A Prolog-Based MAS for Railway Signalling Monitoring: Implementation and Experiments

Daniela Briola†, Viviana Mascardi†, Maurizio Martelli†,
Gabriele Arecco†, Riccardo Caccia‡, Carlo Milani‡
† DISI, Università degli Studi di Genova,
Via Dodecaneso 35, 16146, Genova, Italy
{Daniela.Briola, Viviana.Mascardi, Maurizio.Martelli}@unige.it, gabriele.arecco@gmail.com
‡ IAG/FSW, Ansaldo Segnalamento Ferroviario S.p.A., Italy
{Caccia.Riccardo, Milani.Carlo}@asf.ansaldo.it

***Abstract*—This paper describes the outcomes of a project that involved DISI, the Computer Science Department of Genoa University, and Ansaldo Segnalamento Ferroviario, the Italian leader in design and construction of signalling and automation systems for conventional and high speed railway lines. The result of the project, started in February 2008 and ended in September 2008, is an implemented MAS prototype that monitors processes running in a railway signalling plant, detects functioning anomalies, and provides support to the early notification of problems to the Command and Control System Assistance. The MAS has been implemented using DCaseLP, a multi-language prototyping environment developed at DISI, that provides libraries for integrating TuProlog agents into Jade. Due to the intrinsic rule-based nature of monitoring agents, Prolog has been proved extremely suitable for their implementation.**

I. Introduction

Distributed diagnosis and monitoring represent one of the oldest application fields of rule-based software agents.

ARCHON (ARchitecture for Cooperative Heterogeneous ON-line systems [12]) was Europe's largest ever project in the area of Distributed Artificial Intelligence. It was employed for monitoring and controlling the cycle of generating, transporting and distributing electrical energy to industrial and domestic customers, for the Iberdrola company, one of the world's leading private energy groups [8]. ARCHON's Planning and Coordination Module was implemented as a rule-based system.

In [25], Schroeder et al. describe a declarative and reactive diagnostic agent based on extended logic programming. Both the inference engine used for computing diagnoses and the reactive layer that implements a meta-interpreter for the agent were implemented in Prolog extended with communication facilities.

Both ARCHON and Schroeder's agent systems date back to more than ten years ago. In the meanwhile, a large number of MASs for diagnosis and monitoring has been developed, many of them based on rule-based approaches.

There are many good reasons for choosing a MAS approach to process diagnosis and monitoring. Some of them had been clearly stated by N. Jennings[1]:

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

- *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, 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, and reporting aggregated information and diagnosis to the agents higher in the hierarchy.
- *To enable a number of different problem solving paradigms to be utilised*: Rephrasing Jennings' considerations,

  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 process diagnosis and monitoring domain: *economy*, *robustness*, *reliability*, *natural representation of the domain*.

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 [13]. When combined with reactivity, situatedness may lead to the early detection of, and reaction to, anomalies.

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

The simplest and most natural form of reasoning for producing diagnoses starting from observations is rule-based. Also, a monitoring activity may be profitably modelled by means of reactive rules. If the agents employed in the MAS are implemented in a rule-based language, the implementation of a rule-driven reasoning mechanism is greatly simplified.

This paper describes the results of a joint academy-industry project started at the beginning of 2008. The project involves the Computer Science Department of Genoa University, Italy, and Ansaldo Segnalamento Ferroviario, a company of Ansaldo STS group controlled by Finmeccanica, the Italian leader in design and construction of railway signalling and automation systems.

The outcome of the project is a MAS prototype that monitors an Ansaldo process, which controls railway signalling, and reacts to anomalies either by interacting with other agents in the MAS or by killing the process that raised the anomaly. The MAS has been implemented in Jade [6] extended with TuProlog [9] by means of the DCaseLP libraries [18].

At this stage of the project, the MAS is running in an "off-line" modality: agents are not installed on machines in Ansaldo and the MAS tests have been carried out at DISI. Agents read original log-files provided by Ansaldo as if new lines were added by the monitored process once every m minutes. Agents act in accordance to the content of the last lines read, thus simulating an "on-line" reading phase. Also the "kill process" action is just simulated at the time of writing. When Ansaldo will fully integrate the MAS into its system, log-files will be read in real-time as they are produced by the monitored process. Furthermore, the MAS will be allowed to really kill processes and to contact the Assistance Centre of the Command and Control System for Railway Circulation to report the anomalies in an automatic way. When the MAS will be installed on the Ansaldo SCC system, other ways to manage processes, aside from the "kill process", will be studied.

The paper is structured in the following way: Section II describes the operating scenario and the MAS architecture, Section III describes the rule-based implementation of the agents, Section IV shows the potential of the system by discussing different execution runs. Section V overviews the related work and concludes.

II. Operating Scenario and MAS Architecture

The architecture of the MAS and its operating scenario have been extensively described in [17]. In this section we briefly recall them to allow the reader to understand the original contribution of this paper, namely the system implementation and execution described in Sections III and IV.

A. 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.

The MAS we have implemented 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 (Figure 1).

One of the 16 user workstations of Genoa's SCC is shown in Figure 2. 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

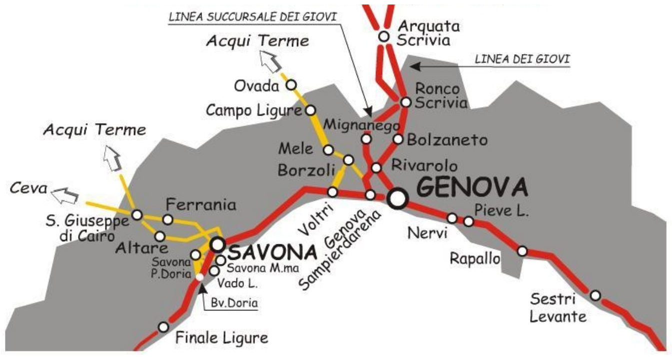

Figure 1. Railway tracks controlled by Genoa's SCC.

Figure 2. Operator and synoptic frame in Genoa's SCC.

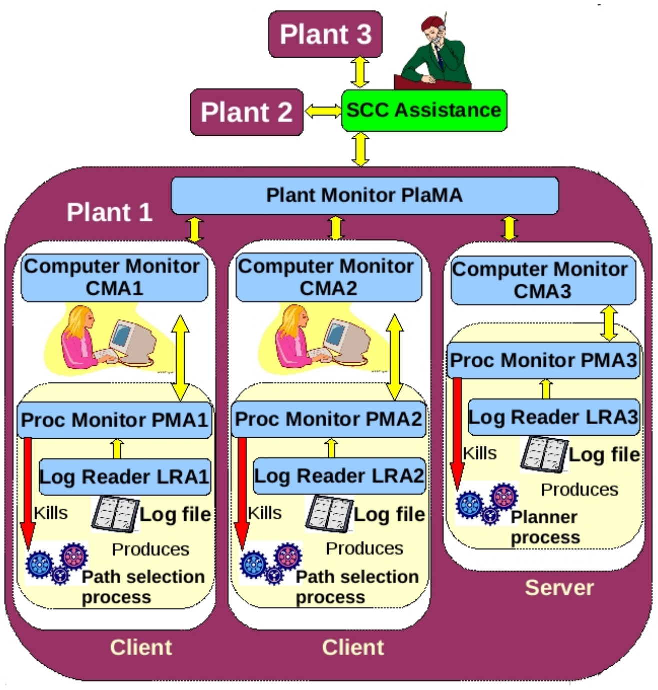

Figure 3. MAS architecture, from [17].

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.

B. MAS architecture

Our MAS consists of the four kinds of agent depicted in Figure 3.

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, 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.

III. 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]. The extension consists in making a *blocking_selective_receive* predicate available to the agents, which takes three arguments: *Performative*, *Content*, *Sender*. It was motivated by the need to allow our agents to retrieve only messages respecting a given pattern (in particular, messages arriving from a given *Sender*) from their message queue. Jade offers the *MessageTemplate* class that provides static methods to create filters for each attribute of the *ACLMessage*. The Jade *blockingReceive* method can accept a message template as argument, and retrieve only those messages that match the template. The DCaseLP *blocking_selective_receive* predicate creates a template that filters on the name of the *Sender*, and then calls the Jade *blockingReceive(mt)* to return the value for *Performative* and *Content*.

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 modeled as a set of Prolog facts, the behavior 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.

An example of message sent by the LRA to the PMA is: `log(time(``Mon Feb 11 21:30:43 CET 2008''), [view(normal), cpu_usage(normal), connection_to_server(active), disk_usage(normal), answer_to_life(slow), errors(absent), memory_usage(normal)])`, whose meaning is easy to understand.

Currently the agents do not use a common ontology, that is implicitly known as the set of the monitored parameters and their possible values.

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 (kill o 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.

In the sequel we show some schemes of interaction protocols among agents aimed at managing some parameters. The translation from these schemes into Prolog code has been done creating and distributing rules among the agents involved in the interaction. We did not use any standard interaction protocol: indeed, we designed the needed interaction protocols on an application-driven basis.

For example, in order to manage the "connection_to_server" parameter, the MAS acts in the following way:

1) If the value of the "connection_to_server" parameter received by the PMA from the LRA is "active", no action has to be taken; instead
2) if the value received by the PMA from the LRA is "lost", the PMA asks the CMA to know if this problem is

common to other processes or not.

a) If the CMA has no recent information[2] about this problem in its history, it notifies the PMA that the problem is not a "net problem" (that is, it is not common to other processes); in this case, the PMA kills and restarts the process, and informs CMA of this.
b) If the CMA has other (one or more) recent notifications of the problem, before answering to the PMA, it asks the PlaMA to know if the problem is local to the machine where the CMA runs, or has been reported also on other computers.
    i) If the PlaMA received no notifications of this problem from other CMAs recently, it answers that the problem is not common to the MAS; in this case, the CMA answers the PMA that there are no net problems, and the PMA kills and restarts the process and informs CMA of this. Otherwise,
    ii) if the PlaMA already received notifications of the same problem in the last M minutes, it answers the CMA that there are network problems; the CMA forwards this answer to the PMA, which, in this case, does not kill the process.

There are also situations where the PMA waits for two (or more) consecutive messages from the LRA notifying the same problem, before reporting it to the CMA. This situation is shown below, for the parameter "answer_to_life":

1) If the value of the "answer_to_life" parameter received by the PMA from the LRA is "ready", no action has to be taken.
2) If the value received by the PMA from the LRA is "absent", the PMA kills and restarts the process (and informs CMA).
3) If the value received by the PMA is "slow", the PMA must wait for the successive message arriving from the LRA. If the successive message reports again a "slow" answer to life, then the PMA must ask the CMA to know if this problem is common to other processes or not.
    a) If the CMA received no recent notifications of this problem, it notifies the PMA that the problem is not a "net problem" (that is, it is not common to other processes). In this case, the PMA waits for two more messages from the LRA and, if the problem persists, kills and restarts the process and notifies it to CMA.
    b) If the CMA received recent notifications of the same problem, it asks the PlaMA if the problem is local to the machine or had also been reported on other computers.
        i) If the PlaMA answers that the problem is not common to the MAS, since no notifications of the same problem have been reported in the last M minutes, the CMA sends a "no net problem". The PMA waits for two next messages and kills and restarts the process (and notifies CMA), if the problem persists.
        ii) If the PlaMA was recently notified of the same problem, it answers the CMA that there are network problems; this answer is forwarded by the CMA to the PMA, that does not kill the process.

[2]By recent information, we mean information stored no later than M minutes ago, where M is a parameter that can be set by the person in charge of configuring the MAS.

The hierarchical structure of the system ensures scalability: there will always be only one PlaMA in the MAS, but many CMAs may be connected to it, and many PMAs can be started on a machine and controlled by the local CMA. In case other PMAs should be developed in the future, with rules ad hoc for different processes, only the new PMAs and their associated LRAs should be developed from scratch, with no impact on the entire system.

IV. Running the system

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 4, 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 5 depicts the behaviour of the PMA with respect to the "answer_to_life" parameter. In this configuration we have only one CMA. The PMA will wait for two messages

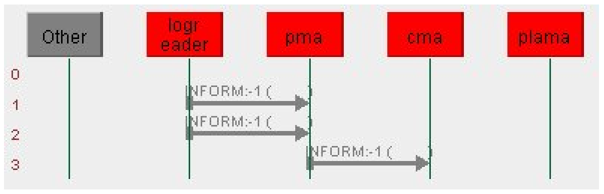

Figure 4. Execution run concerning the “cpu_usage” parameter.

from the LRA with the value “slow” for the parameter, then it contacts the CMA for further information: CMA received no recent information from other PMAs, so PMA kills (after two messages from LRA with the same problem) the monitored process and informs of this the CMA, as described in the previous section.

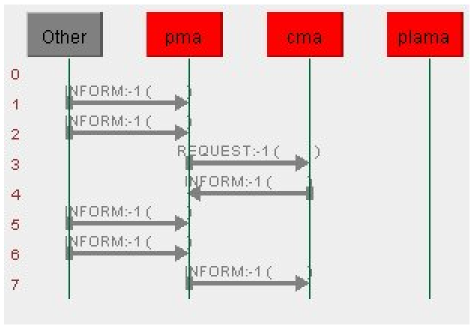

Figure 5. Execution run concerning the “answer_to_life” parameter.

The third example shows the behaviour of the system for the management of the “connection_to_server” parameter. The behaviour has been illustrated in Section III and is 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.

The first configuration, shown in Figure 6, 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, 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.

If we make the configuration even more complex (Figure 7), the behaviour of the MAS changes. We add another CMA named Cma2, controlling two PMAs (Pma3 and Pma4). The

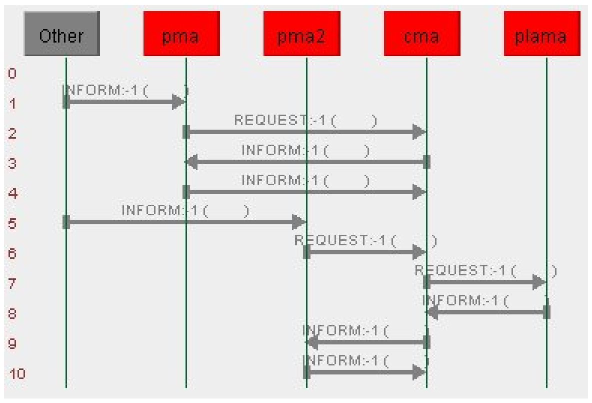

Figure 6. Execution run concerning the “connection_to_server” parameter.

agents shown in Figure 6 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.

V. Related Work and Conclusions

The exploitation of intelligent agents for monitoring and diagnosing distributed processes 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. 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

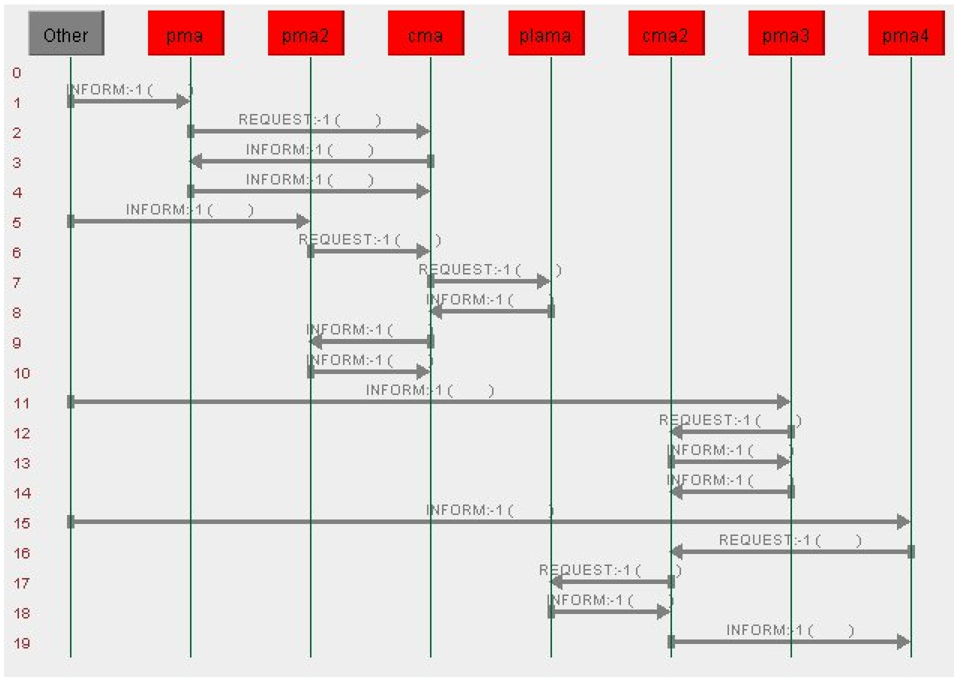

Figure 7. Execution run concerning the "connection_to_server" parameter, complex configuration.

agents in the monitoring and diagnosis domain, besides those already cited in Section I, 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 is used to realize a system that helps the monitoring of all the processes, instrumentation and data flows of the Kennedy Space Center'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 do not rely on a rule-based approach. For example, the paper [16] 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. 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.

Our project, although similar in its purposes to other applications developed in the past, demonstrates an increased industrial interest and trust in both agent-based and 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], [26]). The industrial strength system described in [19], despite not using rule-based technologies, shares with our project the choice of Jade as the agent middleware.

In 2004, the *Agent Technology Roadmap: Overview and Consultation Report* 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, agent tools, languages (rule-based ones in particular), 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 project discussed in this paper represents a success story in this direction.

Acknowledgments

The authors acknowledge the "Iniziativa Software" CINI-FINMECCANICA project that partially funded this work.

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, 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.o 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*, 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.