DA-SoC: A Testbed for Modelling Distribution Automation Applications Using Agent-Oriented Programming

Staffan Hägg
Department of Computer Science
and Business Administration
University of Karlskrona/Ronneby
Ronneby, Sweden
Staffan.Hagg@ide.hk-r.se

Fredrik Ygge
Department for Research
and Development
ElektroSandberg/Sydkraft
Malmö, Sweden
Fredrik.Ygge@ide.hk-r.se

Rune Gustavsson
PhD, Professor
Swedish Institute of Computer Science
Kista, Sweden
Department of Computer Science
and Business Administration
University of Karlskrona/Ronneby
Ronneby, Sweden
rune@sics.se, rune@ide.hk-r.se

Hans Ottosson
Director
Distribution Systems
& Market Applications
Corporate Research & Development
Sydkraft
Malmö, Sweden

Abstract

As computing systems are being applied to ever more demanding and complex domains, the infeasibility of constructing a single, monolithic problem solver becomes more apparent. Furthermore, important applications such as different types of management control systems for power generation and distribution are inherently distributed. A promising novel research direction to overcome complexity barriers in the design and maintenance of complex distributed applications is based on the view of such systems as societies of cooperating agents.

In this paper we describe an agent oriented design for load management in an automated power distribution system and a testbed for implementations of such systems. We also present an agent with some novel and important features. The research reported here is a cooperation between the authors' affiliations mentioned above, and it is a part of the larger projects, Intelligent Distribution Automation (IDA) at Sydkraft, Sweden, and Societies of Computation (SoC) at the University of Karlskrona/Ronneby, Sweden.

1 Introduction

As computing systems are being applied to ever more demanding and complex domains, the infeasibility of constructing a single monolithic problem solver becomes more apparent. To combat this complexity barrier, system engineers are starting to investigate the possibility of using multiple, cooperating problem solvers in which both control and data are distributed. Each *agent* has its own problem solving competence; however, it needs to interact with others in order to solve problems which lie outside its domain of expertise, to avoid conflicts and to enhance its problem solving.

Important application domains, such as different types of management control systems for telecommunication and power distribution, are inherently distributed. As those systems become more and more integrated and provide new types of services, novel agent oriented architectures have become natural extensions of more traditional architectures for distributed systems.

In this paper we outline an agent oriented design for load management in an automated power distribution system. The testbed explores new technologies to meet future network operations and management requirements of Distribution Automation (DA).

1.1 The SoC Project

Societies of Computation is a research project at the University of Karlskrona/Ronneby in Sweden. The SoC provides a framework for assessments of agent-oriented architectures of tomorrow's distributed systems and the use of those systems in industrial applications. For more detail, see [Gustavsson94].

Research within the SoC framework is conducted in cooperation with the Swedish Institute of Computer Science (SICS), the Technical University of Lund, The University of Kalmar, and with the industrial partner Sydkraft. The Sydkraft Group consists of some 50 wholly or partly owned companies, most of which are active in the electricity sector. The research reported in this paper is also supported by Blekinge Research Foundation, Ronneby SoftCenter, and Ronneby local government.

1.2 Outline of the Report

Chapter 2 of the report gives a background to the work at hand, and chapter 3 shows the principal architecture of the society of agents. In chapter 4 the testbed is described, and chapter 5 gives an example of how a DA application can be implemented on the testbed. The report ends with related work, conclusions, and an outline of future activities. The agent language is given in the appendix.

2 Background

2.1 Distribution Automation

Power distribution grids are evolving very rapidly into multi service integrated networks. As such they are converging with public and private tele/data communication networks, both in transport mechanisms and in higher level inter-operability between networks. This will be necessary in the near future.

Power distribution is the distribution of electricity from the 50kV level to the end user (e.g. a household at 220V). DA is a very important issue for energy production and distribution companies of today. DA is commonly defined as follows: *Automation of all processes (both technical and administrative) at energy (gas, electricity, district heating) distribution companies' sites.*

There are at present a number of DA related projects world-wide. The Intelligent Distribution Automation (IDA) project at Sydkraft is one of them. IDA's main aim is to develop DA services, using distributed computers and data communication. Among the most interesting services in (future) power distribution automation, we identify:

- Remote meter reading
- Fault detection and location
- Grid restoration
- Minimizing of losses
- Load management
- Remote or automatic control of transformers, capacitor cells, etc. in order to achieve higher quality electricity

Besides these services supporting the distribution of electricity, there are similar services for gas, district heating, water, etc. The challenge is to design and maintain networks that can deliver sustained high quality services to the customer. In a future market this could be a matter of survival.

Services in DA are often related to large and geographically dispersed distribution grids. The efficient management of such grids requires well structured computer networks and a lot of automated, distributed control. As pointed out above, the main purpose of the DA-SoC is to achieve those goals by utilizing structured societies of agents.

In this report we will concentrate on one DA service: load management in static grids. This service has a suitable level of complexity for a first generation of the DA-SoC testbed.

The main purpose for the load management service is to reduce the amplitude of power peaks, in case of a production shortage, limitations due to bottlenecks in the grid, or when the customer wants to reduce his power peaks.

2.2 Why Multi-Agent Systems?

Efficient and flexible management of power distribution highlights several advantages of an agent-oriented system approach, compared to more standard approaches for design of open distributed systems. The following list of features characterizes important aspects of a DA system.

- Complexity: The environment of hardware, controlling software, and functionality can be very complex, which requires powerful structuring tools.
- Heterogeneity: The underlying hardware and software components as well as the basic communication methods can be very heterogeneous, which requires powerful abstraction mechanisms.
- Dynamics: The DA system must be able to handle frequent changes in the environment, such as reconfigurations, load changes and faults, while maintaining system goals.
- Levels of distribution and abstraction: Power distribution is, physically, hierarchically organized and heavily distributed. The design and functionality of the DA system must be able to reflect this organization.
- Real-time properties: The DA system is a real-time system with different real-time demands for different operations and at different levels.

The SoC Multi-Agent System (MAS) approach of realizing a DA system offers a number of interesting properties addressing the issues above:

- It is inherently modular with distributed control.
- It supports evolutionary design.
- It has reasoning and abstraction mechanisms.
- It has a model of interaction, including a high level communication protocol, handling cooperation between heterogeneous agents.

The purpose of this report is to show how these properties of a MAS can be useful for the design of a DA system, and to describe how these ideas are realized in the testbed for modelling such systems.

2.3 Threads of Theoretical Background

The DA-SoC testbed is a project in the SoC framework. In SoC we pursue R&D in three parallel strands: theoretical foundations, generic technologies, and applications.

Theoretical foundations includes, as background, work done elsewhere. There is work on belief, desire, and intention from a philosophical perspective [Bratman90]. A formalization of an agent's rational behaviour in modal logic is given by [Cohen90]. An extension of those ideas to include the interaction between agents, especially in a real-time environment is given by [Rao92]. Furthermore, the proposal of Agent-Oriented Programming (AOP) [Shoham91, Shoham93] as a programming paradigm and AGENT0 as an agent language has influenced us when defining a language for reasoning, action and communication.

Another thread of theoretical background comes from the field of Distributed Systems. This includes synchronization, scheduling, and communication issues. But rather than trying to adapt distributed algorithms to a MAS, we identify two major problems with traditional approaches for distributed systems: the lack of an ability to reason about heterogeneity and change as such, and the difficulty in finding distributed algorithms that are both efficient and fault tolerant. For a good overview see e.g. [Coulouris88] or [Tanenbaum92]. Therefore, we view MAS and AOP as new vehicles for designing DA systems with distributed control.

3 Architecture of the DA-SoC

The general architecture of the DA-SoC is as follows:

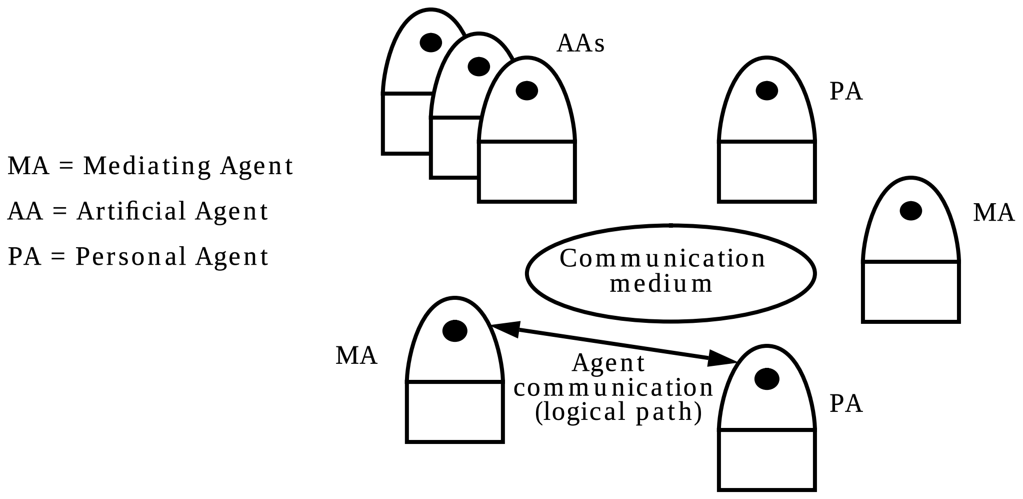


Figure 1 *A society of communicating agents*

Mediating agents maintain different services to other agents in the society, utilizing the resources on the communication medium at hand. Resources could be e.g. a database, a communication line, or an interface to a controlled industrial process. A personal agent is an intelligent user interface, and an artificial agent is the society's primary problem solver. The type of communication medium is not specified; the existence of a transport service (cf. OSI Transport Level) is assumed, upon which agent communication is built.

The agent head consists of three parts: a monitor for communication with the body and with other agents, an interpreter for the agent language, and an internal database. The database contains models of agents' object world (beliefs), a set of goals and plans to direct its actions, and a Model of Interaction (MoI).

The DA Agent Language (DAAL, see Appendix) consists of three

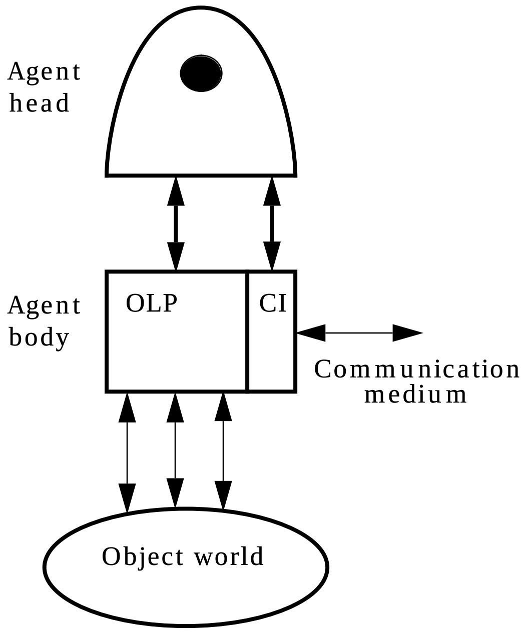


Figure 2 *An agent*

The agent is divided into two parts (***Figure 2***), a head and a body. The agent body consists of an object level program (OLP), which can be any kind of traditional program, and a communication interface (CI).

For an MA, the OLP can be e.g. a database interface or a real-time control system. For a PA the OLP can be a graphical system, while an AA has a minimal body with only the CI. The CI is implementation dependent.

Head - body communication follows a well-defined connection oriented protocol.

types of constructs, control constructs, database manipulation constructs, and action constructs. Programming the agent head means first defining a MoI, which is built up of a set of Interaction Plans, expressed in DAAL. Then Agent Plans are programmed in DAAL, and finally the agent is given initial beliefs and goals. The ability in DA-SoC to actually program the MoI allows the designer to tailor agent interaction and coordination of joint activities.

The models of the body, of other agents, and the agent's mental state is declarative, using the BEL operator. BEL(ME time_t fact_f), where fact is on the form fact_id(parameter1 parameter2 ...), means that at time time_t this agent believes fact_f. If fact_f relates to the agent body, this belief is part of the body model; otherwise it is part of the mental state. If agent a holds BEL(b time_t fact_f), it means that this belief is part of a's model of b. In this case, the belief is in the database, due to a message from b, a message from another agent about b, or caused by some reasoning related to b.

Reasoning within the agent head is procedural and goal-driven. When the goal GOAL(a t fact) is set, it is entered into the goal queue, for effectuation at time t. It is then (at time t) matched with plan declarations in the plan library, and upon success, the matched plan is executed.

The ability to create models, to reason about them, and changing them, gives a tool for distributed decision-making with a minimal dependency on communication. An agent has beliefs about the capabilities and states of other agents. These beliefs can be changed dynamically, without reconfiguration of the whole system; and at the time of a decision, it can be made without explicit communication. This is an important property, as the application is dependant on an unreliable communication network.

At this stage of the project (and the current status of the testbed), the model structure is quite simple, as described above, but it is possible to give the agents learning abilities, thereby emphasising the independency of agents. This is also one of the items for future activities in the project.

4 Testbed Description

4.1 General Concepts

The first implementation is an emulation of the agent society on a single IBM compatible PC, based on MS-DOS/Windows operating system. The Dynamic Data Exchange (DDE) communication mechanism is used for data transportation services.

When developing applications, the testbed allows the programmer to create agents, program them and enter them into the society of already executing agents. They can independently be stopped, modified and restarted.

4.2 Implementation

The testbed consists of two Windows applications:

- *Agent Manager* and
- *Agent Setup Program*

The Agent Manager is the overall controlling application, with features as listing all the agents in the society and logging all messages between agents. It is configured as a DDE-server.

Every agent has its own instance of the Agent Setup Program. With this program it is possible to set up the agents as desired. The configuration of an agent can include the following steps:

- Load a complete agent from disk or start with a new.
- Define a body for the agent.
- Add, delete or modify plans in the agent's head. Editing plans is done with standard Windows editors, allowing many to plans to be edited simultaneously, text to be cut and copied between plans etc. Plans can be saved to disk in a plan file, which is has a special file format, where several plans are saved in the same file. This allows the user to assemble related plans in some sort of libraries.
- Manually add or delete initial goals and beliefs.
- Define the maximum number of beliefs and goals, respectively, allowed in the agents head. This prevents running out of memory because of erroneous growing databases.
- Save parts of the agent to disk. When the agent is properly configured it can be started, stopped, modified and restarted, as mentioned above.

The Agent Setup Program allows the user to debug the agent by watching, manually adding or deleting the current goals and beliefs. They can also be logged in files. These features are available regardless of if the agents is stopped or running. The Agent Setup Program is configured as a DDE-client.

5 Load Management: A Sample DA Application

5.1 Problem Overview

As stated in Section 1.2 we will first address the load management problem in a static power grid.

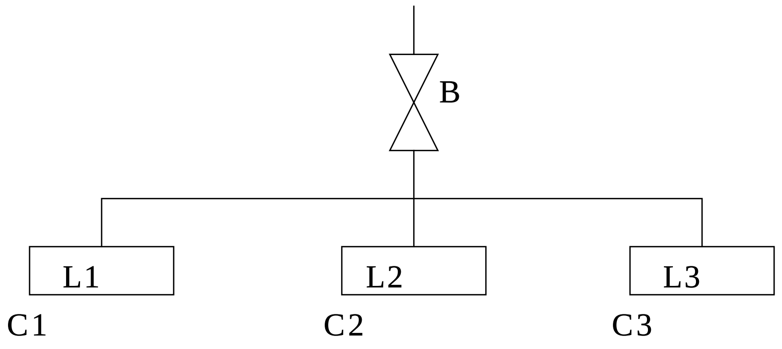


Figure 3 *A load management area*

Figure 3 illustrates a load management situation. In this particular example three customers, C1 - C3, have one controllable load each, L1 - L3, which are supplied with electricity through the bottleneck B. The loads might only be either fully connected or disconnected. There are primarily three situations when it would be desirable to disconnect one or more of the loads:

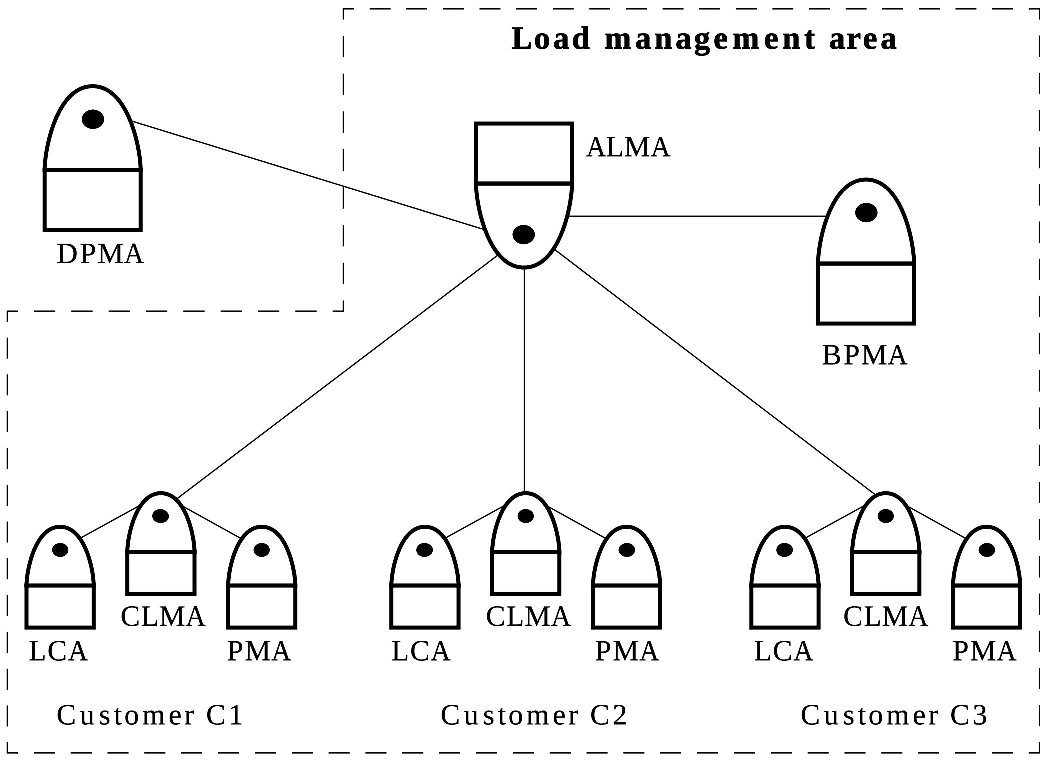
- Remote request: Shortage of power in the production or overload in the distribution grid at some level above B. The required power is larger than the power that can be delivered.
- Local control: Overload in the bottleneck B. The required power is larger than the constraint power in B.
- Customer control: The customer might want to avoid having too high a peak power.

The first one is a typical negotiation situation: the initiator of the load management operation is not prepared to disconnect loads in the load management area without considering the cost. A negotiation has to take place between the initiator and the controllers of the loads to find out which load is the most appropriate one to disconnect in the current situation, regarding the more or less complex contracts between the customers and the producer or distributor.

The second case is quite different. Here, one or more loads must be disconnected, even if it means a cost for the distributor. Otherwise, the overload might destroy the bottleneck, e.g. a transformer, which would lead to power failure for all loads. Thus, it is only a question of which load(s) to disconnect.

The third situation is similar to the first one. Here, the customer, or the agent representing him, has to decide whether to disconnect the load or pay charges to the producer or distributor.

5.2 Problem Solving Methods



DPMA = Distributor/Producer Management Agent
ALMA = Area Load Management Agent
BPMA = Bottleneck Power Measurement Agent
CLMA = Customer Load Management Agent
LCA = Load Control Agent
PMA = Power Measurement Agent

Figure 4 *Logical relations*

A DA-SoC architecture corresponding to ***Figure 3*** is given in ***Figure 4***. Case one above (remote request) is represented as a request from the DPMA. In the logical relations perspective, case two (local control) appears when the BPMA indicates that the power consumption in the load management area is too high. Finally, case three is represented as an indication from a PMA to its CLMA.

If the ALMA receives a request for load management, a negotiation takes place between the ALMA and the CLMAs, to find out the cost for the desired power reduction, and to conclude which load(s) to disconnect (if any). Parameters in the contract with a customer might be:

- how much is the distributor allowed to disconnect at various times
- the price for power at different times
- for how long are the loads allowed to be disconnected each day
- the longest period of disconnection

We note that each agent has its local view of the network and that the agent is organized in a hierarchical structure, reflecting among other things different real-time constraints and levels of competence.

5.3 Implementation

This section shows a small part of the implementation. The agent plan *reduce_power* in **Figure 5** belongs to the ALMA and represents a part of the process of detecting an overload, finding the cheapest disconnectable power, and disconnecting it. The plan handles the request for a reduction of power; it updates the ALMA's belief about how much to reduce, asks every other agent how much power they can spare and, finally, sets up a goal to handle the responses later (when they arrive).

```
AGENT PLAN $agent reduce_power($req)
                    //An agent wants to reduce power in the area
BEGIN
      ?BEL(ME $VOID more_to_reduce($to_reduce))
      // Am I currently asking customer load agents for power?
      IF SUCCESS THEN
      BEGIN // Yes, I am waiting for a reply. Redo with new values
            ~GOAL(ME $VOID check_for_responses())
                        // Cancel the check for responses.
            ~BEL(ME $VOID more_to_reduce($to_reduce))
                        // Retract the old belief about reduction
            BEL(ME NOW more_to_reduce(EVAL(Add($req
                  $to_reduce)))) // Insert new belief about reduction
      END
      ELSE // There is no current request for reduction
            BEL(ME NOW more_to_reduce($req))
                  // Insert new belief about reduction
      SACT(request($every_agent NOW ask_for_power()))
            // Ask all agents if they have power to spare
      ?BEL(ME $VOID comm_time($c_time)) // comm_time tells
            // how long I should wait for a response
      GOAL(ME LATER(NOW $c_time) check_for_responses())
            // Check for responses when comm_time has expired
END
```

Figure 5 *An agent plan of the ALMA*

The interaction plans *request* and *inform* in **Figure 6** are used by other agents to add goals to the ALMA and beliefs to its database, respectively. As they are programmable, agents can be given different ways of interaction, according to their roles in the overall system.

6 Related Work

The ARCHON architecture [Wittig92] is used for DA. The DA-SoC architecture has, in comparison to ARCHON, less complex agents, but a larger expected society of agents. Therefore they can be implemented farther out in the distribution network.

The PRS system [Ingrand92] is used for applications in several real-time domains. Compared to PRS, the DA-SoC architecture has a

```
INTERACTION PLAN request(ME $time $$pred)
BEGIN
        GOAL(SENDER $time $$pred)
END

INTERACTION PLAN inform(ME $time $$pred)
BEGIN
        BEL(SENDER $time $$pred)
END
```

Figure 6 *Interaction plans of the ALMA*

more distinct head - body dichotomy. Thereby, part of the real-time demands are given to dedicated control systems.

TEAM-CPS [Covo92] is an agent architecture for communication network management systems, with a structure similar to DA-SoC. Again, DA-SoC is designed for less complex agents and a higher degree of distribution.

The AGENT0 language [Shoham91] allows agent interaction, but our DAAL is, in comparison, more flexible with its programmable MoI. Though, the extensive analysis of the MoI lies outside the scope of this report, and will be presented elsewhere.

7 Conclusions and Future Activities

Our design and experiments with the DA-SoC testbed for Multi-Agent Systems for realizing services such as load management in static grids have illustrated the power of the agent oriented approach for this class of applications. However, much research and development remains of course, before we can have a full fledged implemented system at Sydkraft.

The next phase of the DA-SoC project will include the following:

- extensive assessments of our powerful agent language
- development and testing of different types of addressing mechanisms embedded as interaction plans in our language
- assessments of the Echelon Local Operating Network (LON) as an implementation infrastructure for the next generation of DA-SoC
- design and implementation of additional services in Distributed Automation
- work on theoretical foundations for MAS, such as team formation of agents, and articulation work in cooperation between agents
- introduction of learning abilities within agents
- identification of suitable tests for assessments of quality of service for different cooperation models

The next version of DA-SoC is expected at the beginning of next year.

Appendix: The Agent Language

```
agent:             string | var | ME | SENDER
basic_condition.   SUCCESS | term relation term | (condition) |
                   NOT basic_condition
block:             BEGIN statementlist END
boolop:            AND | OR
condition:         basic_condition boolop condition | basic_condition
criterion:         string // pre-defined or user-defined function
expression:        EVAL(function(plist))
fact:              fact_const | fact_var
fact_const:        fact_id(plist)
fact_id:           string
fact_var:          $$string
function:          string             // pre-defined or user-defined
modal:             modalop(agent time predicate)
modalop:           string
plan:              AGENT PLAN agent fact_const statement |
                   INTERACTION PLAN modal statement
plist:             pterm plist | ε
pterm:             string | var | expression
relation:          = | < | >
seconds:           // float
statement:         ACT(fact) | SACT(modal) | GOAL(agent time fact) |
                   ~GOAL(agent time fact) | BEL(agent time fact) |
                   ~BEL(agent time fact) | ?BEL(agent time fact) |
                   CHOOSE(criterion(var)) | block |
                   IF condition THEN statement ELSE statement |
                   IF condition THEN statement | EXIT
statementlist:     statement statementlist | ε
string:            // any alpha-numeric string not beginning with $
                   // and without white-space or comment marks
term:              time | agent | pterm
time:              var | NOW | LATER(time time) |
                   EARLIER(time time)
var:               $string | $VOID

DELIMITERS:        any sequence of white-space (space, tab, CR or NL)
COMMENTS:          any text starting with // to end of line (CR or NL)
CASES:             the interpreter is case sensitive
```

References

[Bratman90] Bratman M. E.: “What Is Intention?”, in Intentions in Communication, MIT Press, ISBN 0-262-03150-7.

[Cohen90] Cohen P. R., Levesque H. J.: “Persistence, Intention, and Commitment”, in Intentions in Communication, MIT Press, ISBN 0-262-03150-7.

[Coulouris88] Coulouris G. F., Dollimore J.: “Distributed Systems, Concepts and Design”, Addison-Wesley, ISBN 0-201-18059-6.

[Covo92] Covo A., Gersht A., Kheradpir S., Weihmayer R.: “New Approaches to Resource Management in Integrated Service Backbone Long Haul Communication Networks”, Proc. IEEE Network Operations and Management Symposium, 1992.

[Gustavsson94] Gustavsson R., Hägg S.: “Societies of Computation - A Framework for Computing & Communication,” University of Karlskrona/Ronneby, Research Report 1/94, ISSN 1103-1581.

[Ingrand92] Ingrand F.F., Georgeff M.P., Rao A.S.: “An Architecture for Real-Time Reasoning and System Control”, IEEE Expert, pp. 34-44, December 1992.

[Rao92] Rao A. S., Georgeff M. P.: “Distributed Real-Time Artificial Intelligence, Final Research Report, July 1990 to June 1992, available at the Australian Artificial Intelligence Institute, 1 Grattan Street, Carlton, Victoria 3053, Australia.

[Shoham91] Shoham Y.: “AGENT0: A Simple Agent Language and Its Interpreter,” Proceedings of the Ninth National Conference on Artificial Intelligence, July 14-19, 1991.

[Shoham93] Shoham Y.: “Agent-oriented programming,” Artificial Intelligence 60, 1993.

[Tanenbaum92] Tanenbaum A. S.: “Modern Operating Systems”, Part 2, Prentice-Hall, ISBN 0-13-595752-4.

[Wittig92] Wittig T., ed.: “ARCHON, an architecture for multi-agent systems”, Ellis Horwood, ISBN 0-13-044462-6.