Using Archon to Develop Real-World DAI Applications, Part 1

Nick R. Jennings and E.H. Mamdani, Queen Mary and Westfield College
Jose Manuel Corera, Iberdrola
Inaki Laresgoiti, Labein
Fabien Perriollat, Paul Skarek, and Laszlo Zsolt Varga, European Laboratory for Particle Physics

ARCHON PROVIDES A SOFTWARE FRAMEWORK THAT ASSISTS INTERACTION BETWEEN THE SUBCOMPONENTS OF A DISTRIBUTED AI APPLICATION, AND A DESIGN METHODOLOGY THAT HELPS STRUCTURE THESE INTERACTIONS.

IN MANY INDUSTRIAL APPLICATIONS, substantial time, effort, and finances have been devoted to developing complex and sophisticated software systems. These systems are often viewed piecemeal—as isolated islands of automation—when, in reality, they are components of a much larger business function.[1] A holistic perspective can integrate the subsystems into a coherent and consistent supersystem in which they work together to better meet the entire application's needs. Because the subsystems are integrated, the finite budgets available for information technology development can be made to go further. All the problem solvers can share consistent and up-to-date versions of the data; basic functions need only be implemented in one place; problem solving can use timely information that might not otherwise be available; and so on.

Developing a well-structured distributed artificial intelligence (DAI) system requires a software framework that assists interaction between the subcomponents, and a design methodology that helps structure these interactions. Archon (*ar*chitecture for *c*ooperative *h*eterogeneous *on*-line systems)[2,3] addresses both of these facets. It provides a decentralized software platform that offers the necessary control and level of integration to help the subcomponents work together. It also provides a concomitant methodology that offers guidance on how to decompose the overall application and how to distribute the constituent tasks throughout the community to best use the capabilities of the Archon framework.

The Archon project has applied both of these facets to several real-world industrial applications. Two of these applications, electricity-transportation management and particle-accelerator control, have run on line in the organizations for which they were developed—Iberdrola, a Spanish electric utility, and the European Laboratory for Particle Physics (CERN). (Several other real-world applications have also used Archon software and methodology: electricity distribution and supply,[4] control of a cement kiln complex,[5] and control of a flexible robotic cell.[6] However, these are not yet operational in their host organizations.)

Designing a multiagent community

Archon's problem-solving entities are called *agents*; they can control their own problem solving and interact with other community members. The interactions typically involve agents cooperating and communicating with one another to enhance their individual problem solving and to better solve the overall application problem. Each agent consists of an *Archon layer* and an application program (known as an *intel-*

0885-9000/96/$4.00 © 1996 IEEE

ligent system). The Archon approach clearly distinguishes between an agent's social know-how (the AL) and its domain-level problem solving (the IS). Such an approach is flexible and open—imposing relatively few constraints on the application designer, yet providing many useful facilities. Custom-built ISs can use Archon to enhance their problem solving and to improve their robustness. However, preexisting ISs can also be incorporated into an Archon-based system, with a little adaptation, and can experience similar benefits. Incorporating preexisting ISs is important, because in many cases developing the entire application afresh would be too expensive or too large a departure from proven technology.[7]

To successfully incorporate both custom-built and preexisting systems, system designers must design the community from two different perspectives simultaneously. This involves a top-down approach to look at the application's overall needs and a bottom-up approach to look at the capabilities of the existing systems. Once designers have identified the gap between what is required and what is available, they can choose to provide the additional functions through new systems, through additions to the existing systems, or through the Archon software itself. This methodology, which is described more thoroughly by Laszlo Varga and his colleagues,[4,8] shapes the design process by providing guidelines for problem decomposition and distribution that reduce inefficiencies.

The design process has two main phases: problem analysis and actual design. The former should encompass both the preexisting systems and potentially necessary new systems. The preexisting systems and their users together already form a partially automated multiagent system. The designers then must extend and enhance this system, by discovering all additional cooperative action in the analysis phase, and by applying the tools provided by the Archon architecture, wherever they pertain, in the design phase. The analysis phase must identify and document the interactions between the existing systems, between the existing systems and their users, and between the different users, for later use in the design phase. The analysis phase might also conclude that some of the existing systems must be split up into several distinct agents—for example, to increase their maintainability or to increase the attainable parallelism.

The actual agent community design goes top-down from the community level to the individual tasks. On the community level, the designers must decide the granularity of the agents: whether it corresponds to the granularity of the preexisting systems (that is, whether each existing system should be encapsulated as an agent) or whether a finer decomposition is more appropriate. Then they must determine each agent's role in the community and the decomposition of that role into separate skills within the agent.

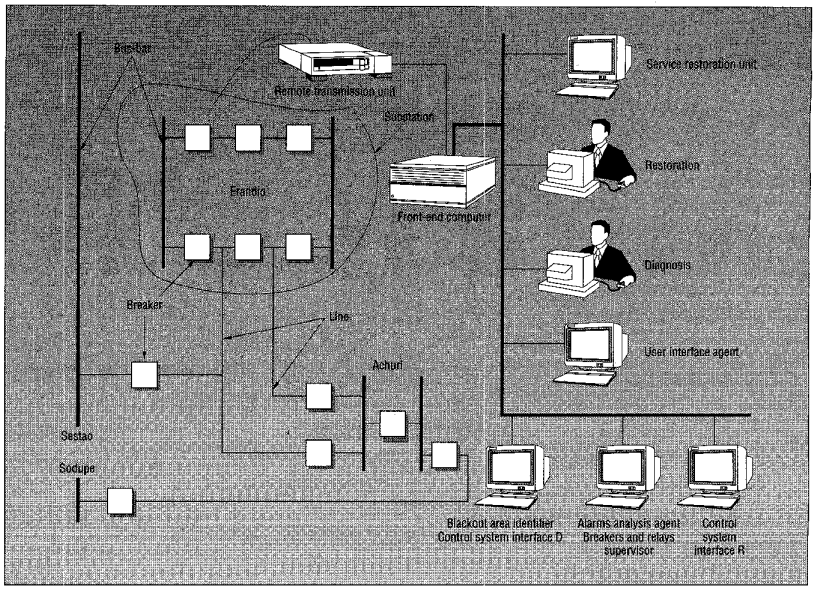

Figure 1. Archon agent architecture.

Another important methodological question relates to the user interface: does the designer hide all agents behind one interface, or should each agent have its own interface (and hence make all the interactions visible)? The former approach suits applications in which the users view the multiagent system as a single unit, while the latter approach is better for applications where agents mainly serve their own goals.

Knowing the skills in the community, designers can describe the types of messages that agents send each other, and can decide the mechanisms to disseminate information among the agents. After these steps and decisions, they can instantiate the AL directly.

The Archon architecture and software framework

The Archon software has integrated a wide variety of application program types under the general assumption that the ensuing agents will be loosely coupled and semiautonomous. The agents are loosely coupled because the number of interdependencies between their respective ISs are minimized; the agents are semiautonomous because their control regime is decentralized (each individual ultimately decides which tasks to execute in which order). The ISs themselves can be heterogeneous—in terms of their programming language, their algorithm, their problem-solving paradigm, and their hardware platform (Claudia Roda and her colleagues give a complete taxonomy and its effect on system design[9]). Their differences are masked by a standard AL-IS interface. An AL views its IS in a purely functional manner; it expects to invoke functions (tasks) that return results; and a fixed language[4] manages this interaction.

An Archon community has no centrally located global authority; each agent controls its own IS and mediates its own interactions with other agents (*acquaintances*). Each community member's local goals express the system's overall objectives. Because the agents' goals are often interrelated, an Archon-based system requires social interactions to meet global constraints and to provide the necessary services and information. The agent's AL controls such interactions; relevant examples include asking for information from acquaintances, requesting processing services from acquaintances, and spontaneously volunteering information that the agent believes is relevant to others.

In other words, an agent's AL needs to control tasks in its local IS, decide when to interact with other agents (for which it needs to model the capabilities of its own IS and the ISs of the other agents), and communicate with its acquaintances. Archon's modular and layered implementation architecture (see Figure 1) embodies these basic

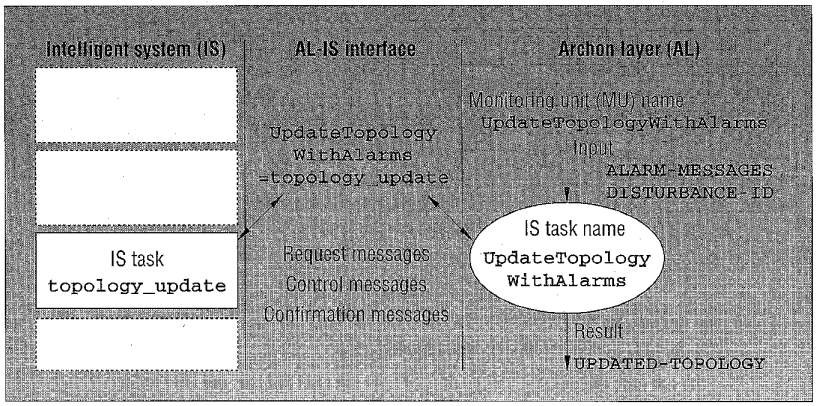

Figure 2. `UpdateTopologyWithAlarms` monitoring unit.

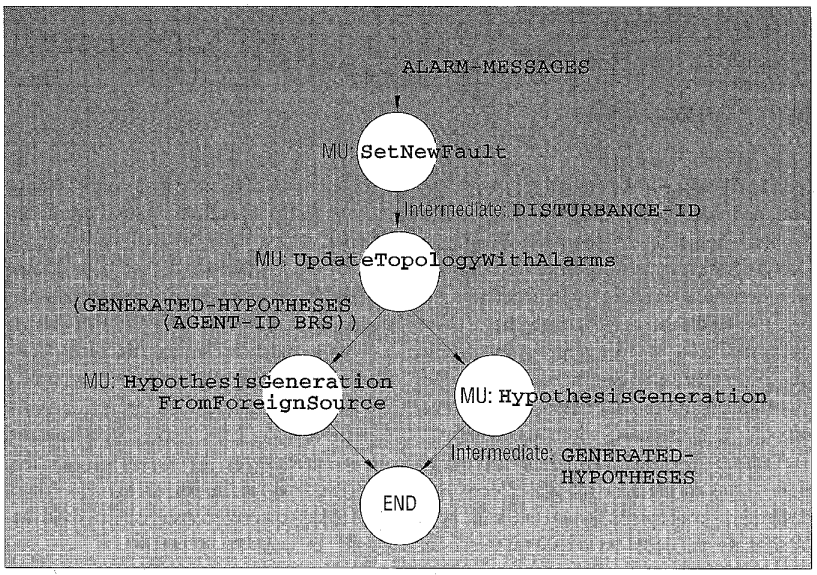

Figure 3. `StartNewDiagnosis` plan.

requirements in four modules: the *monitor*, *planning and coordination module*, *agent information management module*, and *high-level communication module*.

Monitor. The monitor controls the local IS, using three levels of representation: *monitoring units*, *plans*, and *behaviors*.

MUs. Monitoring units are the finest level of control in the AL; an MU represents each IS task. MUs present a standard interface to the monitor whatever the host programming language and hardware platform of the underlying IS. Figure 2 represents an MU from the Iberdrola application, called `UpdateTopologyWithAlarms`, which takes `ALARM-MESSAGES` and `DISTURBANCE-ID` as inputs, and outputs `UPDATED-TOPOLOGY`. The IS task associated with this MU is called `UpdateTopologyWithAlarms` in the AL and `topology_update` in the IS.

MUs can send and receive messages (directives, confirmations, and requests) to and from the IS. All messages have to pass through the AL-IS interface, which performs the translation and interpretation required for the IS to understand the AL directives and for the AL to understand the IS messages. For instance, in the above example, the interface invokes the IS function `topology_update`, passes the arguments in the form in which they are expected by the IS's host language, and returns the updated topology in the format expected by the AL. For the IS to be able to react to an AL directive, the interface must translate the command into the corresponding local control actions. However, this interpretation depends on the vagaries of the IS—for example, `ABANDON` in a C program might involve killing a process, whereas in a rule-based system it might mean clearing all the facts asserted by the said task and then stopping. Such fundamental differences in the way the same AL communications function in different ISs mean that the interface commands must be specialized for the IS's programming language and implementation paradigm. (We'll discuss later how we carried this out for the CERN application.) Other interface functions include specifying how many invocations of a particular task can run in parallel and how many can be queued should that limit be reached.

Plans. At the next level of granularity are plans. Plans are prespecified acyclic OR-graphs in which the nodes are MUs and the arcs are conditions. These conditions can depend on data already available from previously executed MUs in the plan or on data input to the plan when it started. They can use the locking mechanism for critical sections of the plan and can return intermediate results before a plan has completed.

Figure 3 shows a sample plan that starts the fault-diagnosis activity in the Iberdrola application. First, `ALARM-MESSAGES` are input to the `SetNewFault` MU, which notes a new fault in the network and generates a new identifier (`DISTURBANCE-ID`) for it. The MU returns this identifier as one of the plan's intermediate results. The alarm messages and the disturbance identifier then become inputs to the `UpdateTopologyWithAlarms` MU. When this MU is complete, the model of the network on which the diagnosis will be based is up to date, so identification of a list of potential faults can commence. This activity can proceed in two ways. The plan checks whether an agent called BRS has already provided a list of generated hypotheses (and stored it in the agent information management module's domain-data component). If so, these hypotheses should form the start point (the `HypothesisGenerationFromForeignSource` MU should execute). If no pertinent information is available, the plan should generate the list from scratch (the `HypothesisGeneration` MU should execute). In this case, the plan returns the list of generated hypotheses as an intermediate result so that they can be used elsewhere in the agent or can even be disseminated to relevant acquaintances.

The plan mechanism has a built-in backtracking facility that can express preferences and deal with complex alternatives. Consider

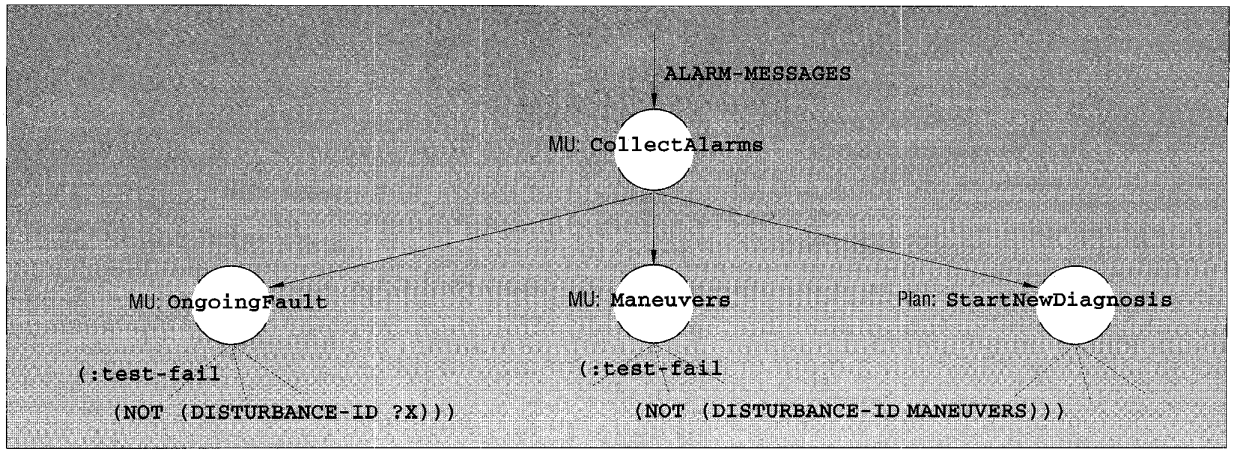

Figure 4. `ReceiveAlarms` plan (with backtracking).

the plan **ReceiveAlarms** (see Figure 4), which determines what course of action an agent should take when it receives alarm messages. This plan has three possible steps:

1) See whether the alarms correspond to a known ongoing fault.
2) See whether the messages have been generated by planned maintenance (maneuvers) on the network.
3) See whether the alarms correspond to a new fault.

The monitor first tries the leftmost branch and executes the **OngoingFault** MU. If this MU cannot give a disturbance identifier for the alarms, they cannot correspond to a known fault, so this branch fails. If, on the other hand, the MU finds an identifier, then the plan mechanism traverses the rest of the branch. In the case of failure, the mechanism backtracks to the last successful execution (the **CollectAlarms** MU) and tries the next branch (the **Maneuvers** MU)—this branch fails if the disturbance identifier associated with the alarms does not have the tag **MANEUVERS**. Finally, if the alarms are not generated by maneuvers, they correspond to a new fault, so monitor invokes the **StartNewDiagnosis** branch (see Figure 3). This branch never fails, so the **ReceiveAlarms** plan will successfully terminate when the plan completes.

Behaviors. The highest level of representation for IS activities is the *behavior level.* Behaviors contain a plan, a trigger condition for activating the behavior, descriptions of the inputs needed by the activity and the results that will be produced, and any children of the behavior. There are two types of behavior: those that are visible to the planning and coordination module (and the other AL components) and those that are purely internal to the monitor (for example, **RefineHypotheses** and **ValidateHypotheses** in Figure 5). The former type, called *skills* (for example, **DiagnoseFaults** in Figure 5), can be triggered by new data (those arriving from other agents or those that the agent itself has generated) or by direct requests from other agents.

DiagnoseFaults triggers when the **ReceiveAlarms** plan detects a new fault. It requires a block of alarm messages as an input and executes the **StartNewDiagnosis** plan (see Figure 3). When this plan is complete, the monitor considers the child behaviors.

The leftmost behavior (**RefineHypotheses**) processes first. It takes the list of generated hypotheses from the **StartNewDiagnosis** plan and tries to refine them (a process that involves removing impossible alternatives—see "A cooperative scenario" in Part 2 for more details). The behavior checks first whether the agent has the domain data **INITIAL-AREA-OUT-OF-SERVICE** in its agent information management module. If it does, the **REF1** (refinement 1) MU executes and the **RefineHypos** plan successfully terminates. If no initial area is out of service, the behavior examines the next branch of the plan. Again, the behavior evaluates the condition on its arc. If the agent has received a list of validated hypotheses from agent BRS, the **REF2** MU executes and the plan successfully completes. If this condition is untrue, the behavior tries the third arc—this tests whether the agent has received a list of generated hypotheses from agent BRS. If it has, the **REF3** MU executes and the plan successfully terminates. If not, then all of the alternatives have been exhausted—hence, the **RefineHypos** plan has failed and so, in turn, has the **RefineHypotheses** behavior.

If the **RefineHypotheses** behavior succeeds (that is, **REF1**, **REF2**, or **REF3** executes), its child, **ValidateHypotheses**, executes next. If it fails, the monitor invokes the backtracking mechanism, and the alternative child of **DiagnoseFaults** (which is also **ValidateHypotheses**) executes. In either case, **ValidateHypotheses** takes the generated and refined hypotheses (the latter only if they are available—**RefineHypotheses** might have failed) as inputs and executes the **Diagnose** plan, which produces **VALIDATED-HYPOTHESES**. Because this behavior has no children, it is deemed finished—this, in turn, means that the **DiagnoseFaults** skill has successfully completed. (We'll present the actual code for many of these monitor concepts in "Part 2: Electricity Transportation Management" and "Part 3: Particle Accelerator Control.")

Agent information management module. The AIM is a distributed-object-management system that was designed to provide information-management services to cooperating agents.[10] In Archon, it stores both the agent models and the domain-level data.

Agent models. Agent models come in two forms: *self models* and *acquaintance models.* The self model contains information about the local IS, and the acquaintance models contain information about the other agents in the system with which the modeling agent will interact. The type of information contained in both models is approximately the same, although it varies in the level of detail, and includes the agent's skills, interests, current status, workload, and so on.

To illustrate the agent models, consider an agent, again from the Iberdrola application, that can produce information about

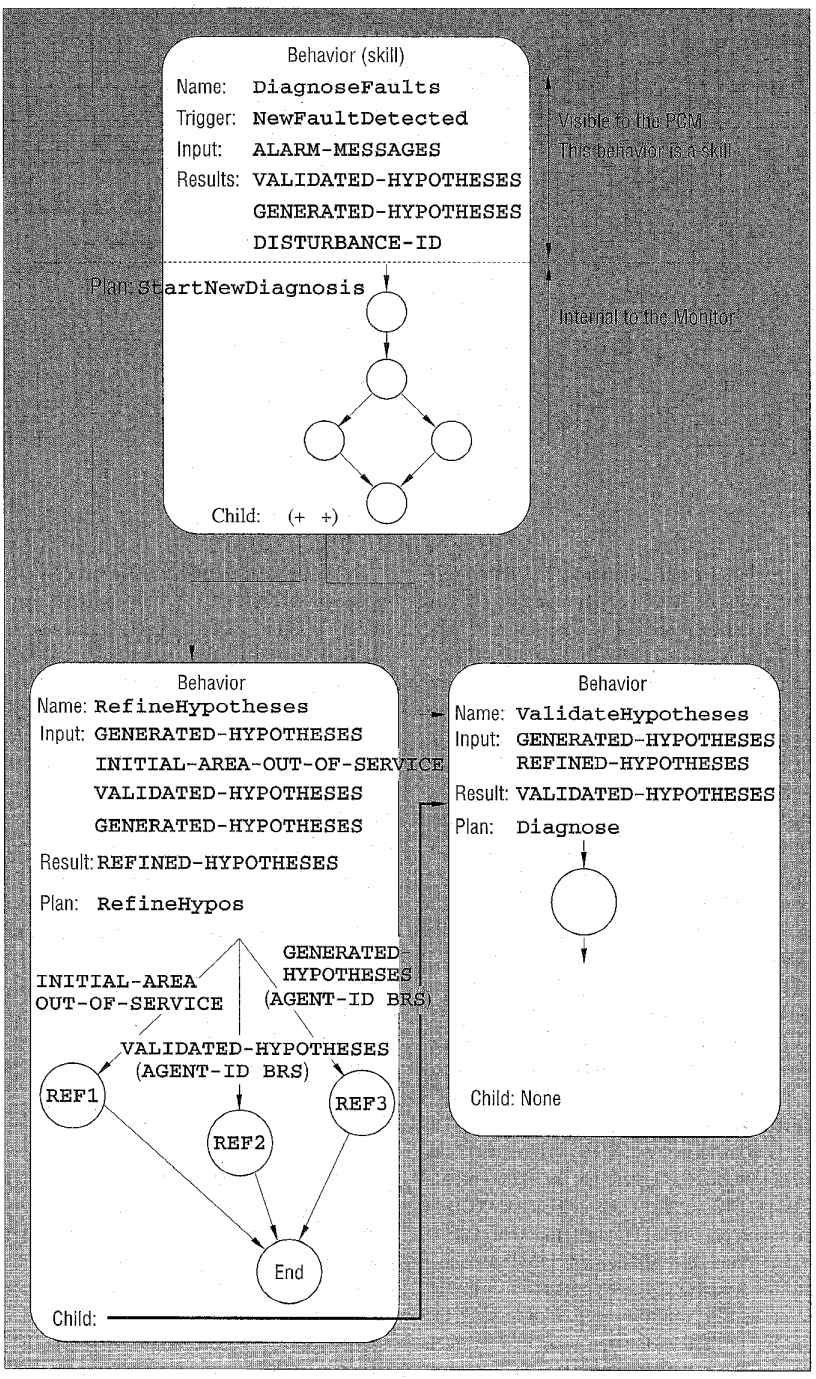

Figure 5. `DiagnoseFaults` skill with `RefineHypotheses` and `ValidateHypotheses` child behaviors.

`ALARM-MESSAGES`. The interest slots of its acquaintance models contain those agents who are interested in receiving this information and the conditions under which they are interested (a null condition signifies in all cases). The following portion of an acquaintance model specifies that agent BRS is interested in alarm messages that contain chronological information, that agent AAA is interested only in nonchronological alarm messages, and that agent BAI is only interested in nonchronological alarm messages that have the string `INT` in their `ALARMS` field (see Figure 6).

Domain-level data. In many industrial applications, the domain-level data that the agents must exchange has a complex internal structure. In Archon, the AIM specifies and maintains this structure. For example, Figure 7 defines the information type `ALARM-MESSAGES`, where each of the following types has the following set of permissible values: `AGENT` (`CSI` | `AAA`), `DISTURBANCE-SOURCE` (`yymmddhhmmss` | `MANEUVERS` | `UNKNOWN`), `Y-N-FLAG` (`YES` | `NO`), `BLOCK-TYPE` (`UNIQUE` | `UNKNOWN` | `MIXED`), `ID-TYPE` (`YYMMDDHHMMSS`), and `LIST-OF-ALARMS` (`ALARM1`, ..., `ALARMN`). Figure 8 shows a specific instance. (Based on the interest descriptor in Figure 6, this alarm message would be deemed of interest to agent AAA but not to agent BRS. The first two alarm fields (both starting with `INT`) would be of interest to agent BAI, but the third field (starting with `OSCILOS`) would not.)

Once domain data is stored in the AIM, the AL's reasoning and control mechanisms can retrieve it (for example, the retrieval of `GENERATED-HYPOTHESES` with agent identifier `BRS` in the `StartNewDiagnosis` plan). Because the data has a definite structure, it is possible to access the named subparts (for example, the AIM can check that `CHRONOLOGICAL` exists as a distinct attribute and that it has the `YES` value).

Planning and coordination module. The PCM is the reflective part of the AL, reasoning about the agent's role in terms of the wider cooperating community.[11] This module assesses the agent's current status and decides which actions to take to exploit interactions with others while ensuring that the agent contributes to the community's overall well-being. PCM functions include deciding which skills should execute locally and which skills to delegate to others, directing requests for cooperation to appropriate agents, determining how to respond to requests from other agents, and identifying when to disseminate timely information to acquaintances who would benefit from it.

The PCM consists of generic rules about cooperation and situation assessment that apply to all industrial applications—the AIM's self and acquaintance models store all the domain-specific information needed to define individual behavior. To perform its duties, the PCM refers to the self and

```
INTEREST-DESCRIPTOR
  INFORMATION-NAME: ALARM-MESSAGES
  INFORMATION-CONDITION:
    ["BRS",   (CONTAIN (ALARM-MESSAGES "CHRONOLOGICAL "YES"));
     ("AAA",  (CONTAIN (ALARM-MESSAGES "CHRONOLOGICAL "NO"));
     ("BAI",  (AND  (CONTAIN (ALARM-MESSAGES "CHRONOLOGICAL "NO""))
                    (CONTAIN (ALARM-MESSAGES.ALARMS "INT"))));]
```

Figure 6. An exemplar interest slot of an acquaintance model.

```
ALARM-MESSAGES
  AGENT-ID: AGENT                    DISTURBANCE-ID: DISTURBANCE-SOURCE
  CHRONOLOGICAL: Y-N-FLAG            BLOCK-TYPE: BLOCK-TYPE
  BLOCK-ID: ID-TYPE                  ALARMS: LIST-OF-ALARMS
```

Figure 7. Definition of `ALARM-MESSAGES` information type.

```
((ALARM-MESSAGES
 (AGENT-ID CSI) (DISTURBANCE-ID 921008121423) (CHRONOLOGICAL NO)
 (BLOCK-TYPE UNIQUE) (BLOCK-ID 921008121423)
 (ALARMS (  (INT 921008 121745 HER22EBAB CA HER LOCAL)
            (INT 921008 121745 HERLABA_HER22 CA HER LOCAL)
            (OSCILOS 0))
```

Figure 8. A specific instance of the `ALARM-MESSAGES` information type.

acquaintance models. For example, to determine how to obtain currently unavailable information needed to execute a behavior, the PCM refers to its agent's self model to see if the PCM can obtain the information locally by executing an appropriate skill. If it cannot, it checks the acquaintance models to see if another community member can provide the information.

Interplay between the PCM and the agent models also occurs when the monitor provides some results from a behavior (for example, the intermediate result `GENERATED-HYPOTHESES` from `StartNewDiagnosis`). First, the PCM checks the self model to see if the data can be used locally, and then it examines the acquaintance models to see if any other agents are believed to be interested in receiving the data.

The PCM also refers to the self model when deciding whether to honor a request from another agent. If it decides to honor the request, it activates the necessary skill to provide the requested data; when the information is available, it ensures that the reply goes to the request's source.

High-level communication module. The HLCM lets agents communicate with one another using services based on TCP/IP. It incorporates the functionality of the International Organization for Standardization/Open Systems Interconnection (ISO/OSI) Session Layer, which continuously checks communication links and provides automatic recovery of connection breaks when possible. The HLCM can send information to named agents or to relevant agents (it decides whether an agent is relevant by referring to the interests registered in the acquaintance models).

IN THE FOLLOWING TWO ARTICLES, we'll describe in detail how we used Archon to develop the CERN and Iberdrola DAI applications. These two descriptions complement each other: the CERN application involves two relatively homogeneous agents, both of which were originally conceived as stand-alone preexisting systems, and concentrates on the problems of integrating them into an Archon community and getting them to interact in a functionally accurate, cooperative manner.[12] The Iberdrola application, on the other hand, involves seven very heterogeneous agents, five of which were custom built, which perform three main types of activity (data acquisition, fault diagnosis, and service restoration) and cooperate in different styles.

Acknowledgments

This work was carried out in the Esprit II project Archon (P-2256), whose partners were Atlas Elektronik, Framentec-Cognitech, Labein, Queen Mary and Westfield College, Iberdrola, EA Technology, Iridia, Amber, the Technical University of Athens, FWI University of Amsterdam, CAP Volmac, CERN, and the University of Porto. We wrote this article on behalf of the whole consortium; we do not claim to have conceived or implemented all of the concepts that we've described. These concepts originated from interactions between all of the consortium's members. However, these individuals contributed significantly to certain aspects of the project: Thies Wittig, Abe Mamdani, Erick Gaussens (architecture design); Erick Gaussens, Daniel Gureghian, Jean-Marc Loingtier, and Bernard Burg (the monitor); Nick Jennings, Jeff Pople, Jochen Ehlers, and Eugenio Oliveira (the PCM); Frank Tuijnman, Hamideh Afsarmanesh,

and Giel Wiedijk (the AIM); Claudia Roda and Jutta Mueller (the HLCM); Nick Jennings (the agent models); and Rob Aarnts (the C++ implementation). Juan Perez, Inaki Laresgoiti, Esther Abel, Jon Barandiaran, Luis Fernando Penafiel, Vicente Ferrer, Jose Corera, and Javier Echavarri helped develop the Iberdrola system; and Laszlo Varga, Paul Skarek, Fabien Perriollat, Joachim Fuchs, Sergio Pasinelli, and Elena Wildner helped develop the CERN system.

Laszlo Varga performed this research while on leave from KFKI-MSZKI (Central Research Inst. for Physics/Research Inst. for Measurement and Computing Techniques), in Budapest.

References

1. N.R. Jennings, *Cooperation in Industrial Multiagent Systems*, World Scientific Press, London, 1994.

2. T. Wittig, ed., "ARCHON: An Architecture for Multiagent Systems," Ellis Horwood, Chichester, United Kingdom, 1992.

3. T. Wittig, N.R. Jennings, and E.H. Mamdani, "ARCHON —A Framework for Intelligent Cooperation," *IEE-BCS J. Intelligent Systems Engineering*, Vol. 3, No. 3, Autumn 1994, pp. 168–179.

4. D. Cockburn and N.R. Jennings, "ARCHON: A Distributed Artificial Intelligence System for Industrial Applications," in *Foundations of Distributed Artificial Intelligence*, G.M.P. O'Hare & N.R. Jennings, eds., John Wiley & Sons, New York, 1996, pp. 319–344.

5. G. Stassinopoulos and E. Lembesis, "Application of a Multiagent Cooperative Architecture to Process Control in the Cement Factory," ARCHON Tech. Report 43, Atlas Elektronik, Bremen, Germany, 1993.

6. E. Oliveira and C. Ramos, "Cooperation in the University of Porto Robotic Testbed," tech. report, ARCHON Public Deliverable 1050, Atlas Elektronik, Bremen, Germany, 1993.

7. N.R. Jennings and T. Wittig, "ARCHON: Theory and Practice," in *Distributed Artificial Intelligence: Theory and Praxis*, N.M. Avouris and L. Gasser, eds., Kluwer Academic Press, Dordrecht, The Netherlands, 1992, pp. 179–195.

8. L.Z. Varga, N.R. Jennings, and D. Cockburn, "Integrating Intelligent Systems into a Cooperating Community for Electricity Distribution Management," *Expert Systems with Applications*, Vol. 7, No. 4, Oct.–Dec. 1994, pp. 563–579.

9. R. Roda, N.R. Jennings, and E.H. Mamdani, "The Impact of Heterogeneity on Cooperating Agents," *Proc. AAAI Workshop on Cooperation Among Heterogeneous Intelligent Systems*, AAAI Press, Menlo Park, Calif., 1991.

10. F. Tuijnman and H. Afsarmanesh, "Distributed Objects in a Federation of Autonomous Cooperating Agents," *Proc. Int'l Conf. Intelligent and Cooperative Information Systems*, IEEE Computer Society Press, Los Alamitos, Calif., 1993, pp. 256–265.

11. N.R. Jennings and J.A. Pople, "Design and Implementation of ARCHON's Coordination Module," *Proc. Workshop on Cooperating Knowledge Based Systems*, Dake Centre, Univ. of Keele, Keele, UK, 1993, pp. 61–82.

12. V.R. Lesser, "A Retrospective View of FA/C Distributed Problem Solving," *IEEE Trans. Systems, Man and Cybernetics*, Vol. 21, No. 6, Nov.–Dec. 1991, pp. 1347–1362.

Nick R. Jennings is a reader in the Department of Electronic Engineering at Queen Mary & Westfield College (University of London). He heads the Distributed AI Unit, which conducts basic and applied research into the development of multiagent systems. His research interests include multiagent systems, agent-based computing, and the information superhighway. Contact him at the Dept. of Electronic Engineering, Queen Mary & Westfield College, Mile End Rd., London E1 4NS, United Kingdom; n.r.jennings@qmw.ac.uk.

Jose Manuel Corera is responsible for advanced applications in Iberdrola's Control Systems Department. His research interest is software applications for network management, including areas such as AI, databases, and human-computer interaction. He has a bachelor's degree in electrical engineering from the Bilbao Engineering School and an MSc in advanced manufacturing from the Cranfield Technology Institute. His address is Iberdrola, Gardoqui 8, Bilbao, 48008 Spain; jose.corera@iberdrola.es.

Inaki Laresgoiti is a member of the Information Technology Department of Labein (a research institute in the Basque country). He is responsible for R&D in distributed systems and distributed artificial intelligence and their applicability to the supervision of complex industrial processes. He has been responsible for the implementation of several knowledge-based systems for the electrical utilities in the surrounding area. He is a member of the Official College of Industrial Engineers of Bizkaia. Contact him at Labein, Information Technology Dept., Parque Tecnologico. Edificio 101, 48170 Zamudio, Bizkaia, Spain; lares@labein.es.

E.H. Mamdani holds the Nortel/RAE Chair of Telecommunications Strategy and Services in the Department of Electrical and Electronic Engineering, Imperial College, London. He is well known for his research in fuzzy logic during the seventies and eighties. His recent research has focused on uncertainty in artificial intelligence and on intelligent agents for telecommunications. He is a Fellow of the Royal Academy of Engineering and the IEEE. Contact him at the Dept. of EE, Imperial College, Exhibition Rd., London SW7, United Kingdom; e.mamdani@ic.ac.uk.

Fabien Perriollat, a civil engineer in CERN's Nuclear Physics Experiment, is the project leader of the controls for the Compact Muon Solenoid, one of the two large experiments of the future Large Hadron Collider. He has 25 years' experience in control systems for large experimental facilities at CERN. He is a member of the New York Academy of Sciences. He can be contacted at the PS Division, CERN, CH-1211 Geneve 23, Switzerland; fabien.perriollat@cern.ch.

Paul Skarek works at CERN, where his research focuses on introducing and applying AI in the accelerator domain. His special interest is in expert systems and cooperating multiagent systems. He got his PhD in physics and in mathematics in 1960 from the University of Vienna. He is a member of the Swiss Informaticians Society and the Austrian Society for Artificial Intelligence. Contact him at the PS Division, CERN, CH-1211 Geneve 23, Switzerland; paul.skarek@cern.ch.

Laszlo Zsolt Varga is a senior scientific associate at the KFKI-MSZKI research institute. His research interests include distributed artificial intelligence and computer communication. He graduated with distinction in 1984 and received his PhD in 1988 from the Technical University of Budapest. His address is KFKI-MSZKI, H-1525 Budapest POB 49, Hungary; varga@sun60.mszki.kfki.hu.