ADAPTATION IN A MULTI-AGENT SYSTEM THROUGH SEMANTIC ADDRESSING

Staffan Hägg
Department of Computer Science
University of Karlskrona/Ronneby, Sweden
Staffan.Hagg@ide.hk-r.se
http://www.pt.hk-r.se/~staffanh

Keywords: Distributed Computing, Multi-Agent Systems, Semantic Addressing, Adaptation.

ABSTRACT

In many application domains the design of complex systems must cope with rapid and unforeseeable change, requiring an elaborate integration of execution and communication methods. This is especially the case when real-time constraints are present.

In this paper a Multi-Agent Systems approach to distributed computing is presented. It has a programmable model of agent interaction, and the interaction is tightly integrated with the mechanisms that control the individual agent behaviour. We introduce the concepts of tailored interaction and semantic addressing which support abstraction, flexibility and robustness in open and emergent distributed systems. We then show that it enables the system to adapt to a number of frequent types of change in the environment.

The automation of power distribution is a domain where new services are introduced and new infrastructures are exploited. DA-SoC is an agent architecture developed for this domain, and it exemplifies the need for adaptability and the use of semantic addressing to accomplish this.

1. INTRODUCTION

The background to this paper is a project, the Societies of Computation (SoC), at the University of Karlskrona/Ronneby in Sweden [2]. In a joint effort with Sydkraft, a major Swedish power distribution company, technologies for automating the power distribution process are studied. The goal is to make the distribution process flexible and robust to load changes, reconfigurations, faults, etc. (Distribution Automation - DA), and to offer new services to customers, such as the choice between different levels of service and even between different suppliers (Demand Side Management - DSM). Furthermore, and perhaps the most challenging, the goal is to exploit a new infrastructure for Home Automation (HA), integrated with the DA/DSM system. The physical electric grid is used for communication, and small and simple pieces of hardware are plugged into wall-sockets and lamp-sockets, operating as interfaces to appliances, or are integrated within them. Basic techniques for this is in the process of being standardized. Our challenge is therefore to develop methods for using this infrastructure in terms of computation and communication, thus creating truly plug-in software. Computation in this environment is thus distributed *en masse*, and communication is highly unreliable. Moreover, changes are unpredictable; there exists no generic way of describing the total system at any one point in time.

In section 2 of this paper, the adaptation problem is outlined from the application point of view. Section 3 gives a brief overview of the DA-SoC agent architecture, and in section 4 the concept of semantic addressing is introduced. In section 5 we show through some sample scenarios how semantic addressing can be used to give adaptability to the system, and then we end with conclusions and an outline of on-going and future activities.

2. THE ADAPTATION PROBLEM

Given the problem of the application, we can easily conclude that a distributed computing system with our goals must be adaptive. We do not try to give a proper definition of adaptation in this context[1]. Rather, we identify a number of events that the system must cope with and adapt to:

(1) Communication is unreliable.
(2) Modules may be entered to and retracted from the system.
(3) Parts of the system (a module or a group of modules) may be modified.
(4) Functionality may be entered, removed, or redefined (as a result of items (2) and (3) above).
(5) The “outside” world may change.

When looking at traditional approaches to distributed computing, we find that there are

1. In fact, there is not, to our knowledge, any real effort to define adaptation for this kind of real-time systems. The concept is commonly used with an intuitive understanding of it.

well-established protocols for handling unreliable communication media. The second and third items are sometimes provided for (using a logical addressing scheme rather than a physical one, and using version handling mechanisms, respectively). The last two items concern issues at a higher abstraction level and is more or less overlooked in traditional distributed systems.

When turning to Distributed AI, we find that basic communication issues are seldom addressed. On the other hand, item (5) is studied quite a lot, e.g. leading to adaptation based on cooperative learning [5], or specific negotiation or bidding schemes [4]. Items (2) - (4) above, i.e. agent configuration issues, are generally thought of as addressing problems and left to the underlying communication system.

With this background, we have drawn the following conclusions: First, a DAI approach allows us to define distributed entities as intelligent modules with decentralized control, preserving essential functionality in case of isolation. Second, as we emphasize computational aspects, the major part of events that require adaptation must be handled at a quite low level. And third, we find that current research often takes a layered approach, letting issues like negotiation and adaptation form a layer on top of layers that take care of addressing, coordination, and representation issues. We know from experience of other kinds of systems that good performance generally requires low-level support for high-level functions and, thus, an integration of functions across levels is asked for.

3. DA-SoC: AN AGENT ARCHITECTURE FOR DISTRIBUTED AND OPEN COMPUTING

A DA-SoC agent is procedural and goal-driven, and it holds a world model in declarative form. Its most elaborate feature is a programmable model of interaction, that lets the user tailor its interactive behaviour. A simplified picture of the agent head is shown in Fig. 1. The world model consists of a set of beliefs. The reasoner holds agent plans and a goal queue, letting the user program the agent in an event driven fashion. The programmable Model of Interaction (MoI) is realized through a set of interaction plans that are matched and, eventually, executed when a message comes in. By programming the MoI, the user can give agents specific roles in the society.

The DA-SoC architecture, the Distribution Automation Agent Language (DAAL), and the applicability of DA-SoC for our application domain are outlined in [3]. Here we concentrate on the agent interaction.

The basic communication mechanism in DA-SoC is a broadcast message passing scheme[1]. When a message is sent to an agent, all agents receive it, and every agent tries

1. In [1] Fisher discusses the impact of broadcast communication in MAS.

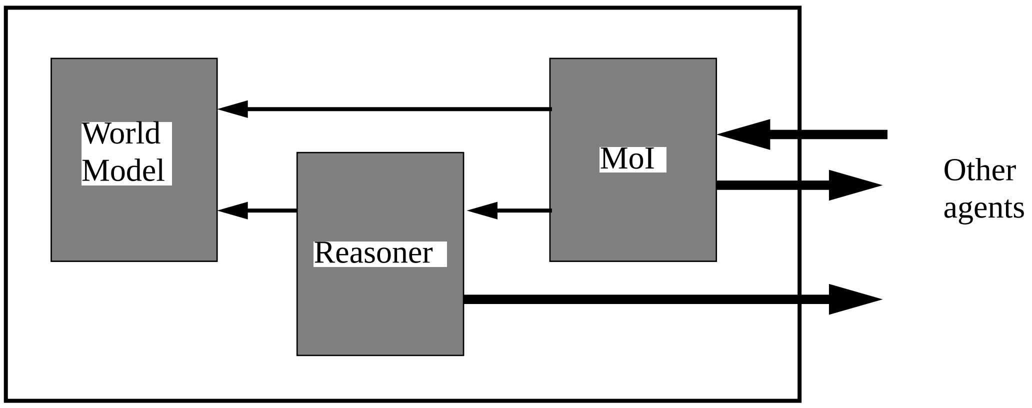


Fig. 1: Structure of the DA-SoC agent head

to match it with an interaction plan. The declaration part of an interaction plan has an agent name as the receiver's address and a pattern for what message contents it will accept. Both the agent name (the address) in the message and the message contents must therefore match the declaration part of an interaction plan. Upon a successful match, the plan is executed. During execution, the agent's world model may be altered. There can also be set an internal goal. In this case, the goal is matched with agent plans and, eventually, an agent plan is executed, during which the world model can be altered, goals can be set, and messages can be sent. A special feature is that one agent can hold an interaction plan with the name of another agent as the receiver's address. This enables for letting one agent play the role of another agent within the society.

4. SEMANTIC ADDRESSING AS A MEANS FOR ADAPTATION

Let us define our plans to be methods, accessible by other agents. Then the broadcast mechanism, the MoI, and agent plans form consecutive steps in an access model which we call *semantic addressing* (Fig. 2). (1) shows an agent executing an agent plan. During execution, a message "request ..." is broadcast (2). The receiving agents try to match the incoming message with interaction plan declarations (3), and in (4) we see that agents 2 and 4 execute successfully matched interaction plans, while agents 1 and 3 failed in matching the message. Finally, (5) shows that agent 2 set a goal during execution of the interaction plan, and the matched agent plan is now executed. Here, agent 4 either did not set a goal or failed in matching the goal with an agent plan.

Semantic addressing allows agents to access methods within other agents in the society without knowing where they should be found. But moreover, a method is accessed wherever it is semantically meaningful according to the application definitions, and where it is consistent with the agent's world model (which can change). The world model may be altered due to changes in the surrounding world or in agents' capabili-

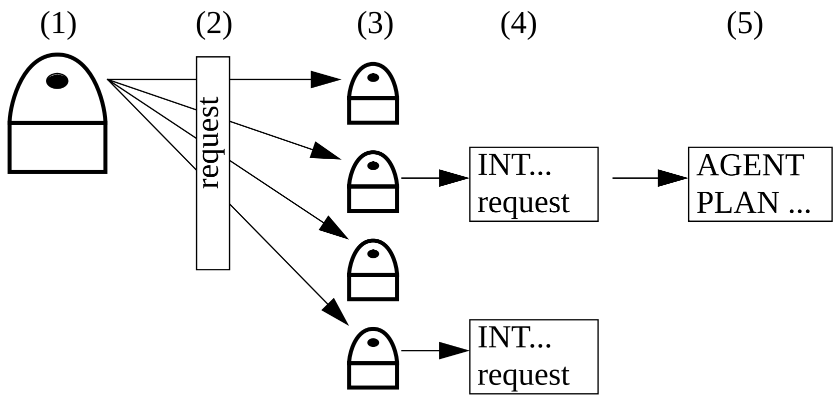


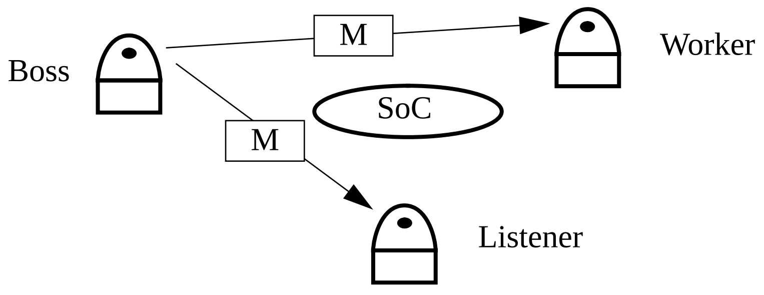
Fig. 2: Semantic addressing in DA-SoC

ties. Thus, the next time the same message is sent, there may be a goal set and a plan executed within another agent than the first time.

5. SAMPLE SCENARIOS

5.1. The Worker

The worker is represented by the agent Worker in Fig. 3, where the agent Boss sends a message M to Worker. When M comes in, Worker’s interaction plan[1]



Message M:							
	Worker	Boss	request	t_0	level	100	3

Fig. 3: Boss sends (broadcasts) a request message to Worker

```
INTERACTION PLAN request(Worker $time level($max $var))
   GOAL(SENDER $time level($max $var))
```

1. The full description of syntax and semantics for the language is given in [3], but for clarity, a string starting with $ is a variable that matches any string or numeric value. A string starting with $$ is a fact variable that matches any sequence of strings/numeric values, starting with a string.

successfully matches M, and the plan body is executed resulting in setting an individual goal for Worker.

The scenario describes a request for service. It includes no confirmation and shows no reply, but either or both can easily be realized in the plans.

5.2. The Listener

We introduce another agent to the scene, agent Listener in Fig. 3. Agent Boss sends the same message M to Worker. As M is broadcast, it also reaches Listener who has the interaction plan

```
INTERACTION PLAN request($agent $time $$fact)
BEGIN
   ACT(log_request(SENDER $agent $time))
   ACT($$fact)
END
```

and the plan declaration successfully matches M. When the plan is executed, two messages are sent to the agent body (the two ACT statements), containing all the information in the message. Observe that no agent plan is needed.

Listener effectively monitors all “request” messages in the society, and the body program could be e.g. a database for logging or a communication supervising program.

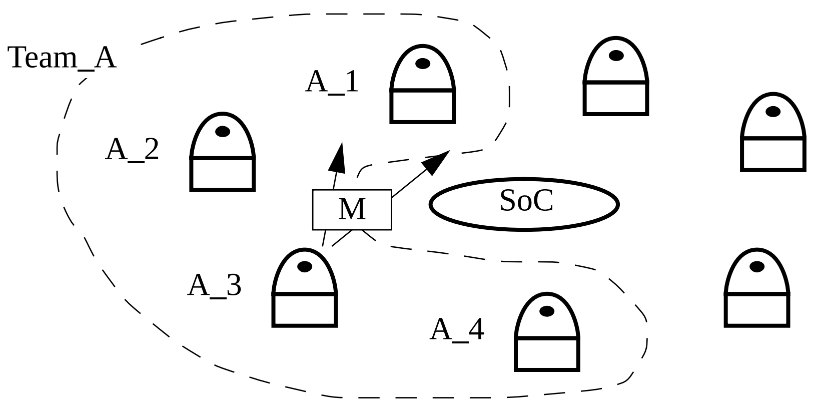
5.3. The Mirror

Let us change the presumptions in the last scenario and name the third agent Mirror instead of Listener. As before, Boss sends message M to Worker, and now Mirror has the same interaction plan as Worker, playing the role of Worker. The plan is successfully matched and executed, and if Mirror also has an appropriate agent plan that matches the goal that is set, it mirrors Worker’s behaviour with regard to the “level” request.

5.4. The Joint Venture

In this scenario agents A_1, A_2, A_3, and A_4 form a cluster within the society (Fig. 4). Agent A_3 now sends message M. Each of the agents in the cluster has the interaction plan[1]

1. The cluster is actually *defined* by the agents A_1 .. A_4 having interaction plans that answer to the same message, using the name Team_A as the receiver’s address.



Message M:	Team_A	A_3	joint_goal	t_0	obj_5	start	100	3

Fig. 4: The cluster Team_A, and a message M from agent A_3

```
INTERACTION PLAN joint_goal(Team_A $time $$fact)
BEGIN
   GOAL(SENDER $time $$fact)
   IF SUCCESS THEN
      SACT(agree(SENDER NOW $$fact))
END
```

which successfully matches M and is therefore executed in A_1, A_2, and A_4. Furthermore, suppose agents A_1 and A_2 (but not A_4) have an agent plan matching the goal set during execution of the interaction plan. A_1 and A_2 will thereby succeed in setting up their individual goals and report this to A_3 (the SACT statement). Agent A_4 fails in setting up an individual goal and therefore does not respond[1]. The individual agent plan bodies may be different for A_1 and A_2; these two agents can play different roles in the joint venture.

The decision is now A_3's on how to proceed. The scenario can be the first step in a *two phase commit* scheme, where resources are first reserved and then used. It can also be used for *planning* within the cluster; A_3 can decide on the behaviour of the other agents in the cluster, depending on the response.

6. CONCLUSIONS

If we return to section 2 and the events that should be handled, we first find that the problem of unreliable communication is addressed through giving agents true local

1. This can of course be replaced by a negative response.

control, not (in the general case) depending on other agents for basic functionality. For items (2) to (4) we see from the scenarios that semantic addressing lets the user modify the society of agents on the fly in a variety of ways, thereby adding, removing, and redefining functionality. One agent can effectively mirror the behaviour of another agent, thereby giving redundancy to the system. Alternatively, the mirroring agent can be given a different behaviour in response to the same messages. Both Listener and Mirror exemplify how *flexibility* and *robustness* can be achieved in the system; new functionality and redundant resources can be entered on the fly. In the last scenario, if the "outside" world changes (and the agents' models of it), other agents in the cluster may answer to a certain message or act differently in response to it, thereby *adapting the behaviour of the society of agents* rather than that of individual agents.

Semantic addressing is based on broadcast communication and a specific addressing scheme, but it is not thereby merely a low level mechanism. It integrates the lower level message matching with internal agent reasoning in terms of goals, plans and a world model.

7. ON-GOING AND FUTURE ACTIVITIES

The project continues the work with the presented and similar architectures, and there are efforts to make commercial products rather soon. Research is directed towards methods for getting fault tolerant systems, as well as studying the real-time properties of alternative solutions.

REFERENCES

[1] Fisher M., Representing and Executing Agent-Based Systems, *Proceedings of the ECAI'94 Workshop on Agent Theories, Architectures, and Languages,* August 8-12, 1994, Amsterdam, The Netherlands.

[2] Gustavsson, R. et al., *Societies of Computation (SoC) - A Framework for Open Distributed Systems, Phase II: 1995-98,* University of Karlskrona/Ronneby, Research Report 8/95, ISSN 1103-1581.

[3] Hägg S., and Ygge F., *Agent-Oriented Programming in Power Distribution Automation - An Architecture, a Language, and their Applicability,* Ph.L. Thesis, LUNFD6/(NFCS-3094)/1-180/(1995), Lund University, Sweden, May 1995.

[4] Rosenschein J. S., and Zlotkin G., *Rules of Encounter*, MIT Press, 1994, ISBN 0-262-18159-2.

[5] Sen S., and Sekaran M., Multiagent coordination with learning classifier systems, in *Proceedings of the IJCAI'95 Workshop on Adaptation and Learning in Multiagent Systems,* Montréal, Canada, August 21, 1995.