Intelligent Agents that Monitor, Diagnose and Solve Problems

Two Success Stories of Industry-University Collaboration

Daniela Briola, Viviana Mascardi and Maurizio Martelli

DISI, Università degli Studi di Genova,
Via Dodecaneso 35, 16146, Genova, Italy
{Daniela.Briola, Viviana.Mascardi, Maurizio.Martelli}@unige.it

Abstract: This paper describes the results of two joint academy-industry projects that involve the Computer Science Department of Genoa University, Italy, and Ansaldo-STS, the Italian leader in design and construction of railway signalling and automation systems.
The MAS developed as part of the MAD Agents ("Monitoring and Diagnostic Agents") project monitors the behaviour of a set of processes running on an Ansaldo-STS server, whereas the goal of the MAS developed in the FYPA ("Find Your Path, Agents") project is to find a feasible allocation of resources to agents over time that emerges as the result of a sequence of local negotiation steps.
Both MASs have been developed in collaboration with scientists and engineers from Ansaldo-STS and have been widely tested on real data provided by the company.

Keywords: agents, multi-agent systems, diagnosis, monitoring, distributed problem solving.

1. Introduction

The AgentLink III Technology Roadmap [33] defines an agent as: "a computer system that is capable of flexible autonomous action in dynamic, unpredictable, typically multi-agent domains."

According to [34], agents should be
- **autonomous**: they should operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;
- **responsive**: they should perceive their environment and respond in a timely fashion to changes that occur in it;
- **pro-active**: they should not simply act in response to their environment, but should exhibit opportunistic, goal-directed behaviour and take the initiative where appropriate;
- **social**: they should be able to interact, when appropriate, with other artificial agents and humans in order to complete their own problem solving and to help others with their activities.

Another characterizing feature of agents is **situatedness**: the agent receives sensory input from its environment and it can perform actions which change it in some way [13].

As far as sociality is concerned, it is now widely recognized that **interaction** is probably the most important single characteristic of nowadays' complex software.

A multi-agent system, or MAS for short, is a system composed by many interacting agents. In a MAS, each agent has incomplete information or capabilities for solving the problem, thus each agent has a limited viewpoint, there is no global system control, data is decentralized, and computation is asynchronous. Also, due to the dynamicity and unpredictability of scenarios where agents live, MASs are open to changes. This means that the topology of a MAS cannot be fixed a priori, but it dynamically changes as agents enter and leave the MAS.

Distributed diagnosis, monitoring, and problem solving within complex and dynamic systems are three of the oldest applications of software agents. There are many good reasons for choosing a MAS approach for facing them [32]:

- *To permit reasoning based on information of different granularity*: the MAS may be organised in a hierarchy of agents with different competencies, starting from those at the lowest level, directly interfaced with the processes to monitor and diagnose, and going up towards more and more sophisticated agents, equipped with expert system-like rules for devising problems according to the information coming from agents below in the hierarchy, reporting aggregated information and diagnosis to the agents higher in the hierarchy, and proposing solutions to problems.
- *To enable a number of different problem solving paradigms to be utilised*: there is no universally best problem solving paradigm: procedural techniques may be required for algorithmic calculations, whereas symbolic reasoning based on heuristic search may be the best approach to diagnosis. A distributed approach enables each component to be encoded in the most appropriate method.
- *To meet the application's performance criteria*: the distributed nature of a MAS makes it a suitable solution for monitoring different processes

concurrently, thus gaining in performance and responsiveness.

The motivations for choosing a Distributed Artificial Intelligence approach given by [5], [1] also apply to the diagnosis and monitoring domains: economy, robustness, reliability, natural representation of the domain. Also, distributed problem solving, with the Contract Net Protocol [36], represents one of the first examples of autonomous entities that negotiate for solving a problem that none of them, individually, could solve.

Situational awareness, that is mandatory for the successful monitoring and decision-making in many scenarios, is one of
the founding characteristics of intelligent software agents. When combined with reactivity, situatedness may lead to the early detection of anomalies and to the formulation of a suitable plan for solving them. The solution that agents find out to solve emerging problems may be submitted to a human expert, in case of safety-critical scenarios, or may directly be implemented by the agents themselves, if they are empowered to make corrective actions in the environment, without the approval of a human expert.

Last but not least, an agent-based distributed infrastructure can be added to most existing systems with minimal or no impact over them. Agents monitor processes, be them computer processes, business processes, chemical processes, by "looking over their shoulders" without interfering with their activities.

This paper describes the results of two joint academy-industry projects that involve the Computer Science Department of Genoa University, Italy, and Ansaldo-STS, the Italian leader in design and construction of railway signalling and automation systems.

The MAS developed as part of the MAD Agents ("Monitoring and Diagnostic Agents") project monitors the behaviour of a set of processes running on an Ansaldo-STS server. These processes control railway signalling. The agents that monitor them react to anomalies either by interacting with other agents in the MAS or by killing the process that raised the anomaly.

The goal of the MAS developed in the FYPA ("Find Your Path, Agents") project is to find a feasible allocation of resources to agents over time that emerges as the result of a sequence of local negotiation steps. Resources are modelled as nodes in a directed, non-planar graph that agents must traverse from one start point to one end point. Resources are indivisible and, in any time instant, they can be occupied by at most one agent.

Both MASs have been developed in collaboration with scientists from Ansaldo-STS and have been widely tested on

real data provided by the company.

The paper is organised in the following way: Section 2 discusses the MAD Agents project, Section 3 discusses the FYPA project, Section 4 overviews works related to the projects and concludes.

2. MAD Agents

The architecture of the *MAD Agents* MAS and its operating scenario have been extensively described in [17], whereas its implementation and execution have been the subject of [35]. In the sequel we briefly summarize them.

2.1 MAD Agents Operating Scenario

The Command and Control System for Railway Circulation ("Sistema di Comando e Controllo della Circolazione Ferroviaria", SCC) is a framework project for the technological development of the Italian Railways ("Ferrovie dello Stato", FS). It is based on the installation of centralised Traffic Command and Control Systems, able to remotely control the plants located in the railway stations, and to manage the movement of trains from the Central Plants (namely, the offices where instances of the SCC system are installed).
The SCC can be decomposed into five subsystems

- *Circulation*, for remote control of traffic and for making circulation as regular as possible;
- *Synoptic Frame*, for representing railway lines, nodes, and trains, in a summarised, easily understandable way;
- *Diagnosis and Upkeep*, for the diagnosis of plants and equipments of the SCC;
- *Information to Customers*, for providing information to the FS customers;
- *Remote surveillance*, intrusion avoidance, fire detection, emergency management, for dealing with all these situations efficiently.

Figure 1. Operator and synoptic frame in Genoa's SCC.

The *MAD Agents* MAS monitors and reacts to problems of

one critical process belonging to the Circulation subsystem: Path Selection.

The Path Selection process is the front-end user interface for the activities concerned with railway regulation. There is one Path Selection process running on any workstation in the SCC and each operator interacts with one instance of this process.

The Path Selection process visualises decisions made by the Planner process and allows the operator to either confirm or modify them.

The Planner process is the back-end elaboration process for the activities concerned with railway regulation. There is only one instance of the Planner process in the SCC, running on the server. It continuously receives information on the position of trains from sensors located in the stations along the railway lines, checks the timetable, and formulates a plan for ensuring that the train schedule is respected. Operators may modify the Planner's decisions thanks to the Path Selection process.

By integrating a monitoring MAS into the circulation subsystem, we equip any operator of the Central Plant (any workstation) with the means for early detecting anomalies that, if reported to the SCC Assistance Centre in a short time, and before their effects have propagated to the entire system, may allow the prevention of more serious problems.

To have an idea of the dimensions of an SCC and of the area it controls, the SCC of the node of Genoa, that we employed as a case-study for the implementation of our MAS, controls an area with 255 km of tracks, with 28 fully equipped stations plus 20 stops. One of the 16 user workstations of Genoa's SCC is shown in Figure 1. The synoptic frame can be seen in the background.

It is worth noting that our MAS does not manage problems tightly connected with the railway domain. Indeed, it monitors parameters which are common to many processes in many domains, like the use of the cpu and the hard disk, the state of the connection to the network, etc.. The aim of our project was to develop a system able to monitor the execution of a process characterised by the above parameters. As a consequence, the architecture and the MAS developed are general and flexible enough for monitoring many different processes, and not only to the Path Selection one: our system could be easily adapted to monitor new processes without changing the architecture of the MAS but just creating specific reader agents and equipping the other agents with new rules.
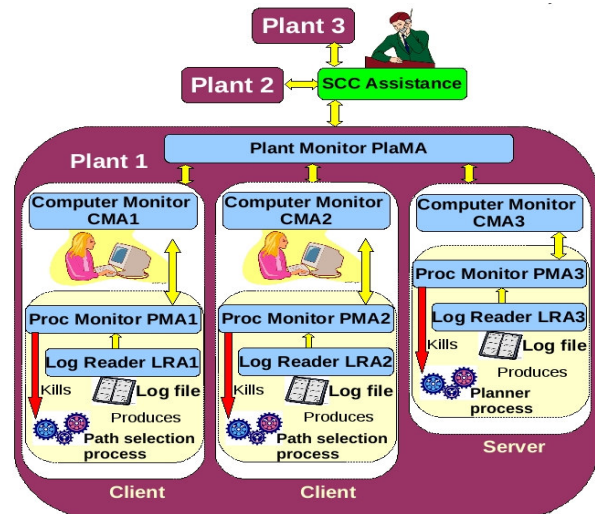
2.2 MAD Agents Architecture

Figure 2. MAD Agents Architecture.

The MAD Agents MAS consists of the four kinds of agent depicted in Figure 2.

Agents are organized in a hierarchy: Log Reader Agents are at the bottom of the hierarchy and interact with Process Monitoring Agents, which in turn interact with Computer Monitoring Agents. At the root of the hierarchy is the Plant Monitoring Agent, unique in each SCC. Agents live and act in the software Environment consisting of the already existing processes developed by Ansaldo-STS, and interact with it in the limited way discussed below.

Log Reader Agent. In our MAS, there is one Log Reader Agent (LRA) for each process that needs to be monitored. Thus, there may be many LRAs running on the same computer (if there are more processes to monitor; at the time of writing, only Path Selection is considered). Once every m minutes the LRA reads the log-file produced by the process P it monitors, extracts information from it, produces a symbolic representation of the extracted information in a format amenable of logic-based reasoning, and sends the symbolic representation to the Process Monitoring Agent in charge of monitoring P. Relevant information to be sent to the Process Monitoring Agent includes loss of connection to the net and life of the process. LRA is the only agent able to get information from the Environment where the MAS is situated.

Process Monitoring Agent. Process Monitoring Agents (PMAs) are in a one-to-one correspondence with LRAs: the PMA associated with process P receives the information sent by the LRA associated with P, looks for anomalies in the functioning of P, reports them to the Computer Monitoring Agent (CMA) and asks it for more information, and in case kills and restarts P if necessary. It implements a sort of social, context-aware, reactive and proactive expert system. PMA can interact with the Environment by killing and restarting the process it monitors.

Computer Monitoring Agent. The CMA receives all the messages arriving from the PMAs that run on that computer, and monitors parameters like network availability, CPU usage, memory usage, hard disk usage. The messages received from PMAs together with the values of the monitored parameters allow the CMA to make hypotheses on the functioning of the computer where it is running. If necessary, the CMA may ask the PlaMA for more information, to know about the state of the entire plant and to act consequently.

Plant Monitoring Agent. There is one Plant Monitoring Agent (PlaMA) for each plant. The PlaMA receives messages from all the CMAs in the plant and in case alerts the SCC Assistance Centre. It interacts with the Environment by alerting the remote assistance centre.

2.3   MAD Agents Implementation

All the agents of the MAS, apart from LRA that is a pure JADE [6] agent, have been implemented in TuProlog [9] integrated into JADE by means of an extended version of DCaseLP libraries [18].

LRAs have been designed and developed as agents for clearly separating what has been developed as part of this project ("agents") from what already existed ("non agents"). We also wanted to emphasise their autonomy (although very limited) and to separate the functionality of parsing the log-file from the one of reasoning over facts. However, LRAs are very trivial agents and we could have designed and implemented them as "Artifacts" in the A&A metamodel [23] or as "Touchpoints" in the Autonomic computing terminology [2] as well.

The CMA, PMA and PlaMA have a cyclic "observe-think-act" behaviour [14] (and a "cyclic behaviour" in JADE) where they

- look if a new message matching a given template has been received;
- retrieve the message from their message queue and store it in their history;
- manage the message according to the rules in their program, and to their knowledge base (that includes all the messages received in the past);
- answer to the agent that has sent the message, and, in case, send messages to other agents in the MAS.

The architecture of each agent, apart LRA ones, is a declarative architecture where the knowledge base is modelled as a set of Prolog facts, the behaviour is determined by Prolog rules, reactivity is implemented by allowing agents to look at their message box and to react to incoming messages. Messages arrive from the LRA to the PMA every m seconds (where m is a configuration parameter of the MAS), and the PMA looks for anomalies and starts the managing process if necessary.

Agents are equipped with different rules dealing with the different parameters to be monitored, namely:

1) parameters tightly connected to the process monitored by the PMA; these parameters include "cpu usage" and "errors" and are not influenced by the state of the network or by other processes;
2) parameters influenced either by the state of the network, or by the behaviour of other processes as those running on the server (for example, "connection to server" and "view").

Parameters of the first type are treated locally by the PMA. Parameters of the second type are dealt with by PMA asking the CMA, which can ask the PlaMA, for more information, since they may involve non-local problems.

The state of an agent consists of a set of facts representing what happened in the past. Different agents store different facts: PMAs store information about what local problems have been found and when (facts reporting a timestamp and what the problem is), CMAs keep information about the problems of all its PMAs and the notifications of a process killing (facts reporting the name of the process, a timestamp and what the problem is and facts reporting why and when a process have been killed), whereas PlaMA records facts about problems in the network (facts reporting the name of the machine and the process, a timestamp and what the problem is), but nothing about the solutions that have been taken (because they are local solutions).

Messages received in previous interactions are also stored by agents in their knowledge bases, since agents may act in different ways if some problem is reported for the first time or if the problem is common to other agents that recently reported it.

This structure allows us to leave the rules that establish how to manage a problem (either kill a process or not, according to the CMA advice) in the PMA, to store the intelligence to monitor a computer and decide when more information is needed in the CMA, and to have the PlaMA look over the whole network and answer CMAs' requests, but without intruding in the local management.

2.4 MAD Agents Execution

In order to run the developed system, JADE and TuProlog (version 1.3, in order to be compliant with the DCaseLP libraries) need to be installed on the machine, as well as the extended DCaseLP libraries. The simplest configuration of the MAS includes

- one PlaMA
- one CMA
- one PMA

but usually the MAS will consist of at least two CMAs controlling different PMAs. At this stage of the project we use more PMAs of the same type, which is not a problem because the rationale is to simulate the behaviour of the CMA with more processes, regardless of their type. The PlaMA is one for each MAS. In the sequel we show the behaviour of the MAS concerning the management of different parameters, and with different configurations and history. Some figures will not show the LRA to let the reader better understand the interactions among the other agents.

The first example shows the behaviour of the MAS when the value "high" of the "cpu usage" is reported by the LRA to the PMA, with the simplest MAS configuration consisting of just one agent of any kind.

When the PMA receives a message from the LRA:
1) If the value of the "cpu usage" parameter is "normal", no action needs to be taken.
2) If the value of the parameter is "high", and it remains high
in the successive message sent by the LRA, the PMA kills and restarts the process, and informs the CMA.

The simplest MAS configuration works well enough to demonstrate this behaviour, because it does not depend on how many PMAs encountered the same problem. As shown in Figure 3, the first message notifying a high cpu usage from the LRA does not cause the delivery of message from the PMA. The second message with the same content, instead, causes the PMA to send a message to the CMA, with the content "process killed".
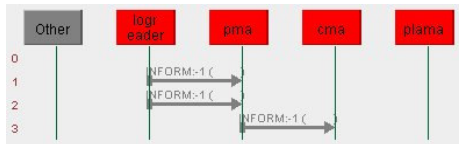

Figure 3. Execution run concerning the "cpu usage" parameter.

The second example shows the behaviour of the system for the management of the "connection to server" parameter. The behaviour is much more complex than the one dealing with the "cpu usage". To allow a good understanding of how it works, we will use two different configurations and histories.
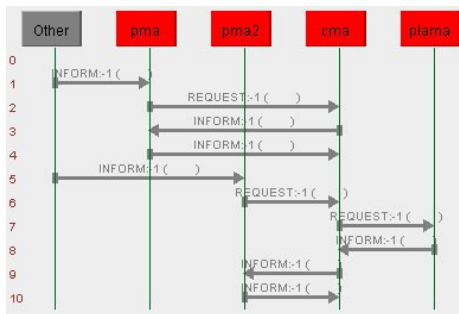

Figure 4. Execution run concerning the "connection to server" parameter.

The first configuration, shown in Figure 4, involves one PlaMA, one CMA and two PMAs, named Pma1 and Pma2. Pma1 receives a message from its LRA with "connection to server(lost)": Pma1 asks for more information to CMA, that has no recent notifications of this problem from other PMAs, and answers "no network problem" to Pma1. Pma1 kills and restarts the process and informs CMA of this. Later on, also Pma2 receives the same message from its

LRA, and, in the same way as Pma1, asks to CMA if the same problem has already been reported. CMA, which had registered the problem of Pma1 in its history, needs to verify if this is a local problem or a problem involving the entire network. Thus, it asks the PlaMA if it is aware of other CMAs with the same problem. For the PlaMA, this is the first notification of the problem so it registers it into its history and answers "no network problem". The CMA forwards the message to Pma2 which kills and restarts the process, and informs CMA of it.
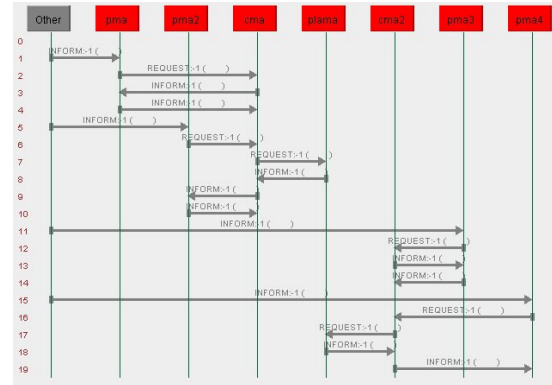

Figure 5. Execution run concerning the "connection to server" parameter, complex configuration.

If we make the configuration even more complex (Figure 5), the behaviour of the MAS changes. We add another CMA named Cma2, controlling two PMAs (Pma3 and Pma4). The agents shown in Figure 4 are still alive and their history includes the events discussed before. If Pma3 receives the notification of the "connection to server(lost)" problem, it reacts exactly as Pma1, and Cma2 acts as Cma1. That is, Cma2 answers to Pma3, without asking the PlaMA, that there are no network problems. But if also Pma4 receives the "connection to server(lost)" message from its LRA, then Cma2 must ask the PlaMA if there are network problems. The PlaMA's history contains the fact that Cma1 reported the same problem a short while ago, so PlaMA sends a message with content "network problem" to Cma2. This answer is propagated to Pma4 by Cma2, and, as a consequence, Pma4 does not kill the process because the problem cannot be managed locally.

3. FYPA

The real application for which the FYPA MAS has been developed is protected by a Non Disclosure Agreement between the Computer Science Department of Genoa University (DISI) and Ansaldo-STS. Thus, in this section we provide a generalization of the problem that we addressed, we show how the complexity of this problem can be profitably faced following an agent-oriented approach, and we discuss the design of the developed MAS maintaining our description at the right level of abstraction. We provide one example of interactions taking place within the MAS and we discuss one screenshot that shows the interactions among the implemented agents.

3.1 The problem faced by FYPA

The problem that the FYPA MAS addresses consists of

- A set of indivisible resources that must be assigned to different entities in different time slots (each resource can be used by only one entity in each time slot).
- A set of entities with different priorities, each needing to use some of the available resources for one or more time slots; entities have preferences over the set of resources they can obtain.
- A directed graph of dependencies among resources: an entity can start using resource R only if it used exactly one resource from $\{R1, R2, ..., Rn\}$ in the previous time slot (we represent these dependencies as arcs $R1 \rightarrow R, R2 \rightarrow R, ..., Rn \rightarrow R$ in the graph).
- A set of resources named "start points" that can be assigned to entities without requiring the prior usage of other resources (no arc enters in the corresponding node).
- A set of resources named "end points" that, once assigned to one entity, allow the entity to complete its job (no arc exits from the corresponding node).
- A set of couples of conflicting arcs in the graph of dependencies: an entity releasing R1 for accessing R2, where the usage of R2 depends on the previous usage of R1, might conflict with an entity releasing R3 for accessing R4. The two entities might indeed need to use the same transportation means for accessing R2 from R1 and R4 from R3 respectively, and the transportation means might be non sharable as well.
- A static allocation plan that assigns resources to entities for pre-defined time slots, in such a way that no conflicts arise.

In an ideal world where resources never go out of order and where any entity in the system can always access the resources assigned to it by the static allocation plan, no problems arise.

In the real world where entities happen to use resources for longer than planned and where resources can break up, a dynamic reallocation of resources over time is often required. Thus, the solution of the real world problem is a dynamic re-allocation of the resources to the entities such that:

- the re-allocation is feasible, namely free of conflicts; in our scenario, conflicts may arise both because two or more entities would want to access the same resource in the same time slot, and because two or more entities would want to use conflicting arcs in the same time slot;
- the re-allocation task is completed within a predefined amount of time;
- each entity minimizes the changes between its new plan and its static allocation plan: the start and end point must always remain those stated in the static allocation plan, but the nodes in between may change, as well as the time slots during which resources are used;

- each entity minimizes the delay in which it reaches the end point with respect to its static allocation plan;
- the number of entities and resources involved in the re-allocation process is kept to the minimum.

We modelled the problem as a directed and non-planar graph that entities must traverse from one start point to one end point. Nodes in the graph are labelled by resources whereas arcs represent dependencies among them. We adopt a discrete and linear time model.

Given this model, the problem to solve can be stated as:

For each entity that enters the graph from a start point either confirm the validity of the plan stated in the static allocation plan, or, if some unexpected event occurred that makes the original plan no longer applicable, find a new plan for reaching an end point of the graph. The new plan should minimize the delay in which the entity exits the graph and the number of required changes with respect to the original plan, as well as the number of entities involved in the re-allocation process.

3.2 FYPA Architecture

In order to face the dynamic resource allocation problem described in Section 3.1 we designed three types of agents each with its own capabilities and view of the system: Resource Agents (RAs), User Agents (UAs), Interface Agents (IAs). The graph becomes the Environment where agents live. There is no central control of the state of the graph, which is indeed spread all over the RAs.

Resource Agent. Each node in the graph is managed by one RA. RAs do not take decisions about which UA will obtain the control of the node but keep track of the node's state (free/occupied). RAs also manage the allocation of arcs entering the node.

UAs interact with RAs for knowing whether the node is free or occupied in a given time slot. RAs answer the question and, in case the node is not free, tell which UA occupies the node for the given time slot, its priority, and when the node will become free again.

RAs also manage the allocation of the arcs incoming into the node they control. The RA controlling node N has the list of all its neighbours, namely those RAs controlling nodes N^{from} such that an arc $N^{from} \rightarrow N$ exists. For each arc $N^{from} \rightarrow N$, the RA also possesses the list of arcs $A1, ..., Ak$ that conflict with $N^{from} \rightarrow N$.

When the RA receives a reservation request for node N and chooses the free arc $N^{from} \rightarrow N$ to reach it, it updates its reservation table by marking the arc as occupied, answers the request, and informs all the RAs that may be interested by this reservation, namely those controlling arcs $A1, ..., Ak$, about the new state of the arc.

These RAs need to know that arcs conflicting with $N^{from} \rightarrow N$ can not be used for the specified time slot. In this way, the neighbours of RA have up-to-date information about the

state of the arcs that might cause conflicts with their own arcs, and will be able to provide conflict-free answers to successive reservation requests.

User Agent. UAs represent entities that want to traverse the graph. Each UA has an original plan stated in the static allocation plan and consisting of the list of nodes to traverse together with arrival and departure time for each of them. As soon as an UA enters the graph, no matter if it is on time or late, it always tries to get a reservation for the nodes in its original path. UAs do not try to reserve a specific arc to reach a node: they ask RAs to reserve the most suitable arc for them.

UAs do not know the topology of the graph; they may interact with the Path Agent introduced later on in this section to obtain information on the paths that connect the start point where they enter the graph with the end point they must reach to exit the graph.

UAs communicate with RAs to reserve resources. Only in one case UAs may communicate with each other.

Every UA has a priority that it uses to reserve a resource or even to steal a resource to another UA. In case of theft of a resource, the UA victim of the theft may directly interact with the thief as discussed later on.

Since each UA has the unique goal of getting out of the graph, it continues to look for a path in it until it obtains the reservation for all the nodes in the path.

If it looses the reservation of one of these nodes, for example because a UA with higher priority stole the node to it, it will start the search again until it will succeed in reserving all the nodes in one path.

Interface Agent. IAs act as an interface between the MAS and the external environment. There are three types of IAs:

- The **Path Agent** (PA) provides an interface between agents in the MAS (in particular, UAs) and the Path Finder Service offered by a software module external to the MAS. The Path Finder Service exploits its knowledge about the graph topology and geometry. Given two nodes, it returns a list of selected paths connecting them ordered from the best one to the worst one. The strategy for selecting and ordering the paths depends on the application. In Ansaldo-STS's application, it depends on the number of nodes in the path and on geometrical constraints. If a UA wants to pass through a particular node that we name "parking node", it asks the PA to look for a path satisfying this requirement. The PA uses this additional parameter to query the Path Finder Service and to obtain the list of all the paths that include the parking node.
- The **RA Manager** reads the structure of the graph from a configuration file that includes real data and creates the RAs corresponding to the nodes, equipped with all the information they need.
- The **UA Manager** creates the UAs that enter the graph according to a configuration file and taking the real data on the agents' delay into account.

3.3 Interactions among FYPA agents: one example

In this section we provide one example of interactions taking place among agents in the FYPA MAS. In this example, a conflict over a resource arises.
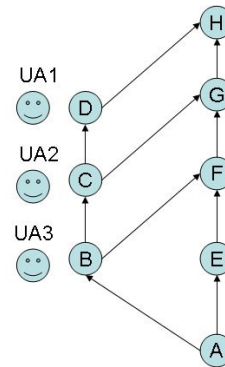

Figure 6. The Graph

The graph that we use in our first example is represented in Figure 6. There are three UAs, UA1, UA2 and UA3, with the same start and end nodes, A and H respectively.

Figure 7 shows how the resources (on the rows) are allocated to the UAs in the time (columns) according to the static allocation plan.

	T1	T2	T3	T4	T5	T6	T7
A	UA2	UA3					
B	UA1	UA2	UA3				
C		UA1	UA2	UA3	UA3		
D			UA1	UA2	UA2	UA3	
F							
G							
H				UA1		UA2	UA3

Figure 7. The static allocation plan.

Note that Figure 7 shows time starting from T1, which is the time slot when UA2 enters the graph, but there were other time slots before T1. For example, UA1 entered the graph at time T0 which is not shown in the figure.

Let us suppose that the current time slot is either T1 or T2, and that UA1 realizes that it must stop on D not only for the time slot T3 as the original allocation plan states, but also for T4 and T5, as it has no means to move from D until T6.

UA1 sends a "non disputable" reservation (namely, a reservation that the RA is forced to accept) for the time slots T3, T4 and T5 to the RA controlling D, that we name D (for sake of readability, we identify the RAs with the names of the resources they control, which also label the nodes in the graph). D has already a reservation from UA2 for time slots

T4 and T5, but, since UA1's reservation is "non disputable", it sends a "cancel" message to UA2 specifying that UA1 stole D for time slots T4 and T5 with a "non disputable" reservation.

UA2 has to change its original path for reaching H:
Case 1: if the delay that it would undergo if it stopped on C for T4 and T5 (waiting for UA1 to release D) is acceptable, it can accept to stop on C for T4 and T5 and then move to D; otherwise
Case 2: it must look for a new path.

Case 1. In the first case, UA2 sends a request to D for T6 and T7, and a request to C for T4 and T5. Both C and D answer with an "occupied" message, specifying that the corresponding resource is occupied by UA3 that has priority P3. UA2 has priority P2 and will behave in two different ways according to the relationships between P2 and P3:

Case 1.1: if P2 >= P3, UA2 will reserve anyway the resources, stealing them to UA3;

Case 1.2: if P2 < P3, UA2 must look for a new path.

	T1	T2	T3	T4	T5	T6	T7	T8
A	UA2	UA3						
B	UA1	UA2	UA3					
C		UA1	UA2	UA2	UA2			
D			UA1	UA1	UA1	UA2	UA2	
F				UA3	UA3			
G						UA3		
H						UA1	UA3	UA2

Figure 8. New plan under hypothesis P2 >= P3.

In the first case (1.1), UA3 will search for a new path because it has lost C and D: let us suppose that it is B → F → G → H. UA3 will send the requests to the RAs in charge for the resources in this path. It will succeed in reserving them and it will exit the graph in time slot T7 (Figure 8).

	T1	T2	T3	T4	T5	T6	T7	T8
A	UA2	UA3						
B	UA1	UA2	UA3					
C		UA1	UA2	UA3	UA3			
D			UA1	UA1	UA1	UA3		
F								
G				UA2	UA2			
H						UA1	UA3	UA2

Figure 9. New plan under hypothesis P2 < P3.

In the second case (1.2), UA2 will use the path C → G → H

and will succeed in exiting the graph at T8, while the reservations of UA3 will not be affected (Figure 9).

Case 2. Case 2, namely the case where UA2 should gain too much delay if stopping on C for T4 and T5, has the same solution as case 1.2: UA2 will use G and H, in the same time slots as shown in Figure 9.

3.4 FYPA implementation and execution

The FYPA MAS has been implemented with JADE and uses the JADE Web Services Integration Gateway, WSIG [37] for interfacing with applications outside the MAS. Due to the confidentiality of the project we can not go into the details of the implementation. We limit ourselves to showing a screenshot of the JADE Sniffer Agent taken during one of our tests or real data provided by Ansaldo-STS (Figure 10).

In this screenshot we see UA1 that enters the graph in node 1 managed by RA1 and wants to move to node 2 managed by RA2 and to exit in node 3 managed by RA3.
As soon as the UA Manager creates UA1, UA1 sends a "query-if" to the three RAs in order to reserve the resources it needs.
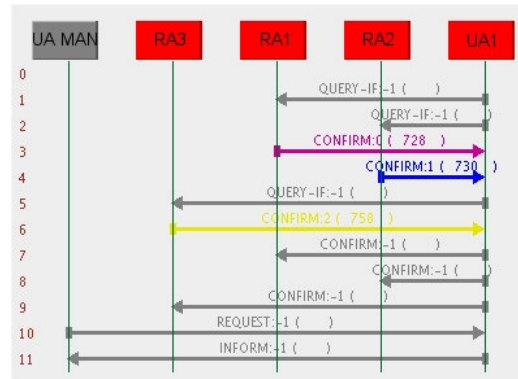

Figure 10. FYPA MAS execution.

After receiving all the three messages from the RAs with "confirm" performative, UA1 replies with three confirmations in order to reserve the path.
Finally the UA Manager sends an information request about the reserved path to UA1, and UA1 informs it of the path that it just reserved.

4. Related work and conclusions

The exploitation of intelligent agents for monitoring, diagnosing and solving problems in complex, distributed systems has a long and successful history dating back to the early and mid nineties.

Before that, Distributed Artificial Intelligence (DAI) techniques were adopted. Even if the first DAI systems did not integrate "agents" as we intend them today, they were

the ancestors of MASs and deserve to be shortly mentioned in this section.

In 1990, the "Large-internetwork Observation and Diagnosis Expert System", LODES [29], was implemented. It represents an interesting example of application of DAI to diagnosis and problem solving. The diagnostic system was created by reusing and unifying pre-existing network diagnosis expert systems. Each sub-LAN had its own LODES system, and problems were solved by their co-operative work. In the same year, Weihmayer and Brandau developed TEAM-CPS [30], a test bed for introducing DAI to control and manage customer networks: in TEAM-CPS the customers' virtual private networks were automatically reconfigured using links from the public network. In 1992, the "Distributed Big Brother" was one of the earliest works where DAI was adopted for monitoring purposes in the telecommunications area [28]. The project applied DAI techniques to Local-Area Networks, to make their management more robust and faster.

Among the oldest applications of rule-based intelligent agents in the monitoring and diagnosis domain we may mention a re-implementation of TEAM-CPS [31] where agents used the PRODIGY planning system [20] for local network planning, and the well-known Agent-Orientated Programming framework [27] for communication and control. In 1997, Leckie et al. [15] developed a prototype agent-based system for performance monitoring and fault diagnosis in a telecommunications network, where agents were implemented using C5 [24], based on the OPS5 rule language [11], and communicated using KQML [10].

An architecture for a software agent operating a physical device and capable of testing and repairing the device's components is described in [3]. In that work, the authors focus on modelling the agent's behaviour after the discovery of a fault in a circuit: the knowledge as well the behaviours of the agent are expressed in A-Prolog [4]. The life of the agent is an "observe-think-act" loop where actions are quite simple, but nevertheless able to modify the circuit in order to repair it. An industrial application of A-Prolog to a medium size knowledge-intensive application for controlling some functions of the Space Shuttle is described in [21]. However, no agents are used there.

Moving to nowadays, [26] describes Space Shuttle Ground Processing with Monitoring Agents. JESS [38] is used to realize a system that helps the monitoring of all the processes, instrumentation and data flows of the Kennedy Space Centre's Launch Processing System. The system, called NESTA, helps to monitor and above all to discover problems concerning the "ground process", i.e. the set of the operations carried out in the weeks before the Space Shuttle's launch. NESTA autonomously and continuously monitors shuttle telemetry data and automatically alerts NASA shuttle engineers if it discovers predefined situations. This system, developed and tested in a real, safety-critical scenario, shows that an agent-oriented solution implemented with a rule-based language may be employed to satisfy concrete industrial needs, and demonstrates the success of agents outside the boundaries of academia.

Other applications of agents for diagnosis and monitoring include [16] that presents a technique for monitoring the start up sequences of gas turbine: the system uses a MAS where decisions are taken by combining partial information possessed by individual agents, thus obtaining a global view of the situation, and producing an automatic fault diagnosis for the engineers. The MAS is implemented with the ZEUS Agent Building Toolkit [22]. In 2006, the Rockwell Automation company applied agents to control manufacturing production [19]. The MAS is implemented with real-time control agents, and also the information transfer among the software agents takes place in real-time, using a Programmable Logic Controller. A MAS for the simulation of the environment for material handling systems has been implemented in JADE.

In [42], the integration of intelligent anomaly agents and traditional monitoring systems for high-performance distributed systems is discussed. The intelligent agents presented in that study employ machine learning techniques to develop profiles of normal behavior as seen in sequences of operating system calls (kernel-level monitoring) and function calls (user-level monitoring) generated by an application. The Ganglia distributed monitoring system (developed by Massie, Chun and Culler at the University of California, Berkeley [43]) was used as a test bed for integration case studies. Mechanisms provided by Ganglia make it relatively easy to integrate anomaly detection systems and to visualize the output of the agents. The results provided demonstrate that the integrated intelligent agents can detect the execution of unauthorized applications and network faults that are not obvious in the standard output of traditional monitoring systems.

Finally, [7] describes a model for managing faults in industrial processes. The model is based on a generic framework that uses MASs for distributed control systems; the system manages faults with feedback control process and decides about the scheduling of the preventive maintenance tasks, also running preventive and corrective specific maintenance tasks.

As far as distributed problem solutions emerging from negotiation, which is the research activity we carried out in the FYPA project, some proposals exist [39], [40], [41] and, among them, some negotiation algorithms similar to the one that we implemented in FYPA exist. However, changing even only one assumption, requirement or constraint in the negotiation algorithm, leads to very different results. Thus, an algorithm "similar" but not identical to the one we needed was not suitable for our FYPA project. To the best of our knowledge, no negotiation algorithm similar to the one we needed is available to the research community as a source code. Thus, even if we had found the right algorithm for FYPA in the literature, the burden of implementing it would have been up to us in any case. Also, we developed the FYPA MAS having a real application in mind. We could not neglect all the constraints on the software environment where our MAS would be integrated. For these reasons we decided to design and implement our own negotiation algorithm.

The MAD Agents and FYPA projects, although similar in their purposes to other applications ([25]) developed in the past, demonstrate an increased industrial interest and trust in both agent-based and, as far as MAD Agent is concerned, rule-based technologies.

To the best of our understanding only few proposals of using rule-based agents led to the development of a MAS prototype used inside an industry ([12], [8], [26]). The industrial strength system described in [19], despite not using rule-based technologies, shares with our projects the choice of JADE as the agent middleware.

The Agent Technology Roadmap [33] observed that *"One of the most fundamental obstacles to the take-up of agent technology is the lack of mature software development methodologies for agent-based systems."*. According to the experience of DISI and Ansaldo-STS, agent tools, languages and methodologies are today mature enough to be adopted by the industry. Although the competencies on how to exploit them are still missing in many companies, companies now know that agents exist, believe in their usefulness for coping with the complexity of open, distributed, dynamic applications, and are more and more keen on integrating them into their projects. The role of academia in providing a good support during the design and implementation of MASs for real applications is a key factor in the take-off of the agent technology, and the joint DISI-Ansaldo-STS projects discussed in this paper represent two success stories in this direction.

Acknowledgement

References

[1] E. Abel, I. Laresgoiti, J. Perez J., Corera, and J. Echavarri. A multi-agent approach to analyse disturbances in electrical networks. In International Conference on Expert Systems Applications to Power Systems, ESAP'93, Proceedings, 1993.

[2] B. A. Miller. http://www.ibm.com/developerworks/autonomic/library/ac-edge5/, 2005.

[3] M. Balduccini and M. Gelfond. Diagnostic reasoning with a-prolog. Theory Pract. Log. Program., 3(4):425–461, 2003.

[4] M. Balduccini, M. Gelfond, and M. Nogueira. A-prolog as a tool for declarative programming. In 12th International Conference on Software Engineering and Knowledge Engineering, SEKE'00, Proceedings, pages 63–72. Knowledge Systems Institute, 2000.

[5] J. Barandiaran, I. Laresgoiti, J. Perez, J. Corera, and J. Echavarri. Diagnosing faults in electrical networks. In S. Hashemi, J.P. Marciano, and J.G. Gouarderes, editors, International Conference on Expert Systems

Applications, EXPERSYS'91, Proceedings. IITT Paris, 1991.

[6] F. L. Bellifemine, G. Caire, and D. Greenwood. Developing Multi-Agent Systems with JADE. Wiley, 2007.

[7] M. Cerrada, J. Cardillo, J. Aguilar, and R. Faneite. Agents-based design for fault management systems in industrial processes. Computers in Industry, 58(4):313–328, 2007.

[8] J. M. Corera, I. Laresgoiti, and N. R. Jennings. Using Archon, part 2: Electricity transportation management. IEEE Expert, 11(6):71–79, 1996.

[9] E. Denti, A. Omicini, and A. Ricci. tuProlog: A lightweight prolog for internet applications and infrastructures. In I. V. Ramakrishnan, editor, 3rd International Symposium on Practical Aspects of Declarative Languages, PADL'01, Proceedings, pages 184–198. Springer, 2001.

[10] T. W. Finin, R. Fritzson, D. P. McKay, and R. McEntire. KQML as an agent communication language. In 3rd International Conference on Information and Knowledge Management, CIKM'94, Proceedings, pages 456–463. ACM, 1994.

[11] C.L. Forgy. Ops5 user's manual. Technical Report CMU-CS-81-135, Carnegie-Mellon University, 1981.

[12] N. R. Jennings, E. H. Mamdani, J. M. Corera, I. Laresgoiti, F. Perriollat, P. Skarek, and L. Zsolt Varga. Using Archon to develop real-world DAI applications, part 1. IEEE Expert, 11(6):64–70, 1996.

[13] N. R. Jennings, K. P. Sycara, and M. Wooldridge. A roadmap of agent research and development. Autonomous Agents and Multi-Agent Systems, 1(1):7–38, 1998.

[14] R. Kowalski and F. Sadri. From logic programming towards multi-agent systems. Annals of Mathematics and Artificial Intelligence, 25(3-4):391–419, 1999.

[15] C. Leckie, R. Senjen, B. Ward, and M. Zhao. Communication and coordination for intelligent fault diagnosis agents. In 8th IFIP/IEEE International Workshop for Distributed Systems Operations and Management, DSOM'97, Proceedings, pages 280–291, 1997.

[16] E.E. Mangina, S.D.J McArthur, J.R. Mc Donald, and A. Moyes. A multi agent system for monitoring industrial gas turbine start-up sequences. IEEE Transactions on Power Systems, 16(3):396–401, 2001.

[17] V. Mascardi, D. Briola, M. Martelli, R. Caccia, and C. Milani. Monitoring and diagnosing railway signalling with logic-based distributed agents. In E. Corchado and R. Zunino, editors, International Workshop on Computational Intelligence in Security for Information Systems, CISIS'08, Proceedings, Advances in Soft Computing Series. Springer-Verlag, pages 108–115, 2008.

[18] V. Mascardi, M. Martelli, and I. Gungui. DCaseLP: a prototyping environment for multi-language agent systems. In M. Dastani, A. El-Fallah Seghrouchni, J. Leite, and P. Torroni, editors, In Proceedings of the First Workshop on LAnguages, methodologies and Development tools for multi-agent systemS, LADS'007

Post-proceedings, volume 5118 of LNCS, pages 139–155. Springer-Verlag, 2008.

[19] V. Mařík, P. Vrba, K. H. Hall, and F. P. Maturana. Rockwell automation agents for manufacturing. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, editors, 4rd International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS'05, Proceedings, pages 107–113. ACM, 2005.

[20] S. Minton, C. A. Knoblock, D. R. Kuokka, Y. Gil, R. L. Joseph, and J. G. Carbonell. Prodigy 2.0: The manual and tutorial. Technical Report CMU-CS-89-146, Carnegie-Mellon University, 1989.

[21] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. An a-prolog decision support system for the space shuttle. In A. Provetti and T. Cao Son, editors, 1st International Workshop on Answer Set Programming, Towards Efficient and Scalable Knowledge Representation and Reasoning, ASP'01, Proceedings, 2001.

[22] H. Nwana, D. Ndumu, L. Lee, and J. Collis. ZEUS: A tool-kit for building distributed multi-agent systems. Applied Artifical Intelligence Journal, 13(1):129–186, 1999.

[23] A. Ricci, M. Viroli, and A. Omicini. The A&A programming model and technology for developing agent environments in MAS. In M. Dastani, A. El Fallah-Seghrouchni, A. Ricci, and M. Winikoff, editors, Programming Multi-Agent Systems, 5th International Workshop, ProMAS 2007, Proceedings, pages 89–106, volume 4908 of LNCS. Springer, 2008.

[24] J.R. Roland, G.T. Vesonder, and J.M. Wilson. C5 user manual, release 2.1. Technical report, AT&T Bell Laboratories, 1990.

[25] M. Schroeder, I. de Almeida Móra, and L. Moniz Pereira. A deliberative and reactive diagnosis agent based on logic programming. In M. P. Singh, A. S. Rao, and M. Wooldridge, editors, 4th International Workshop on Agent Theories, Architectures, and Languages, ATAL'97, Proceedings, volume 1365 of LNCS, pages 293–307. Springer, 1998.

[26] G. S. Semmel, S. R. Davis, K. W. Leucht, D. A. Rowe, K. E. Smith, and L. Boloni. Space shuttle ground processing with monitoring agents. IEEE Intelligent Systems, 21(1):68–73, 2006.

[27] Y. Shoham. Agent-orientated programming. Artificial Intelligence, 60(1):51–92, 1993.

[28] Y. So and E. H. Durfee. A distributed problem-solving infrastructure for computer network management. Int. J. Cooperative Inf. Syst., 2(2):363–392, 1992.

[29] T. Sugawara. A cooperative lan diagnostic and observation expert system. In 9th Annual International Phoenix Conference on Computers and Communications, PCCC'94, Proceedings, pages 667–674. IEEE, 1990.

[30] R. Weihmayer and R. Brandau. A distributed ai architecture for customer network control. In IEEE Global Telecommunications Conference, Globecom'90, Proceedings, pages 656–662. IEEE, 1990.

[31] T. Weihmayer and M. Tan. Modeling cooperative agents for customer network control using planning and agent-oriented programming. In IEEE Global Telecommunications Conference, Globecom'92, Proceedings, pages 537–543. IEEE, 1992.

[32] N. Jennings, Cooperating Agents for Industrial Process Control, http://users.ecs.soton.ac.uk/nrj/download-files/archon/arch10.html

[33] M. Luck, P. McBurney, O. Shehory, S. Willmott, and the AgentLink Community, Agent Technology: Computing as Interaction – A Roadmap for Agent-Based Computing, AgentLink III, 2005.

[34] M. Wooldridge and N. R. Jennings. Intelligent Agents: Theory and Practice, The Knowledge Engineering Review, 10(2):115–152, 1995.

[35] D. Briola, V. Mascardi, M. Martelli, G. Arecco, R. Caccia, C. Milani. A Prolog-Based MAS for Railway Signalling Monitoring: Implementation and Experiments. In Atti del Workshop Dagli Oggetti agli Agenti, WOA'08, M. Baldoni, M. Cossentino, F. De Paoli, V. Seidita eds., Seneca Edizioni, 2008.

[36] R.G. Smith, The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver, IEEE Transactions on Computers, 29(12):1104-1113, 1980.

[37] JADE Board. Jade web services integration gateway (WSIG) guide. http://jade.cselt.it/doc/tutorials/WSIG_Guide.pdf, accessed on June, 10th, 2009.

[38] JESS, the Rule Engine for the Java Platform, http://www.jessrules.com/, accessed on June, 10th, 2009.

[39] Y. Chevaleyre, P. E. Dunne, U. Endriss, J. Lang, M. Lemaître, N. Maudet, J. Padget, S. Phelps, J. A. Rodríguez-Aguilar, and P. Sousa. Issues in multiagent resource allocation. Informatica, 30:3–31, 2006.

[40] Multi-Agent Systems Lab, Department of Computer Science at the University of Massachusetts at Amherst. Publications on negotiation in multi-agent systems. http://mas.cs.umass.edu/pub/search.html?search=true&author=&title=&year=&keywords%5B%5D=Negotiation, accessed on June, 10th, 2009.

[41] Stanford AI Lab, Stanford Computer Science Department. Publications on game theory and multi-agent systems. http://robotics.stanford.edu/~shoham/YoavPublications.htm, accessed on June, 10th, 2009.

[42] G. Florez-Larrahondo, Z. Liu, Y. S. Dandass, S. M. Bridges, and R. Vaughn. Integrating Intelligent Anomaly Detection Agents into Distributed Monitoring Systems. Journal of Information Assurance and Security 1(1):59–77, 2006.

[43] M. L. Massie, B. N. Chun, D. E. Culler, The Ganglia distributed monitoring system: Design, implementation, and experience, Parallel Computing. 30(7):817–840, 2004.

Author Biographies

Daniela Briola was born in Genoa, Italy. She is a Ph.D. student at the Computer Science Department at University of Genoa, Italy. Her research interests include multi-agent systems, negotiation in MAS, logic programming languages and software engineering for MAS. In these areas she published, together with the other authors of this paper, some papers presented in national and international workshops. Daniela has one degree in computer science obtained from the University of Genoa.
Contact her at daniela.briola@unige.it.

Viviana Mascardi was born in Genoa, Italy. She is an assistant professor in the Computer Science Department at University of Genoa. She co-organised national and international workshops on intelligent agents and ontologies. She co-authored more than 50 publications on agent-oriented software engineering (in particular, modelling, verification, rapid prototyping), on development of platforms for specification and implementation of MASs, on languages for intelligent software agents, and on semantic interoperability. She holds a Ph.D. degree in computer science obtained from the University of Genoa in 2002. Contact her at viviana.mascardi@unige.it.

Maurizio Martelli was born in Pisa, Italy. He graduated with honours in Computer Science at the University of Pisa in 1974 and he is now full professor of Computer Science and Artificial Intelligence in the Genoa University. He is Vice-Rector of the University of Genoa. He has been Dean of the Faculty of Science of the University of Genoa during the period 2005-2008. His research interests include from the theoretical point of view the use of Higher Order Linear Logic in Logic Programming and from the applicative point of view the use of LP techniques in the specification and rapid prototyping of Intelligent Multi-Agent Systems, Ontologies and Software Engineering.
Contact him at maurizio.martelli@unige.it.