDA (goal's-parameters) \\
\quad \quad \ldots \\
\quad \quad (create-goal 'Location \\
\quad \quad :PARAMETERS \ldots \\
\quad \quad :EXPECTATION \#'(LAMBDA (goal's-solution) \\
\quad \quad \quad ;specifies when the goal will be solved \\
\quad \quad \quad \ldots \)) \\
\quad (create-goal 'Ambulance Assignation \ldots) \ldots)}
\]

The system execution is started with the creation of the initial goal "Operation" into the operator agent:

\[
\text{(run-system 'operator 'Operation)}
\]

After the execution, the Operator has information about the Locator and the Ambulance Assigner with a positive evaluation. The Ambulance assigner knows to the Damage Evaluator 1 and the Damage Evaluator 2, the first one has a negative evaluation because it couldn’t solve the contracted goal and the second one with a positive evaluation. The methods of the agents will have equivalent evaluations according to the results of their execution.

5. IMPLEMENTATION

The model is working on a TI Explorer in Lisp. The system has been implemented using a blackboard architecture [3]. The main reasons for this selection are:

- Modularity: due to the information structure in the blackboard and the functional independence of the Knowledge Sources.
- Abstraction: the use of high level tools frees from the implementation of low level operators.
- Transportability: the blackboard paradigm is widely extended, therefore the implementation of the system in another environment with blackboards is much easier and faster.

The selected environment has been the GBB [4] with the GBB1 control shell. This is a generic blackboard environment which provides flexibility, ease of implementation and efficient execution. The blackboards remains hidden to the user.
6. CONCLUSION
This article has presented how a single generic system for problem solving is extended, to work in a distributed manner with other equivalent systems, negotiating contracts for the resolution of other agents' goals and offering contracts of those problems that it cannot solve.

However, this step towards a society of agents that work asynchronously, communicating by messages, doesn't affect at all the knowledge about how to solve the problems. In fact, this knowledge can be developed by people that ignore the paradigm of distributed agents, the negotiation processes, the communication protocols and any detail about the architecture and implementation used.

The MAKILA system offers the user an environment in which to define the society, its agents, their knowledge, and control and debugging functions to adapt and trace global performance.

The main advantage of using the system is from the knowledge engineering point of view. Communication and implementation transparency, the abstraction of the way the goals are solved with an integrated problem solver and the division of the knowledge in separated modules, makes solving big problems more accessible, and produce more readable code, with modules that can be used separately for different problems.

ACKNOWLEDGEMENTS
This research was partially supported by the Spanish Council of Scientific and Technical Research CICYT under the national scholarship programme. I want to express my special thanks to Paco Garijo, without whose ideas, comments and encouragement this work wouldn't have been possible.

REFERENCES
List of research reports (1993).


LSI–93–7–R “A characterization of $\mathsf{P}^{\mathsf{NP}} = \mathsf{P}^{\mathsf{NP}[\log]}$, Antoni Lozano.


LSI–93–16–R “The Odisea approach to the design of information systems from deductive conceptual models”, Maria Ribera Sancho and Antoni Olivé.


LSI-93-21-R “Extending a single resolution system towards a distributed society”, Karmelo Urzelai.

Internal reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics (U.P.C.)
Pau Gargallo 5
08028 Barcelona, Spain
secrelsi@lsi.upc.es