

research project

Adaptive Building Automation

A multi-Agent approach

Ueli Rutishauser
<urut@easc.ch>

Alain Schäfer
<alani@easc.ch>

Advisors

Prof. Dr. Rodney Douglas, Institute of Neuroinformatics, ETH/University Zurich
Prof. Dr. Josef Joller, University of Applied Sciences Rapperswil

A cooperation between



Computer Science Department
University of Applied Science Rapperswil
Oberseestrasse
8640 Rapperswil, Switzerland

uni | eth | zürich

Institute of Neuroinformatics
University and ETH Zurich
Winterthurstrasse 190
8057 Zurich, Switzerland

July 20, 2002

Abstract

Modern buildings are equipped with a variety of sensors and actors like temperature, humidity or illumination sensors. A intelligent building is a building that adapts it's behavior to the needs and preferences of it's inhabitants. Such a system must be completely autonomous as inhabitants of a building dont wish to explicitly interact with it.

AHA, for adaptive home automation, is a system for controlling a intelligent building. It is desinged as a cooperative multi agent system (MAS). The basic principle, on top of which AHA is designed, is a room. A room is the the entity into which buildings are structured and used. Buildings are regarded as a collection of rooms. All analysis of data by AHA is done in the context of a particular room. The overall goal of the MAS is to satisfy the inhabitants of the building. This is achived with changing certain aspects of the building (like room temperature or light) before a human inhabitant needs to manually instruct the building to do so.

Even a small building provides such a huge amount of data to be analyzed that conventional techniques aren't applicable anymore. Decisions need to be taken in near-realtime which adds additional constraints on the methods of decision making used. Because of these AHA is distributing it's decision making to different, autonomuous, agents. Each of these agents is capable of making it's own decisions based on the input data relevant in it's context. This structure emphazises local decision making. Every agent in the system has it's own goal to pursue. To reach these goals cooperation among agents is required.

The actual decision making process is modelled as a fuzzy logic controller (FLC) which takes decisions by continually evaluating the input data against a fixed fuzzy ruleset which specifies the governing rules of the system. A fuzzy inferencing process is used to analyze the input values at hand.

The internal architecture of the agents is a mixture of the classical architectures logic agents, reactive agents and belief-desire-intention agents. Agents communicate with each other over a message based middleware that asynchronously forwards messages between agents. This communication facility is used extensively for cooperating among agents.

Contents

1	Preface	1
2	Introduction	2
2.1	Overview	2
2.2	Content	2
2.3	Structure	3
2.4	Related work	3
2.4.1	Conclusions	4
3	Motivation	5
4	Architecture	7
4.1	Basic principles	7
4.2	System Architecture	8
4.3	Software Architecture	9
4.3.1	Overview	9
4.3.2	Agents	9
4.3.3	Multiagent collaboration	10
4.3.4	Implementation requirements	10
5	Design and Implementation	12
5.1	Programming language and runtime environment	12
5.2	MAS framework	12
5.3	3rd party frameworks	13
5.4	Packages	13
5.5	Customizing	14
5.5.1	Configuration File	14

6	Bus Abstraction	15
6.1	Implementation	15
7	Decision Making	17
7.1	Feed-forward Pipeline	17
7.2	Fuzzy Logic primer	18
7.2.1	Definitions	18
7.2.2	Membership functions	18
7.2.3	Fuzzy inferencing	20
7.2.4	Fuzzy logic controllers	20
7.2.5	Fuzzification process	21
7.2.6	Defuzzification process	21
7.2.7	Rule based inferencing	22
7.3	Fuzzy decision making	23
7.3.1	Rule base	23
8	Agents	27
8.1	Overview	27
8.1.1	Agent	27
8.1.2	Agent architectures	28
8.1.3	Multi Agent Systems	28
8.1.4	Agents for intelligent buildings	28
8.2	History Agent	29
8.3	Control Agent	31
8.3.1	Responsibility	31
8.3.2	Design and Implementation	31
8.3.3	Intra agent collaboration	32
8.4	Bus Agent	34
8.4.1	Responsibility	34
8.4.2	Intra-agent collaboration	34
8.5	Boss Agent	35
8.5.1	Responsibility	35
8.5.2	Intra agent collaboration	35
8.6	Display Agent	36

8.6.1	Responsibility	36
8.6.2	Intra-agent collaboration	36
9	Experiments and Results	37
9.1	Experiments	37
9.2	Implementation Issues	38
9.2.1	LNS	38
9.2.2	Presence Detectors	39
9.2.3	Pre-processing of outputs	39
9.2.4	Intra-agent communication	40
9.2.5	Concurrency	40
10	Conclusions and Future work	44
	Glossary	46

List of Figures

4.1	A floor of a building structured into rooms	7
4.2	Network architecture	8
4.3	Layered system architecture	9
4.4	Collaboration between agents	11
6.1	Bus Abstraction Classes	16
7.1	Pipeline with 3 filters (Tasks)	17
7.2	Post-processing of decisions	18
7.3	Triangular membership function	19
7.4	Trapezoidal membership function	19
7.5	Sigmoidal membership function	19
7.6	Gauss distribution as a membership function	20
7.7	Basic layout of a non-fuzzy controller	20
7.8	Basic layout of a fuzzy controller	21
7.9	Fuzzification Process	21
7.10	Defuzzification with center-of-area/gravity	22
7.11	Input and output data for the decision making processes in AHA	23
7.12	Structure of a ARL rule base description	24
7.13	First part of the ARL ruleset (settings)	24
7.14	ARL description of membership functions	25
7.15	Fuzzy set Temperature	25
7.16	Fuzzy set daylight (indoor illumination)	26
7.17	ARL categorical variable declarations	26
7.18	ARL description of input/output variables	26
7.19	ARL block of rules (excerpt)	26

8.1	Database Schema for HistoryAgent	29
8.2	Agent relationship for HistoryAgent	30
8.3	Overview of the decision making process, modeled as a fuzzy controller	31
8.4	UML diagram of important classes in the control agent	32
8.5	Smoothened and raw input data from presence detector	33
8.6	Agent relationship for BusAgent	34
8.7	BossAgent & related Agents	35
9.1	Plot of Exterior illumination 18/19 June 2002	38
9.2	Plot of Exterior Temperatur 18/19 June 2002	39
9.3	Plot of Exterior Humidity 18/19 June 2002	40
9.4	Plot of Exterior Sunlight East 18/19 June 2002	41
9.5	Plot of Exterior Sunlight South 18/19 June 2002	41
9.6	Plot of Exterior Sunlight West 18/19 June 2002	42
9.7	Plot interior Presence, Day and Artificial Light	42
9.8	Plot interior Presence and DayLight	43
9.9	Plot interior Presence, DayLight and Blends	43

Chapter 1

Preface

By
Dr. Joseph M. Joller
University of Applied Sciences Rapperswil, Switzerland

Imagine your life in an intelligent environment (IE) : its control system manages and enables communication between its components (artificial and human ones), protects security and privacy and adapts to changing patterns. It takes care of most of the standard activities that have to be done, changing temperature, humidity, playing background music, displaying changing pictures on the wall, respecting actual moods and many more parameters of the IE.

Traditionally, these systems had to be managed in a centralized manner, often using a hierarchical system of busses, controllers and computers, plus a centralized database. In recent years it became more and more evident, that a better approach is a strong decentralization of responsibilities and a certain degree of decentralized autonomy. These decentralized intelligent components have to be coordinated and need a certain level of intelligence, so that they can act on their own.

What are the driving principles of these environments?

Current research systems (Ada [UNIZH/ETHZ], Oxygen [MIT], DAI[Sussex]) use the layer architecture pattern and an agent based software framework.

Our focus is broader and narrower: we would like to build an intelligent environment consisting of several rooms, computers, sensors, persons, with mobile components that use spontaneous networking and self adapting control systems. As IE's become more intelligent, they naturally become also more complex: we need a realistic test bed to verify our ideas in practice.

The vision of a person walking into an intelligent room and receiving interactive and multimodal assistance or support from the room is not fiction, nor 25 years ahead of us. Such a system will most likely be based on intelligent, adaptive software components, standard devices with limited functionality but significant collective power.

The work of Ueli Rutishauser and Alain Schaefer is a small but important step towards a test environment, where we can explore new ideas and test new technologies and architectures. The test system uses a layer architecture , a (multi-) agent based framework (ABLE) with a fuzzy rule system (ARL). The project showed several typical effects of such a fuzzy control systems (controls switch on and off all the time as soon as their values approaches the switching level). The solution you can find inside their report (fuzzy rule set).

The report explains a first step, the "Semesterarbeit" part of their work. After reading it, I look forward to see the results of the next step, the diploma thesis!

Good luck!

Chapter 2

Introduction

2.1 Overview

Modern industrial buildings are equipped with a huge variety of sensors and actors that are connected to each other over a standard field bus network like LonWorks or EIB. A static, fixed programmed controller unit is usually used to enforce some simple rules like turn off all lights after 22:00 or bring down the blinds when there is a certain amount of outside radiation. But such a controller has a very limited capability in terms of intelligence. It is not able to adapt it's behavior dynamically to the needs of the buildings inhabitants.

Driven by intelligent software in the background this infrastructure could be used to achieve far more advanced functionality. Such a system would be capable of continually analyzing the incoming data from all sensors and adapt the output to the actors according to rules that reflect the needs of the inhabitants of the building. Such a building is called a intelligent building.

Constructing such a system is very challenging because it involves a lot of different issues that need to be considered. First and most important the input acquired from sensors of such a system for even a middle sized building is just overwhelming – ways need to be found to break down this data into pieces that can be processed in near realtime so that the building can react to events as they are happening. An other distinct feature of intelligent buildings is that they are constantly changing over time. It will never reach a final state where it's behavior will be optional – inhabitants of buildings tend to continually change there behavior which means that a system controlling such a building has to continually adapt it's rules of behavior as well.

The goal of the project AHA (Adaptive Home Automation) is to design, develop and test a system which is capable of making a building equipped with sensors and actors intelligent. The project is divided into two different parts. The goal of the first part is to construct a stable software framework on top of which intelligent behavior can be implemented in the second part of the project. This document reflects the status of the project after completion of the first part.

2.2 Content

This document contains everything required to get a detailed overview of the system, it's concepts, ideas and underlying principles. A working knowledge of agents, multi agent systems (MAS), control theory, fuzzy logic and distributed computing is required but introductions to specialities within this areas are given. Citations of literature about topics not covered in these documentation are given. Furthermore a short introduction to fuzzy logic and fuzzy logic control is given to summarize the knowledge required to understand the system.

This document does intentionally not contain detailed class diagrams or detailed documentation of the source code. This information can be found in the separately available javadoc documentation [AJD] which is automatically generated from the source code. All documentation that is directly related to the source

code is part of the respective source files. References to the respective source files, where further information is located, are given throughout the document for readers who want to get more into detail.

AHA is a distinctively research oriented project. Due to this fact the emphasize during the whole first part of the project was not on traditional software engineering but rather on exploring different architectures to find a way how such a system could be built. There is no previous experience on building such a system and thus there is no way traditional software engineering processes could be applied. These processes require fixed assumptions that can be made in early stages of the project which is not possible in a research project like this one. This document therefore focuses on presenting the main ideas and principles of the design and architecture of AHA and not on the details of the implementation itself.

2.3 Structure

This document is structured into four parts. The first parts gives an overview of the whole system and it's basic principles. This includes the chapters 3: Motivation, 4: Architecture, 6: Bus abstraction and 7 Decision making. It also includes the "related work" section (2.4) which gives an overview of related projects done at other institutions.

The second part of the document gives details of all the agents of the system: control agent (chapter 8.3), bus agent (chapter 8.4), history agent (chapter 8.2) and display agent (chapter 8.6).

In the third part (Chapter 9) results of long-term experiments conducted within the real environment are given. This includes diagrams of long-term processes which clearly show the dynamic and unpredictable nature of the overall system.

Lastly our future intentions for the continuation of this project are discussed in Chapter 10. It's not the intention to clearly draw a plan of how we are going to continue but rather to present a number of ideas of which we are going to pursue some further.

2.4 Related work

Intelligent building research is a new research area and thus just a few other intelligent building research projects are currently pursued by other institutions. The features of some of these projects are presented in this section.

The University of Essex has an active research group called "Intelligent Inhabited Environments" which published a lot of papers about the topic. A particularly interesting one that is related to our work is "A Soft-Computing DAI Architecture for Intelligent Buildings" ([VC00]). In contrast to us this research group focuses on embedded agents ("hard" agents). This adds a lot of additional constraints like limited energy consumption, size of hardware and very limited memory/processor resources at hand. The physical and logical entity in [VC00] is the room. Every room has a number of sensors which are controlled by a agent which is physically located in the same room. All of these embedded agents are connected with each other via a computer network. Sensors and actors are wired with a low-level building automation network like LonWorks. Decision making (rule based) is done with fuzzy logic whereas learning of these rules is achieved with genetic algorithms. Manual (explicit) feedback by users is required for training.

The MIT Artificial Intelligence Laboratory is one of the other entities that are doing research in intelligent buildings. There research is mainly focused on intelligent rooms and not intelligent buildings itself. A good example of this work is the intelligent meeting room project ([Bro97]). The aim of the intelligent meeting room is to help people in the room to achieve certain tasks. It gets its explicit commands from the people through speech recognition and vision processing. An example of a interaction would be a person who points at a certain point on a map and asks the room "where is that?". The computer would then answer: "this is zurich". The project focuses on the human-machine-interface rather than on learning of the behavior of the people itself.

An other interesting perspective is presented in "Artificial Decision Making Under Uncertainty in Intelligent Buildings" ([BDY99]). This paper focuses on the decision making under uncertainty itself. It is partly very similar to our project as it is also using a multi agent system (MAS) architecture and LonWorks for connecting to sensors and actors.

Ada ([EBB⁺02]), a project done at the Neuroinformatics Institute of the Swiss Federal Institute of Technology and the University of Zurich, regards a room as a artificial organism. Ada has an emotional state just like every other organism. She expresses here internal states to visitors and interacts with visitors. Input sensors are acoustic/speech recognition, vision processing and touch (floor). The floor of Ada is divided into a large number of tiles. Every of these tiles is a autonomous system that provides input to the overall system and gives feedback from the system. Every of these tiles measures the weight of the person standing on it. These weights are important inputs to the system. Furthermore every of these tiles can emit light of an arbitrary color. Output sensors are video, audio/voice and light (light fingers and tiles). The goal of Ada is to dynamically change its overall functionality and quality through an active dialog with visitors.

2.4.1 Conclusions

Compared to the related projects discussed above ([EBB⁺02], [BDY99], [Bro97], [VC00]) AHA has the following distinct features which significantly make it different from these projects:

- AHA is based on a multi agent architecture that is fully implemented in software (soft agents)
- The scope of the system is a single building and not a particular room
- Buildings are regarded as a collection of rooms. None of these rooms is conceptually different from the others, there is no such thing as a meeting room or a office.

Chapter 3

Motivation

Over the past years building control networks have become common in commercial buildings. This was due to the big energy and cost savings they facilitated. These classical control systems mostly only operate on a few parameters. Like a controller varying heat output in relation to a sensed temperature. But they are not able to automatically adapt to changes in the environment and to changing behavior of the inhabitants. Optimization of energy use was done, but they still miss a system to make the inhabitants life easier. It should for example switch the light on just a few seconds before one enters a room. A room should always have the temperature that its inhabitants prefer. Obviously some people like it warmer than others do.

This would be very complex if not even impossible to achieve with classical rule based systems. Such a modern building has strong similarities to machines. For example, both deal with highly complex environments, have high dimensional inputs of a imprecise nature and deal with moving people or items which behave idiosyncratically.

Further similarities are revealed when we consider how the intelligent mechanisms of both systems work. A machine such as a robot moves in response to sensory information. A building can also be looked at as a robot moving in space [VC00].

In the past people have been trying to make machines intelligent by using machine learning methods. So why not try to apply machine learning technics to a building control system and thus create an intelligent building?

The intelligent behavior in AHA (Adaptive Home Automation) is realized by a system of multiple interacting agents. In such a system the agents roles are :

- ability to learn and predict a person's needs and adjust the system to meet this persons needs.
- tell these conclusions to other agents
- do such learning based on a wide set of imprecise data

The agents have to cope with highly complex situations where every action from an agent might cause an opposite reaction from a person, which again might cause reactions of other agents (non-episodic environment). It has been shown by [Wei99] that such behavior is best controlled by a Multi Agent System (MAS).

With the vision of an adaptive, learning building control system in mind we focused in this first stage on a basic framework for such a system. The decision was taken to implement a framework for a MAS with control agents, who control the rooms based on some fuzzy rule based decision making. The choice of fuzzy rules based decision making was taken because it was, as a first step, the easiest ML technology to implement. Fuzzy Rules were also considered to be an interesting technology and the learning curve still being flat enough for the available time.

Java as a programming language is excellently suitable for MAS, because of its wide availability of libraries

and frameworks for ML. A quick search for Agent Toolkits for Java returns a myriad of results. Since good experiences with ABLE, a Java Agent Framework, have been made at INI, ABLE was chosen to be used for AHA too.

Chapter 4

Architecture

This chapter will outline the general architecture of the adaptive building automation (also called adaptive home automation, abbreviated as AHA) system. The architecture is presented from two perspectives: from the software perspective and from the system perspective. The system perspective shows how the different parts of the system are connected to each other and where the different software parts are located physically. The software perspective depicts how the system is structured into different agents and how they interact with each other.

4.1 Basic principles

Traditionally every room in a building serves a certain purpose and is thus dealt with differently by the people using it. To reflect this in our system design the main logical concept on which the architecture of AHA is based is a *room* (Figure 4.1). All sensory input values, except for external input like weather data, are regarded as being related to one single room. This is also valid for decisions taken by the system. The system evaluates the data available and takes decisions about the state of every room in the building. This enables AHA to quickly adapt to differences in the usage of different rooms.

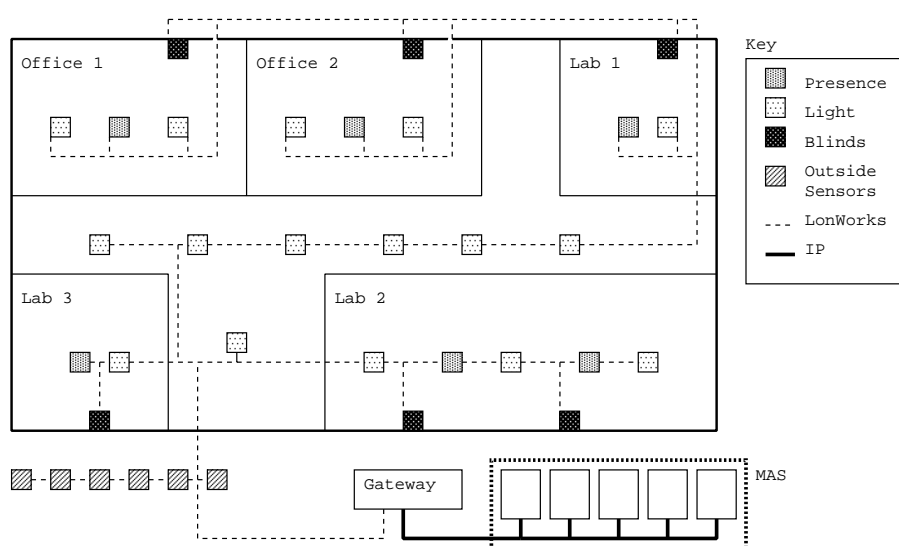


Figure 4.1: A floor of a building structured into rooms

Our agents are logically related to a room but physically they aren't. In contrast to systems which emphasize the use of physically distributed embedded agents we are using soft agents which are running as a piece of

software on a centralized computer system. This enables us to deal with the real issues posed by the system because we don't have to focus on the limited availability of resources that are prevalent in embedded agent systems.

Agents as a software architecture paradigm offer a very flexible way of designing systems in a way such that they are flexible, non-static and scalable. AHA works with a limited number of agent types (5 agents). Some of these agents are just running once whereas other agents are having a large number of instances. All instances of these agents together form the multi agent system (MAS) that is responsible for controlling the intelligent building (IB). Agents are created dynamically as they are required (for instance when there is a new room added to the system). The focus on the concept of a *room* is reflected in the design of the multi agent system by the fact that there is one single instance of an agent that is responsible to take decisions about a single room. This emphasizes local decision making. Decisions about a room can even be made when agents on a higher level become unavailable.

4.2 System Architecture

The overall AHA system is a combination of two completely different network architectures: field bus and a computer network (like Ethernet is). A modern office building is typically wired with these two bus systems. AHA requires access to both, the field bus and the computer network of a building. The field bus (in our case LonWorks) is used to communicate with the sensors and actors of the building automation system whereas the computer network is used to communicate between the different agents of the system.

The two networks are connected to each other with the help of a gateway that is connected to both networks. The gateway forwards data between the computer network and the field bus such that all software parts of AHA can run on the computer network without any direct access to the field bus.

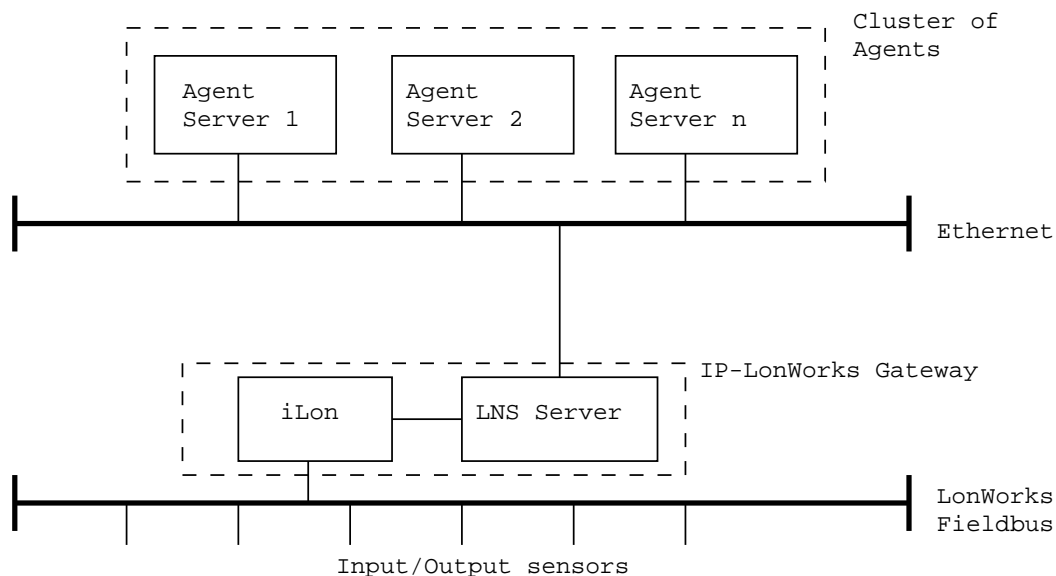


Figure 4.2: Network architecture

Figure 4.2 shows the system architecture of AHA. AHA isn't dependent on any specific kind of field bus system, the LonWorks field bus standard is just used as one possibility in the figure. LNS and iLon are LonWorks specific products that act as an IP-LonWorks gateway. Communication between iLon (which is the real LonWorks-IP gateway) and the LNS server (which is just a software) is already done over IP, so the only hardware device that requires direct access to the LonWork bus is the iLon. The only constraint by the LNS system is that the BusAgent must run (physically) on the same machine than the LNS server. This is due to the fact that the LNS Java API is not capable of making a remote connection over the network to the LNS server.

4.3 Software Architecture

4.3.1 Overview

The system is realized as a collection of agents which together form a multi agent system (MAS). Every agent is responsible for one specific task and offers this task as a service to the other agents. Agents that require functionality they don't offer themselves collaborate with other agents that offer the required capability to achieve its aims. There is no central agent that acts as a kind of coordinator, all agents act independently of each other.

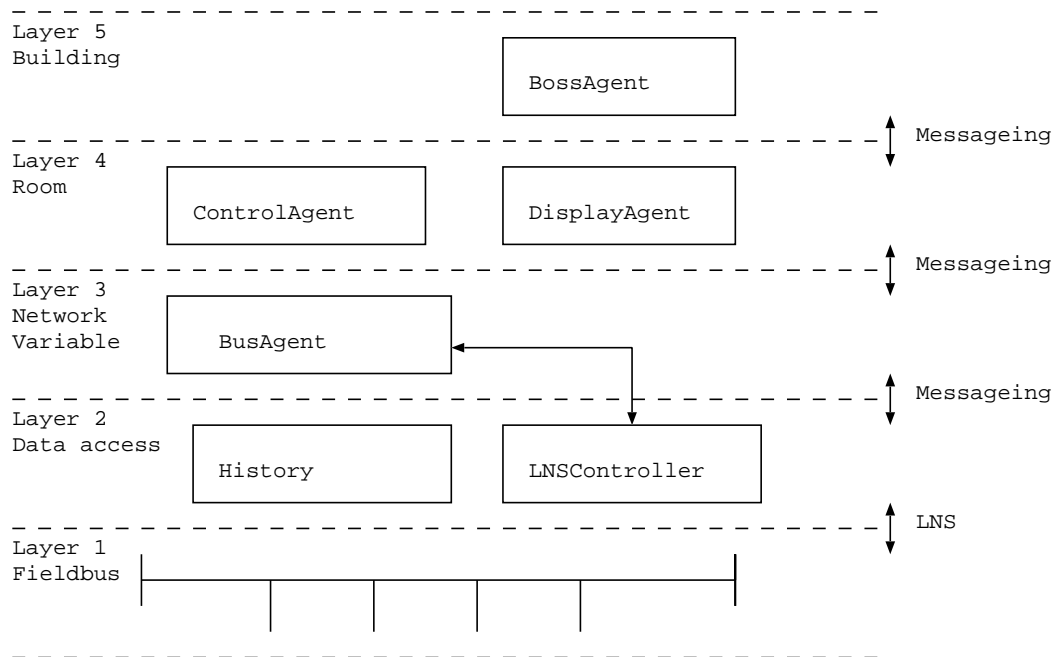


Figure 4.3: Layered system architecture

The MAS is structured into five layers (Figure 4.3). Every agent in the system belongs to one of these layers. The layering is not strict – it is possible for an agent to communicate directly with another agent that is not directly one layer above or below the layer it belongs to. The aim of the layering is to structure the system in a way that higher levels have to deal with less data (more concentrated) than lower level agents.

The five layers are (Figure 4.3):

- Layer 5: Building, everything related to the overall building or multiple buildings
- Layer 4: Room, everything directly related to a room
- Layer 3: Network Variable, everything that directly deals with a network variable or a property
- Layer 2: Base Services / Data access (supporting services, gateways)
- Layer 1: Network abstraction layer (hardware, wire)

This layered architecture follows the guidelines given in [Bro97] and [Bro86].

4.3.2 Agents

The following types of agents are used:

- Bus Agent: field bus abstraction on the level of network variables
- RoomDisplayAgent: displays all information available about one particular room; manual control of all output sensors
- Control Agent: processes input data and takes decisions; distributes and executes decisions
- History Agent: logs relevant data in a database for later use (offline analysis)
- Boss Agent: responsible for dynamically creating new instances of agents as new sensors, rooms or buildings become available.

Multiple instances of these agents are running at the same time. All these instances together form the multi agent system.

Instances are created according to the following rules:

- Bus Agent: Once per Field bus system type (e.g. LonWorks, EIB)
- RoomDisplayAgent: Arbitrarily on demand of users, not required for productive use
- Control Agent: Once per room
- History Agent: Once per log database
- Boss Agent: Once per system

4.3.3 Multiagent collaboration

Communication between agents is not done by directly calling the remote functionality with a technology like RMI (Remote Method Invocation). This would make the system too dependent of each other because agents would have to know exactly what other agents are around in the system and what functionality they offer. Because of this, all communication between agents is done over a middleware that is responsible for distributing messages to the agents that are interested in its content. Agents register themselves with other agents from which they want to receive messages of a certain type in the future. The sender of a message does not have to know how many (if any) agents are interested in the message it sends out. This is rather the task of the middleware. It is also the responsibility of the middleware to automatically remove agents that went down without unregistering or that aren't processing messages anymore. This asynchronous message passing mechanism offers a lot of flexibility which makes a multi agent system much more flexible and dynamic.

Figure 4.4 shows how agents are exchanging messages (only inter-agent communication is shown, intra-agent communication takes place but is not shown). There are three types of messages: NV (network variables) set/get, properties set/get and decisions. The lowest level of abstraction is the level of network variables (NV). NV values are exclusively received and sent by the BusAgent. The syntax of NV's is proprietary and depends on the field bus standard used. The second layer of abstraction in communication is the layer of properties. Properties are the system independent, generic, representation of network variables. Agents can register as listeners for properties so that they get notified by the BusAgent when there is a new value available. Agents can also tell the BusAgent to assign a new value to a property. The highest level of abstraction is the level of decisions. These messages are only sent out by the control agent and can be received by any agent interested in the decisions taken by the system. The control agent also executes the taken decisions which means that the control agent also sends out property set values which reflect the execution of the decision taken.

4.3.4 Implementation requirements

The requirements for the programming language and runtime environment of AHA are:

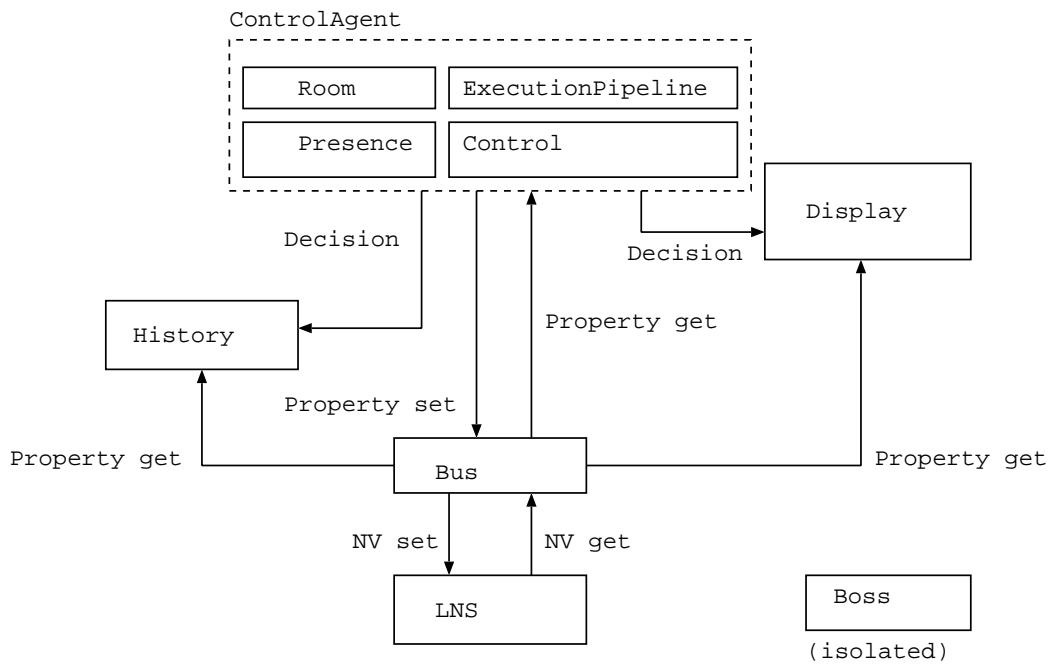


Figure 4.4: Collaboration between agents

- Platform independence: AHA itself is designed as a multi agent system (MAS). It should thus be possible to run different parts of AHA on systems with different operating systems
- Libraries: There should be enhanced libraries for the construction of multi agent systems available
- Distributed computing: the runtime environment should support distributed computing (distributed garbage collection, remote method invocation)
- Availability of API's for field bus systems: API's for the major field bus systems should be available for the programming language chosen

AHA requires an existing advanced MAS framework. Evaluation criteria for such a framework are:

- Support for physically distributed agents (distributed among multiple workstations)
- Asynchronous messaging (for exchanging messages between agents)
- Support for intra-agent communication
- Support for decision making techniques like fuzzy logic inferencing or ILP (inductive logic programming)
- Support for concurrent programming
- The framework needs to be stable and mature

An appropriate programming language, runtime environment and MAS framework was chosen according to these conditions. The choices made are described in chapter 5 (Design and Implementation).

Chapter 5

Design and Implementation

This chapter outlines the technologies used to build AHA. These technologies were chosen because of the requirements and constraints given in chapter 4.

5.1 Programming language and runtime environment

After evaluating several options according to the requirements given in section 4.3.4 it turned out that the programming language (and runtime environment as well) of choice is Java. This is due to the following reasons:

- Libraries: Enhanced libraries for the construction of multi agent systems are available ([ABL])
- Platform independence: AHA is distributed among different systems, some of which are running different operating systems because of additional system constraints (native API's which require a particular platform).
- Distributed computing: suitable for distributed computing because it features distributed garbage collection (DGC)
- Availability of API's for field bus systems: There is a Java API available for every major field bus system

An other reason for using Java for AHA is that a lot of related projects with which interoperability might be crucial in the future are implemented in Java as well.

5.2 MAS framework

Few mature, stable and advanced MAS frameworks are available for Java. There is a lot of literature and theory about MAS systems but there are actually remarkably few implementations of MAS frameworks available. One of the most enhanced ones available is ABLE (Agent building and learning environment), provided by IBM alphaworks ([ABL]). We decided to use ABLE because of the following reasons:

- ABLE is 100% native Java
- ABLE is stable and mature
- ABLE was successfully used in several other large projects, one of which is Ada ([EBB⁺02]).

- ABLE provides advanced inferencing mechanisms like fuzzy logic inferencing.
- ABLE provides mechanisms for intra-agent communication between concurrently running parts of a agent
- ABLE is build on top of the JavaBeans architecture which provides a very easy way to integrate further extensions into it.

The condition that isn't met by the ABLE framework is the asynchronous messaging between agents. ABLE doesn't provide this functionality so that we had to implement it ourself. Details of the architecture of our messaging middleware are given in 4.3.3.

5.3 3rd party frameworks

Besides of the MAS framework some other 3rd party libraries are used. These are:

- JFreeChart ([JFR]): Library that provides basic charting capabilities
- JCommon ([JCO]): collection of useful classes; Required by JFreeChart.
- LNS Java SDK ([LNS]): Java API for accessing the LNS server. This API is a product provided by Echelon (LonWorks).
- Log4J ([L4J]): Logging framework, manages debug/warning/error output, distributed logging
- ANTLR ([ANT]): general purpose parser generator. We use it to parse ARL files

All of these tools and libraries are provided under a opensource license and are thus freely available for everyone.

5.4 Packages

The source is structured into the following packages:

- aha.control: Control Agent
- aha.bus: Bus Agent (generic, field bus system independent parts)
- aha.bus.lonworks: LonWorks specific implementation
- aha.framework: Utility classes, Supporting classes
- aha.history: History Agent
- aha.middleware: Common message format definitions
- aha.rules: fuzzy logic rules (in ARL)

5.5 Customizing

All agents read their configurable values through the static class *Config*. The configuration variables are not organized in namespaces, so developers should be careful to avoid naming conflicts.

The actual implementation for the *Config* class reads their values from a Java properties file. Since the *Config* class is used from different Agents and different Processes, locating the config file is not trivial. It was decided to look for the config file at two locations, either */etc/ahaConfig* or *.ahaConfig* in the startup directory.

5.5.1 Configuration File

The following options can be configured (with an example for every option):

```
#the regular expression patter which matches for different rooms in the bus network variables
RoomPattern=.*-([0-9]{4})_.*
#directory where fuzzy logic rules should be read from
ruleDir=./src/aha/rules
#reciprocal of the sampling frequency for historyAgent, in millisec
historySampling=30000
#variables which should always be sampled even if no listener is subscribed for
#comma delimited list
standardVariables=Y344G-41_-GRELLOW.vnoG91___080MB1.30
#jdbc connection string for the history database
dbConnection=jdbc:mysql://localhost/aha
#size of the history buffer in busAgent
historyBuffer=100;
#class name for the jdbc driver
dbDriver=org.gjt.mm.mysql.Driver
#name of the log4j config file
log4j=.log4j
#class name for the BusController instantiated by BossAgent
BusController=aha.bus.lonworks.LNSController
#ip for the lonworks bus server
lonBusIP=130.60.71.231
#port where lonworks bus server is listening
lonBusPort=2540
```

Chapter 6

Bus Abstraction

Preliminary discussions on the design lead to the idea to encapsulate the access to the bus system. A full encapsulation of the bus access layer would make the system much more portable to other environments as it could be used for different bus systems with only minor modifications.

Analysis of different HLPs (High Level Protocol) for home automation bus systems like ([LNS], [EIB], [COP]) lead us to the conclusion that the core of such a bus system consists of the device variables and their values. At first this seems surprising as the main concept of a field bus system is usually a device. For a system accessing the bus from outside this is rather unimportant as every variable of every device is mapped to a network variable that has a network wide unique name.

Most Java programmers are very familiar with the Java Beans interface specification. Thus we choose to implement this bus abstraction with an interface very similar to the Java Beans specification. Similar to Java Beans we call the variables in the bus network properties. The difference is that the names of these properties are not hardcoded in the interface. But still users are able to register for property changes over the common *PropertyChangeListener* interface.

6.1 Implementation

An implementation for the LonWorks system, a leading bus system for building automation and control, was developed during this project. Implementations for other types of bus systems could easily be added.

This implementation was done using the LNS HMI Developer's Kit [LNS], which gives easy access to any LonWorks device managed by an LNS Server. It also supports listeners for network variable changes, which use a bandwidth efficient push technology to notify the listeners of changes.

Our implementation, the class *LNSController*, uses *DeviceListener* to allow several agents to share one [LNS] *RawUpdateListener* (figure 6.1). For more details please consult the AHA Javadoc

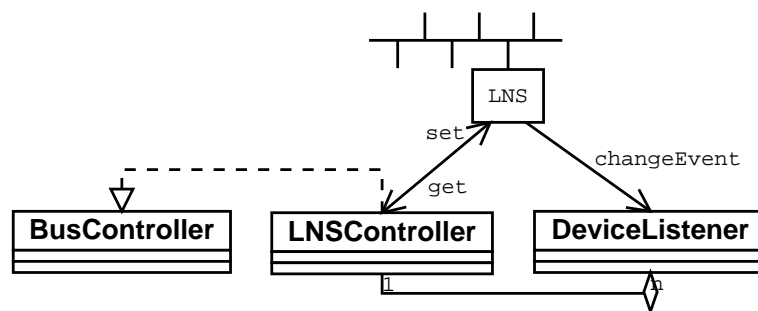


Figure 6.1: Bus Abstraction Classes

Chapter 7

Decision Making

One of the main tasks of an intelligent control system is to take decisions. All other parts of the system are there to support the decision making process. This includes acquisition of the data, pre-processing, storage, history and post-processing (including execution of decisions).

This chapter begins with an introduction to important basic aspects of fuzzy logic. It continues with a detailed explanation of the design for the different decision making processes and their supporting infrastructure.

All decisions in AHA are made on a high level which requires the data to be heavily pre-processed before they are fed forward to the inferencing process. To make this complicated process more flexible a feed-forward pipeline is used. The feed-forward pipeline is a very flexible software structure that allows to plug-in an arbitrary number of processes which deal with all data that are fed to the pipeline in a sequential way.

7.1 Feed-forward Pipeline

The pipeline software architecture pattern is a typical application of the pipes and filter architecture. It is a generic architecture for the sequential processing of data. The processing itself is divided into small tasks which get data from the pipeline as input and return some data to the pipeline after execution. This data is then taken back by the pipeline which passes it further on to the next task registered in the pipeline (Figure 7.1).

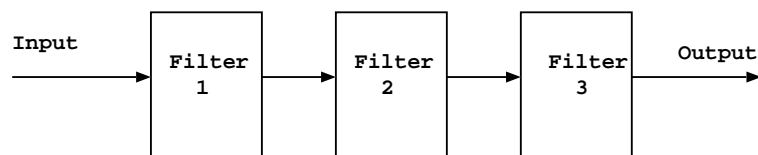


Figure 7.1: Pipeline with 3 filters (Tasks)

Decision making is a sequential process (pre-processing, inference-making, post-processing) and as such a perfect example for a pipeline architecture.

Post-Processing in AHA involves many different independent tasks that need to process the output of the inferencing in some way and communicate it to other agents. It is implemented as an 4-step pipeline (Figure 7.2). Post-processing directly ensues defuzzification.

The following section refers to Figure 7.2. The first registered task in the pipeline is the Able Event Gateway. This task is responsible for communicating decisions to other agents in the system. It acts as an extension of the local pipeline to remote pipelines. It uses the Able Event processing middleware for this task. Local

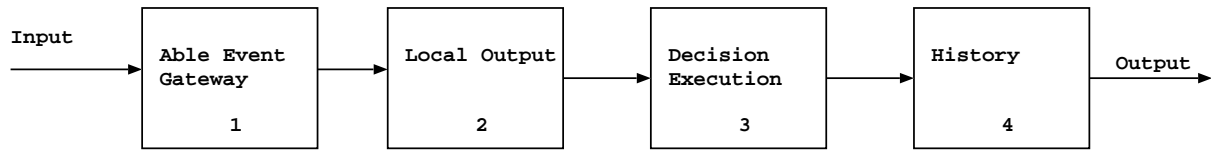


Figure 7.2: Post-processing of decisions

output is the task responsible for updating all local user interfaces that are directly controlled by the agent where the decision making is done. This consists mainly of debug information output. It is the responsibility of task number 3 to execute decisions taken by the inferencing engine. Execution involves deciding whether a decision should be turned into reality or not (e.g. switch a light off). There are many reasons why a decision might not be executed after all – one of the most important reasons not to execute a decision is the prevention of oscillation, e.g. avoidance of frequent changes of decisions to emphasize the lethargic nature of the system. The last task which gets decisions from the inferencing engine is the gateway to the history. This task sends all relevant data to the history agent which makes this data available for statistical analysis of the behavior of the system.

7.2 Fuzzy Logic primer

Proofs for the following definitions are in [Ful00].

7.2.1 Definitions

We always assume that X denotes a non-fuzzy set and A denotes the corresponding fuzzy set.

The Membership Function (7.1) is a function that assigns every $t \in X$ a value $0 \leq x \leq 1$.

$$\mu_A : X \rightarrow [0, 1] \quad (7.1)$$

Equation 7.2 shows a frequently used short form for the membership function.

$$A(x) = \mu_A(x) \quad (7.2)$$

The support (7.3) of a fuzzy set A is the set of all $x \in X$ such that $A(x) > 0$.

$$\text{supp}(A) = \{x \in X | A(x) > 0\} \quad (7.3)$$

The α cut (Equation 7.4) of a Fuzzy set A defines a subset of X (non fuzzy).

$$[A]_{\{\alpha\}} = \begin{cases} \{t \in X | A(t) \geq \alpha\} & \text{if } \alpha > 0 \\ \text{supp}(A) & \text{if } \alpha = 0 \end{cases} \quad (7.4)$$

7.2.2 Membership functions

The following equations assume that $\alpha > 0$ is the left width and $\beta > 0$ is the right width with $\alpha \leq \beta$.

Triangular fuzzy number (Equation 7.5 and Figure 7.3). Shortform: $A = (a, \alpha, \beta)$.

$$A(t) = \begin{cases} 1 - (a - t)/\alpha & \text{if } a - \alpha \leq t \leq a \\ 1 - (t - a)/\beta & \text{if } a \leq t \leq a + \beta \\ 0 & \text{otherwise} \end{cases} \quad (7.5)$$

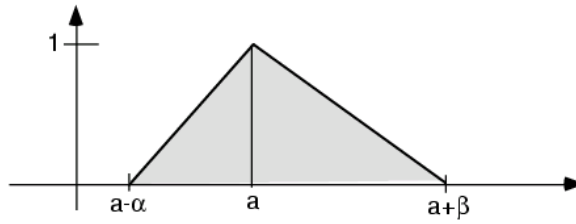


Figure 7.3: Triangular membership function

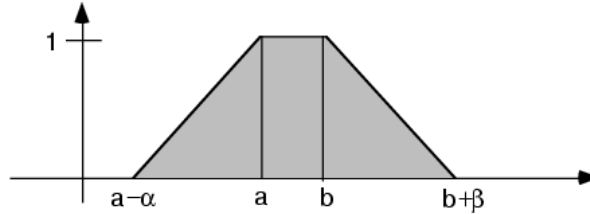


Figure 7.4: Trapezoidal membership function

Trapezoidal fuzzy number (Equation 7.6 and Figure 7.4). Shortform: $A = (a, b, \alpha, \beta)$

$$A(t) = \begin{cases} 1 - (a - t)/\alpha & \text{if } a - \alpha \leq t \leq a \\ 1 & \text{if } a \leq t \leq b \\ 1 - (t - b)/\beta & \text{if } a \leq t \leq b + \beta \\ 0 & \text{otherwise} \end{cases} \quad (7.6)$$

LR-representation (Equation 7.7) of fuzzy numbers, which is a generalized form of the trapezoidal representation. Shortform: $A = (a, b, \alpha, \beta)_{LR}$

$$A(t) = \begin{cases} L((a - t)/\alpha) & \text{if } t \in [a - \alpha, a] \\ 1 & \text{if } t \in [a, b] \\ R((t - b)/\beta) & \text{if } t \in [b, b + \beta] \\ 0 & \text{otherwise} \end{cases} \quad (7.7)$$

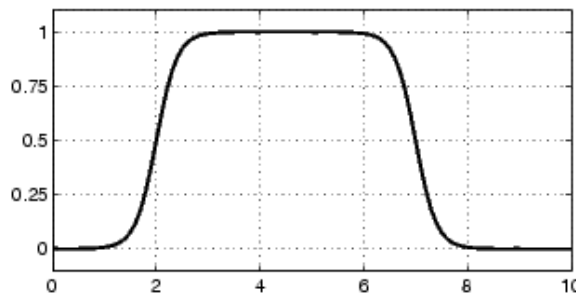


Figure 7.5: Sigmoidal membership function

Sigmoidal fuzzy number/variable (Equation 7.8 and Figure 7.5).

$$f(x) = \frac{1}{1 + e^{-x}} \quad (7.8)$$

Gaussian membership function (Gaussian fuzzy number). In equation 7.9 ρ is the standard deviation and μ is the mean of the distribution (Figure 7.6).

$$A(x) = \frac{1}{\rho\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\rho}\right)^2} \quad (7.9)$$

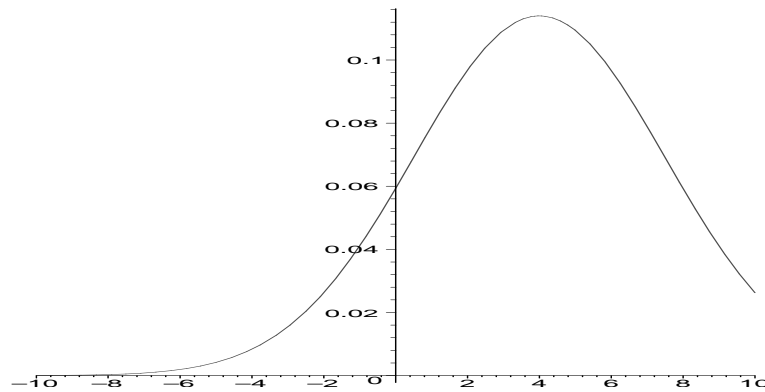


Figure 7.6: Gauss distribution as a membership function

Other membership functions that aren't explained here in detail are shoulder, linear, segments and beta function. Consult [Ful00] for details about these functions.

7.2.3 Fuzzy inferencing

The basic inferencing process involves the following steps:

1. fuzzification of the input variables
2. rule evaluation
3. aggregation of the rule outputs
4. defuzzification

This specific inferencing process is called Mamdani-style fuzzy inferencing process.

7.2.4 Fuzzy logic controllers

Control theory looks at complex systems from a black box perspective. The system to be controlled is idealized as a simple black box with an input and an output. A controller is the part of the system (Figure 8.3) which is responsible to adjust the input to the system according to the output of it. This is made in an attempt to achieve the desired output.

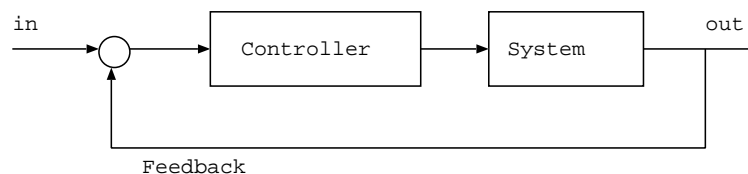


Figure 7.7: Basic layout of a non-fuzzy controller

Fuzzy logic control is built on the same principles but uses a different technique to decide what the output of the controller should be. All decisions of the controller are made on the basis of fuzzy logic rules which describe the relationship between fuzzy variables. A fuzzy logic controller (FLC) can only work with fuzzy input and output values. Because of this the input values must be converted from crisp numbers to fuzzy variables and the output values must be converted from fuzzy variables to crisp numbers. This two processes are called fuzzification and defuzzification. An enhanced version of the original control system is shown in Figure 7.8.

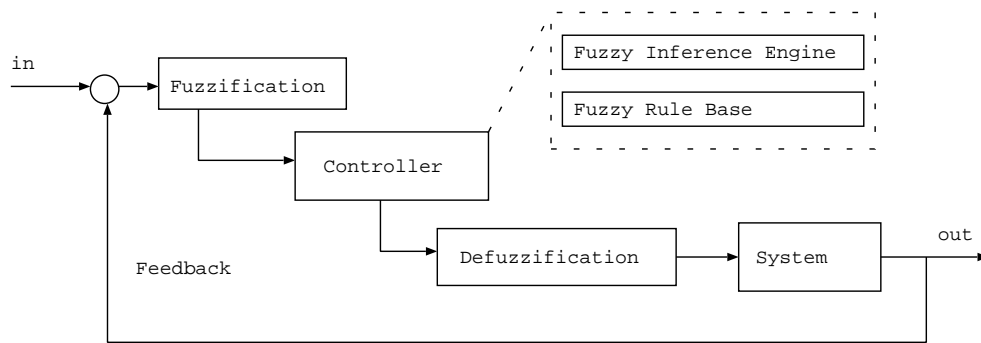


Figure 7.8: Basic layout of a fuzzy controller

A FLC consists of two different parts: the inferencing engine and the fuzzy rule base. The fuzzy rule base is a number of fuzzy logic rules which are used by the inferencing engine to reach a conclusion from the input presented.

FLC's are classified as either multi-input-multi-output (MIMO), multi-input-single-output (MISO) or single-input-single-output (SISO) fuzzy systems (see [Ful00], page 67).

7.2.5 Fuzzification process

Fuzzification is the process that uses the membership functions to convert a crisp value to a fuzzy variable. A fuzzy variable associates a membership value/strength for every fuzzy variable to a crisp value: $\mu_A : X \rightarrow [0, 1]$.

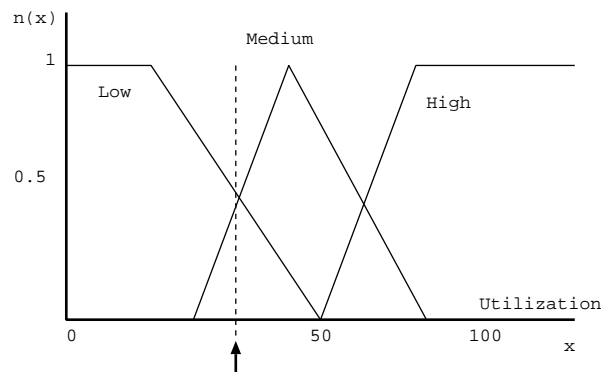


Figure 7.9: Fuzzification Process

Figure 7.9 shows an example of the fuzzification process for three fuzzy variables. If, for example, the crisp input value is 37.5 the values of the three fuzzy variables would be Low = 0.5, Medium = 0.5 and High = 0.0.

7.2.6 Defuzzification process

The end result, after all rules have been evaluated and aggregated (combined to one fuzzy set), is a single fuzzy set that represents the decision taken. To apply the output of the fuzzy inferencing a crisp value is required. Defuzzification is the process that is converting a single fuzzy set back into a crisp number. Defuzzification is deferred till the last possible point of time. This is because defuzzification is not for free. All the information that is contained in a fuzzy set is reduced to one crisp number which contains obviously less information than the fuzzy set contains. This is a considerable loss of information.

There are many defuzzification algorithms. Some of the most important ones are:

- Center of Area / Center of Gravity
- Center of Sums, Center of Largest Area
- First of Maxima
- Middle of Maxima
- Max Criterion
- Height defuzzification

Only center of gravity is explained in more detail here since this is the only method used in AHA. For details about the other methods see [Ful00].

$$z_0 = \frac{\int \mu(x)dx}{\int xdx} \quad (7.10)$$

Center of gravity (Equation 7.10) calculates the center of mass if the area of the fuzzy set is regarded as a physical area. Calculation of the center of gravity is usually not possible in continuous space. Thus, Equation 7.11 is used in practice to calculate the center of gravity of a fuzzy set that has a discrete membership function (which is usually the case).

$$z_0 = \frac{\sum x\mu(x)}{\sum \mu(x)} \quad (7.11)$$

z_0 is the crisp result of the fuzzy inferencing process (Figure 7.10).

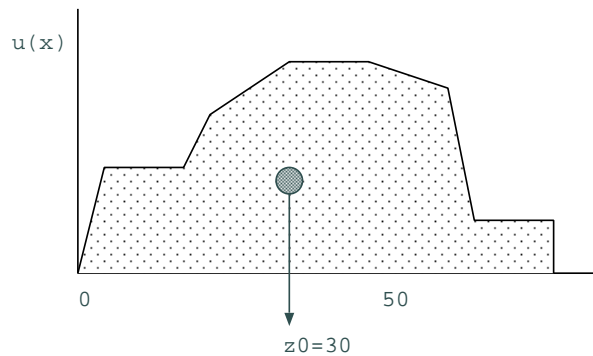


Figure 7.10: Defuzzification with center-of-area/gravity

7.2.7 Rule based inferencing

The basis for decisions taken by a fuzzy logic controller are linguistic description rules. These rules are expert knowledge that needs to be set up by a human domain expert. The rules are of the form:

- R_1 : if x is A_1 and y is B_1 then z is C_1
- R_2 : if x is A_2 and y is B_2 then z is C_2
- R_3 : if x is A_3 and y is B_3 then z is C_3
- R_n : if x is A_n and y is B_n then z is C_n

R_n is the identification of the rule, x and y are input variables and z is the output (control) variable. A_i , B_i and C_i are linguistic variables. x , y and z are fuzzy variables.

To obtain the output z_i for a particular set of input values all rules are evaluated in parallel. This involves the following steps:

1. Find the firing level of each of the rules
2. Find the output of each of the rules
3. Aggregate the individual rule outputs to obtain the overall system output

7.3 Fuzzy decision making in intelligent buildings

Sensory input data to AHA is acquired over a field bus system from all kinds of sensors. This includes temperature, humidity, illuminance, presence, switch (e.g. light switches), status of lights, status of blinds, blind switches and indoor illumination (daylight). This input data is, after pre-processing, evaluated by a fuzzy logic controller (FLC). Decisions by the FLC are directly transmitted to the responsible devices on the field bus system. Figure 7.11 shows the type of input data that is processed by the FLC and the output signals that are generated by the FLC.

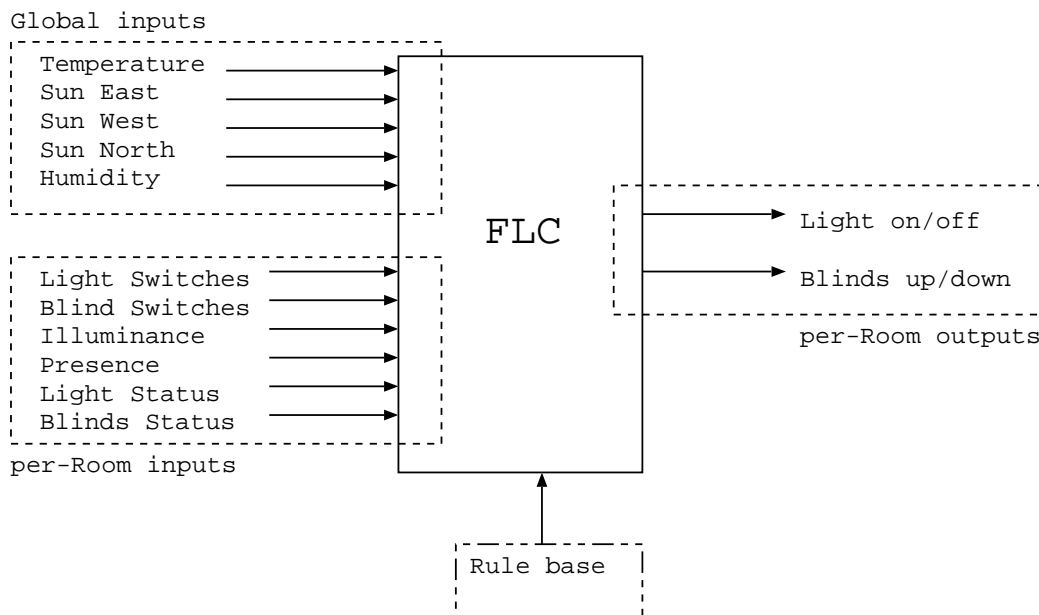


Figure 7.11: Input and output data for the decision making processes in AHA

There are two categories of input data: global inputs and per-Room inputs (Figure 7.11). Global inputs are acquired from external sensors that are available at most once per building whereas per-Room inputs are acquired from sensors that are placed in every room in a building at least once. All decisions are taken in the context of one particular room. For this every room has its own FLC running (see the chapter about Control agents for further details about the per-room decision making).

7.3.1 Rule base

The following ruleset (consisting of Figure 7.13, 7.14, 7.17, 7.18, 7.19, in this order) shows excerpts from the rule base that is used for controlling one single room. ARL (Able Rule Language) (see [IBM] for an introduction) syntax is used to describe the rules in a machine-readable format.

```

RuleSet <nameOfRuleSet> (
  <Processing Options Statement>* // Zero or more statements

  Variables( // Global variable declaration section
    <Variable Declaration Statement>+ // One or more statements
  )

  GoalVariable ( <variableName> ) // Backward chaining only
  InputVariables ( <variableName>* ) // Exactly one statement, zero or more names
  OutputVariables( <variableName>* ) // Exactly one statement, zero or more names

  UserDefinedFunctions ( <name/arity>* )* // Zero or more statements, zero or more names

  <Rule Block Statement>+ // One or more statements

) ;End of ruleset

```

Figure 7.12: Structure of a ARL rule base description

The ruleset is structured (Figure 7.12) as following (in this order):

- Settings
- Variable Declarations
- Description of Membership functions
- Specification of input variables
- Specification of output variables
- Several blocks of rules

```

RuleSet BuildingControl (
  InferenceMethod   ( FuzzyAdd )
  AlphaCut          ( 0.2 )
  CorrelationMethod ( Product )
  DefuzzifyMethod   ( Centroid )
)

```

Figure 7.13: First part of the ARL ruleset (settings)

Settings (Figure 7.13) are:

- InferenceMethod: FuzzyAdd. Specifies that the fuzzy solution set is updated by adding the minimum truth value of the consequent fuzzy region, bounded by 1.0. This method is generally used with CorrelationMethod(Product).
- AlphaCut: the alpha cut level is set to 0.2. The alpha cut level is the threshold at which truth values become insignificant (Equation 7.4).
- CorrelationMethod: Specifies that the membership value of the consequent fuzzy region is the product of the fuzzy region and the truth of the premise.
- DefuzzifyMethod: Centroid (center of mass) is used as defuzzification algorithm (Equation 7.10)

```

Temperature Fuzzy(-30.0 to 100.0) SetDefinitions (
  warm Triangle (8.0, 15.0, 22.0)
  Cold Triangle (-5.0, 5.0, 11.0)
  hot Shoulder (28.0, 100.0, Right)
  veryWarm Triangle (18.0, 25.0, 33.0)
  veryCold Shoulder (-30.0, 0.0, Left)
)

RadiationEast Fuzzy(0.0 to 10000.0) SetDefinitions (
  completelyDark Sigmoid(0.0, 10.0, 20.0, Down)
  littleLight Triangle (15.0, 50.0, 250.0)
  light Triangle (200.0, 600.0, 1000)
  sun Triangle (900, 2000, 3100)
  lotsOfSun Shoulder (3000.0, 5000.0 , Right )
)

DayLightIndoor Fuzzy(0.0 to 3000.0) SetDefinitions (
  dark Sigmoid (0.0, 150.0 300.0, Down)
  middle Triangle (180.0, 350.0, 520.0)
  normal Triangle (450.0, 1075.0 , 2000.0)
  blinding Shoulder(1500, 2500, Right)
);

```

Figure 7.14: ARL description of membership functions

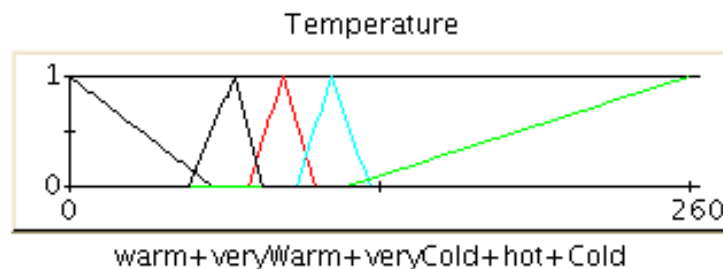


Figure 7.15: Fuzzy set Temperature

The three membership functions (Figure 7.14) temperature, radiationEast and DayLightIndoor describe the mapping between crisp values and the corresponding fuzzy value.

Figures 7.15 and 7.16 illustrate two of the fuzzy sets listed in Figure 7.14.

There are 4 input variables (Figure 7.18) to the FLC: temperature (external), radiation (external), presence (room) and daylight (room). Two of the outputs are actions (Blinds and Lights), one output is a state (RadiationStatue, for further calculation).

This simple ruleset (Figure 7.19) doesn't have fuzzy output values so there is no need for defuzzification. All output variables are categorical (discrete) states like DOWN, UP, ON and OFF.

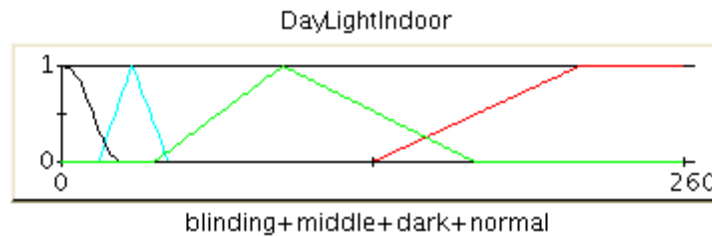


Figure 7.16: Fuzzy set daylight (indoor illumination)

```

Variables (
  BlindAction Categorical("UP" "DOWN")
  LightAction Categorical("ON" "OFF")
  BlindState Categorical("UP" "DOWN")
  LightState Categorical("ON" "OFF")
  RadiationState Categorical("lotsOfSun" "littleLight" "sun" "completelyDark" "light")
  Presence Categorical("YES", "NO")
)

```

Figure 7.17: ARL categorical variable declarations

```

InputVariables (Temperature RadiationEast Presence DayLightIndoor )
OutputVariables(BlindAction LightAction RadiationState)

```

Figure 7.18: ARL description of input/output variables

```

Rules Main (
  DetermineBlindAction1: if
    RadiationEast is completelyDark
  then
    BlindAction = "DOWN"
    RadiationState ="completelyDark"

  DetermineBlindAction5: if
    RadiationEast is sun
  then
    BlindAction = "DOWN"
    RadiationState ="sun"

  DetermineLightAction3: if DayLightIndoor is dark and Presence=="YES"
  then
    LightAction = "ON"

  DetermineLightAction4: if DayLightIndoor is middle and Presence=="YES"
  then
    LightAction = "ON"
)
)

```

Figure 7.19: ARL block of rules (excerpt)

Chapter 8

Agents

This chapter serves two purposes: to give a short introduction to the theory of multi agent systems and DAI; and to describe in detail every type of agent in use in AHA.

8.1 Overview

Following is a short summary of concepts in multi agent systems (MAS) and distributed artificial intelligence (DAI) on top of which AHA is built. It is not the intention to give a complete introduction to these topics here. The only purpose of giving these definitions here are to recall them so that we can later use them to reason why we chose to implement our system the way we did it. There are many excellent resources available for readers who want to read more about MAS and DAI before continuing: [WJ94], [Woo99] and [HS99].

8.1.1 Agent

First and most important is the concept of an agent. What exactly is a agent? There is no agreement on a formal definition of a agent yet but the following definition is generally accepted:

An *agent* is a computer system that is situated in some environment, and that is capable of *autonomous action* in this environment in order to meet its design objectives ([WJ94]).

A more informal definition of an agent would be that a agent is a piece of software that is continually reevaluating the input it gets from it's environment to determine the output it should send back (action) to the system. A agent is generally non-determinating which means that it doesn't end at any point of time.

An agent, as defined above, that is only reacting to its environment can not be called intelligent. What are the differences between an agent and an intelligent agent? An intelligent agent is an agent that has the following characteristics ([WJ94]):

- *reactivity*: intelligent agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their design objectives
- *pro-activeness*: intelligent agents exhibit goal-directed behavior by *taking the initiative* in order to satisfy their design objectives
- *social ability*: intelligent agents are capable of interacting with other agents (and possibly humans) in order to satisfy their design objectives.

An agent can by no means be restricted to be some part of software as an agent can also be a person or a

piece of hardware (embedded agent). This is especially the case for intelligent buildings where there are all three sorts of agents present: persons, software agents and hardware agents.

8.1.2 Agent architectures

The architecture of an agent describes the internals of an agent (type of algorithms and data structures an agent uses). There are four different classes of architectures for agents mentioned in the literature ([Woo99]):

- *logic based agents*: decision making is realized through logical deduction.
- *reactive agents*: decision making is realized in some form of direct mapping from situation to action.
- *belief-desire-intention agents*: decision making depends upon the manipulation of data structures representing the beliefs, desires, and intentions of the agent.
- *layered architectures*: decision making is realized via various software layers, each of which is more-or-less explicitly reasoning about the environment at different levels of abstraction.

The design of AHA is based on a mixture of the above architectures (see the following sections for details about every agent). Some of its agents are reactive because they react (in near real-time) to input from the environment. At the same time they are also logic based because they make decisions by inference making and they are belief-desire-intention agents because they are striving to achieve multiple goals (desires). This clearly shows that real-world applications of agents can't be associated to one of the many definitions given in theory – they are a mixture of multiple of these definitions.

8.1.3 Multi Agent Systems

A multi agent system is a system where multiple agents are running (acting) in the same environment. Running in the same environment implies that they get the same inputs (at least partially) and that their actions affect the same environment. This automatically leads to more complex issues – as soon as there are multiple agents in the same environment the action of one agent affects all the other agents in the system as well (non-episodic). This clearly shows the need for some sort of communication between these agents. There are two basic modes of multi agent systems: *cooperation* and *competition*. In a cooperative MAS agents cooperate to reach a common goal. Agents in such a system might for example each have a very specialized function so that they can only together achieve a goal that requires several of these specialized functions. In a competitive MAS, on the other hand, agents compete with each other for limited resources like computation time. For a more detailed survey of MAS refer to [HS99].

AHA is designed as a cooperative MAS. Agents in AHA are specialized agents that are each pursuing their own goals. To reach these goals (design objectives) these agents must cooperate with other agents in the system because they can't achieve their goal without cooperation.

8.1.4 Agents for intelligent buildings

The environment of a IB is, before assumptions and simplifications, inaccessible, non-deterministic, non-episodic ([Oza]), dynamic and continuous which is the most complex environment of all. It is inaccessible because the agent can't observe the complete state (there aren't sensors for everything), non-deterministic because an agent pursuing an action can't know for sure what effect it will have on the building, non-episodic because different goals interact and influence each other, dynamic because there are many other processes in the system (building) that can't be controlled (e.g. humans) and it is continuous because events can happen at every point of time.

8.2 History Agent

Several reasons lead us to the decision that data about the activity in the building, it's environment and the systems activity should be logged.

These reasons were :

- requirement to make statistical tests about the behaviour of the system
- have a learning base, where the system could learn from experience
- ability to plot charts about the behavior of the system

Of course this data had to be stored in a database. MySQL's performance, pricing, features and it's previous application in the AHA Project were the reasons why we chose that DBMS.

The frequency at which such data should be sampled is always a delicate decision to take. Too frequent sampling requires a lot of performance and disk space for the sampling process. It also slows down all later data mining activities. If one goes for a low sampling frequency important peaks could be missed. We decided for a sampling frequency of 2 per Minute. Still we made this sampling frequency configurable in the configuration facility.

Special care was taken to be economical with the CPU resources. Imagine if a whole building with hundreds of rooms was controlled and observed by this system. The database would have to store 1000s of changes per minute. To accomplish this we are doing caching of variables and deferred logging with prepared statements to the DBMS.

HistoryAgent only stores values of bus properties which are actually monitored by some agents in the system. The bus system of an ordinary building has hundreds if not thousands of different variables, so the database would be jammed after a short time. To avoid this, the BusAgent stores all pending variable updates until HistoryAgent fetches them. See figure 8.2 for the relationship between HistoryAgent and other Agents.

We tried to design a simple database scheme (Figure 8.1) which is easy to handle for the HistoryAgent (because active logging is time critical). We did not worry about the effort needed for later processing of the data because the prevalent task is to log to the database.

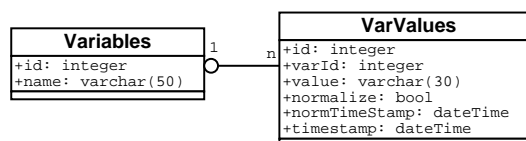


Figure 8.1: Database Schema for HistoryAgent

Note that the storage type for the field *value* in the table *varValues* is `varchar(30)`. HistoryAgent is directly storing the value returned from BusAgent which sometimes returns values like `SET_ON 0.0 0.00`. Also note that *varValues* has two timestamp fields. This is because HistoryAgent is normalizing the time stamp for some entries. The field *normalize* denotes entries which are normalized. For entries which are not normalized both timestamps are the same.

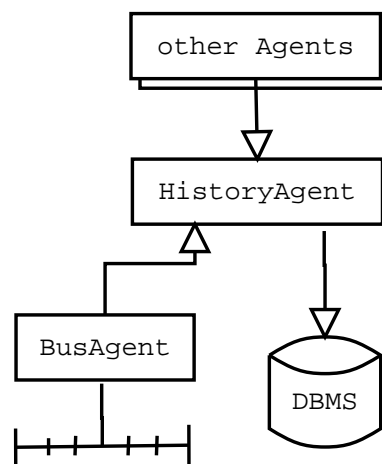


Figure 8.2: Agent relationship for HistoryAgent

8.3 Control Agent

8.3.1 Responsibility

The control agent is the representative of the concept *room* of the system. Every instance of this agent covers one specific room of the IB. It is the responsibility of the control agent to analyze the incoming sensory signals from all sensors belonging to the room and to make inferences from this information. It is also responsible for converting these decisions into actions. The control agent is running once for every room in a building that is controlled by the system. One instance of the control agent is automatically created for every room discovered by the boss agent. After startup a instance of the control agent is allocated to one particular room for its whole lifetime.

8.3.2 Design and Implementation

The decision making process as described in detail in chapter 7 is implemented in the control agent. Decisions are taken by a fuzzy logic control (FLC) as shown in Figure 8.3. The decisions are revised as new data becomes available that could possibly influence the outcome of the FLC.

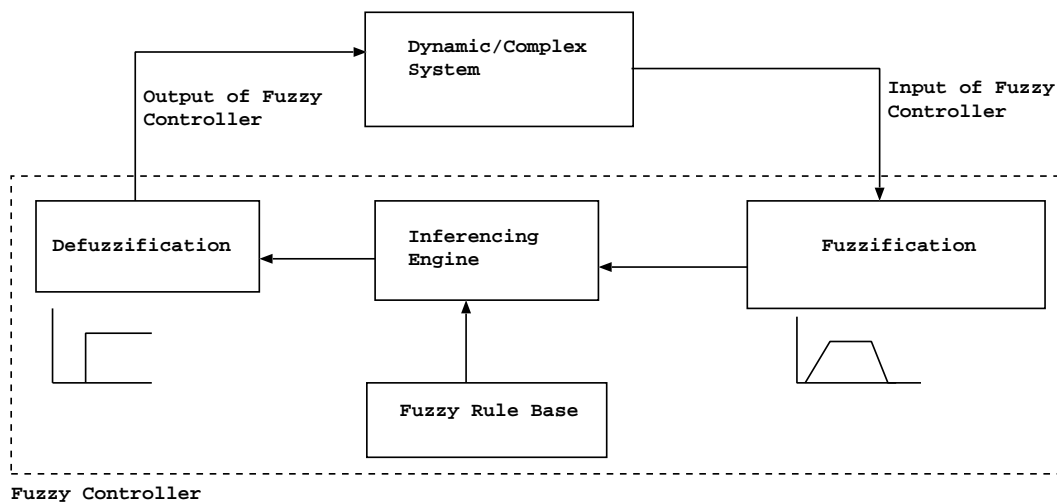


Figure 8.3: Overview of the decision making process, modeled as a fuzzy controller

Data that is used by the FLC is firstly preprocessed so that only prepared data is being evaluated by the FLC.

Important classes in the control agent (Figure 8.4) are Room and Presence. Messages between a Room and it's connected control Agent instance are exchanged with an intra-agent communication mechanism which is based on the same mechanisms than the inter-agent communication. Room is registering itself as a listener for all properties that it discovers to be related to the room it covers. It furthermore preprocesses all incoming updates for these properties and only sends a message to the control agent if there is a need to revise a decision because new relevant data has become available.

Presence detector handling

Presence (Figure 8.4) is the class that is abstracting a presence detector. There can be multiple presence detectors belonging to one single room but typically there is just one (in the middle of the room). Presence is sending out a message whenever there is a change of status which triggers an automatic update of the inferencing. The output of a presence detector is very simple – yes or no. But deciding whether someone is present or not is a very complex process. The presence detector delivers a non-continuous signal which

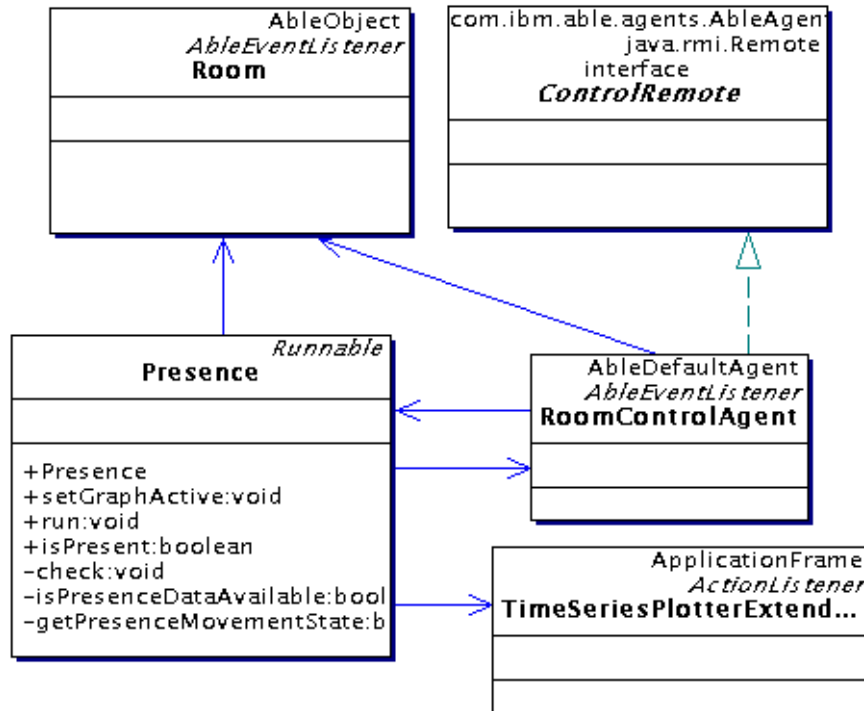


Figure 8.4: UML diagram of important classes in the control agent

peaks whenever there is a certain amount of movement in the room. But no movement in a room of course doesn't mean that there is no one present. The consequence of this behavior is that there has to be a smoothing process (for reducing the frequency of the signal) that constantly reevaluates the input of the presence detector in combination with past values and time to decide whether there is someone present or not. Figure 8.5 shows how the raw input and the smoothed input look like. Presence itself is running concurrently inside of the control agent (in its own thread that continuously reevaluates all necessary values).

Decision execution

Taken decisions are distributed to other parts of the system with the help of a distribution pipeline as described in section 7.1. Additionally to the internal processing entities of the control agent, all external agents that registered itself as being interested in new decisions taken get notified when there is a new decision available.

Internal decision execution involves additional pre-processing before a taken decision is actually executed. This is to prevent the system from changing an output signal too often (and thus start to oscillate). To achieve this, output signals are continually reevaluated in context of past values (history), new decisions taken and global system constraints (like maximal update frequency). If the update frequency starts to get too high the execution of the decision is delayed and queued for later. In case the decision becomes obsolete before it gets executed (e.g. if it stays for too long in the queue waiting to get executed), the decision is removed from the queue before getting executed.

8.3.3 Intra agent collaboration

The following messages are sent out by the display agent:

- Decisions

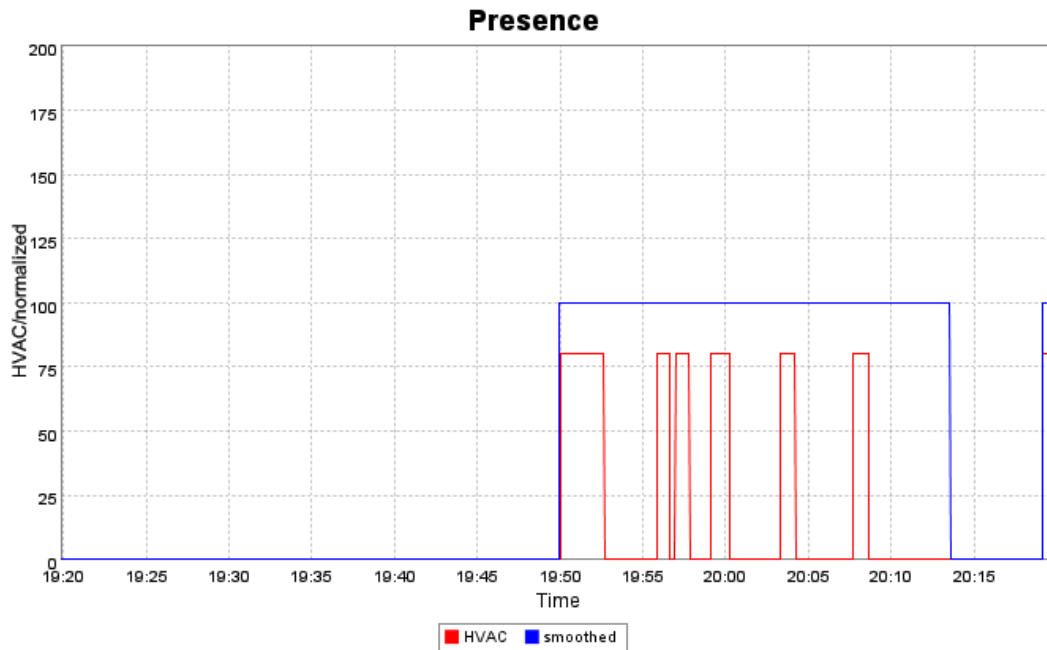


Figure 8.5: Smoothened and raw input data from presence detector

The control agent registers itself as interested in messages of the following types:

- Property updates (for all properties belonging to the room for which this instance of the control agent is responsible)
- Global property updates (for all global properties belonging to the building in which the currently selected room is physically located)

8.4 Bus Agent

The bus agent is the lowest level agent (Layered architecture, section 4.3.1) in the system. It acts as a gateway between the MAS and the actual field bus system.

8.4.1 Responsibility

The bus agent provides a generic way of accessing the control bus of a building. Bus agent provides this by instantiating a concrete implementation of the bus abstraction layer, which is described in section 6. It reads from the config file which implementation to instantiate. The bus agent acts as a proxy between the other agents and the bus abstraction implementation itself. It maintains a list of interested agents for every network variable available. All of these agents get a notification from the bus agent when there is a new value available for a network variable.

The bus agent furthermore provides the capability to explore the network variables available in the building. For this it provides ways to search for specific variables with search patterns expressed as regular expressions.

HistoryAgent relies on BusAgent providing it with updates for all subscribed variables. It does this by locally storing all conducted updates until HistoryAgent fetches them. In case of absence of a HistoryAgent or if HistoryAgent only rarely fetches these updates, BusAgent would be in danger of jamming it's memory with these queued updates. As a preventive measure this queue is implemented as a ringbuffer of a configurable size.

8.4.2 Intra-agent collaboration

BusAgent ist the most basic agent in AHA. Almost every other agent uses BusAgent. Especially HistoryAgent relies heavily on the services that the BusAgents provides. Other agents can access the building controll network over RMI with the methods defined in the BusRemote interface, which is similar to the BusController from the bus abstraction sub system. See [AJD] for details. BusAgent is not listening to any messages from other agents.

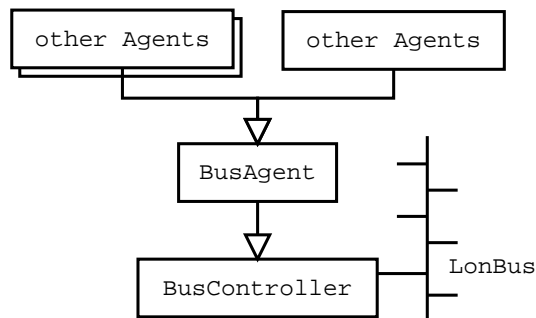


Figure 8.6: Agent relationship for BusAgent

The following messages are sent out by the bus agent:

- property update, is sent to listeners for a certain network property

8.5 Boss Agent

This agent makes the system capable of automatically reacting to changes in the system (dynamic configuration). It provides the functionality that the system automatically instantiates agents as new rooms are added (connected) to the network.

8.5.1 Responsibility

BossAgent is the "Boss" of all the other ControlAgents. It does connect to the BusAgent which has to be started before. BossAgent then creates a ControlAgent for every room it discovers in the building network. Discovering rooms in the network is done by matching property names with a pattern provided in the configuration facility (see section 5.5).

Note, all these ControlAgents are set to be inactive, which means that although they are observing their environment (the room) they are not taking any actions. This was done to not accidentally control rooms which we should not control. Still we are able to gather data about usage of these rooms.

8.5.2 Intra agent collaboration

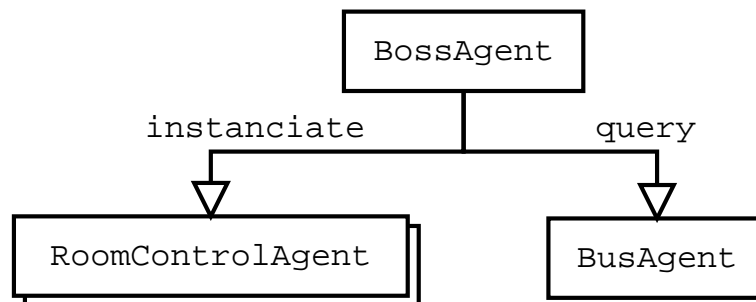


Figure 8.7: BossAgent & related Agents

BossAgent queries the BusAgent for all available network variables and tries to extract room names out of the variable names. It starts (instantiates) the required number of agents for every of the rooms discovered while searching the network (illustrated in Figure 8.7).

8.6 Display Agent

8.6.1 Responsibility

The display agent is the only agent in the system with a graphical user interface. It is designed to be used by expert users of the system to monitor the system or to interact with it manually. The display agent is focused on the room perspective – it always displays the room dependent variables that belong to the currently selected room. It additionally displays global variables that are available once per building (various weather variables like radiation at different places, humidity, temperature).

This agent doesn't necessarily have to run – the system is functional without it. It can be started an arbitrary number of times, even for the same room. It registers itself as listener for decisions for the currently selected room and additionally as listener for property value updates for all properties available for the selected room.

The following actors of a room can be controlled manually by the display agent: blinds (up, down, stop, angle) and light. Much more functionality of these actors can be controlled from the display agent than is possible by interacting with the devices directly via the physical devices of the room (eg light is only on/off and blinds only up/down whereas there are many states between these extremes). It furthermore displays the following data: outside temperature, outside radiation (from all sides of the building), outside humidity, inside illumination/light, status of the blinds and status of the lights.

8.6.2 Intra-agent collaboration

The following messages are sent by the display agent:

- Assign new value to property (to BusAgent)

The display agent registers itself as interested in messages of the following types:

- Decisions (sent out by the ControlAgent)
- Property updates (for all properties belonging to the currently selected room)
- Global property updates (for all global properties belonging to the building in which the currently selected room is physically located)

Chapter 9

Experiments and Results

9.1 Experiments

Since the history agent logs all the important data to the database we are able to do some analysis of the sensory input signals and the behaviour of the control agent. These analysis allowed us to improve the fuzzy rules for the control agent.

History agent logs it's data in a format optimised for insertion into the database. This prevents direct analysis of the data. It first needs to be preprocessed in some way. We chose to use Matlab for the whole data processing and plotting. The reasons why Matlab was chosen are :

- allows direct database access
- almost an industry standard
- widely used at INI
- license available at INI
- many toolboxes available which might be used in future work on AHA

As mentioned before, the data needs to be preprocessed before serious analysis and plotting can be done. The data in the database is not sampled at discrete time intervals. Thus if data of some sensors has to be analysed they first have to be exploded, such that there is the same quantity of sample data for each sensor. This explosion doesn't need a huge effort, since the data in the database is already normalized. The preprocessing thus only has to insert missing data. These missing data are not interpolated because the analysis should be done on the data an agent actually registered.

Because efficient insertion of data was the focus for HistoryAgent some sensors have string values stored in the database. These values have to be converted to numeric values during the data preprocessing.

For the data processing we have written two matlab functions. *connectAha* creates a DB connection for Matlab, see [MLF]. *processVariable* fetches the the sample data and does the preprocessing see [MLF].

Figures 9.1, 9.2 and 9.3 show some plots of sensors for the exterior environment of the building. These figures alone gave us a good impression on the behaviour of these input variables. They show for example that illumination has a much smoother curve than temperature but is much more erratic. On the ascending side, temperature follows illumination very close with a small lag. But on the descending side temperature falls much slower. Although these findings didn't have a direct impact on the chosen fuzzy rules they helped a lot for the understanding of their impact on other sensors.

Figures 9.4, 9.5, 9.6 show the intensity of Sunlight for the three directions east, south and west. They are correlated to the values of inside daylight (Figure 9.8).

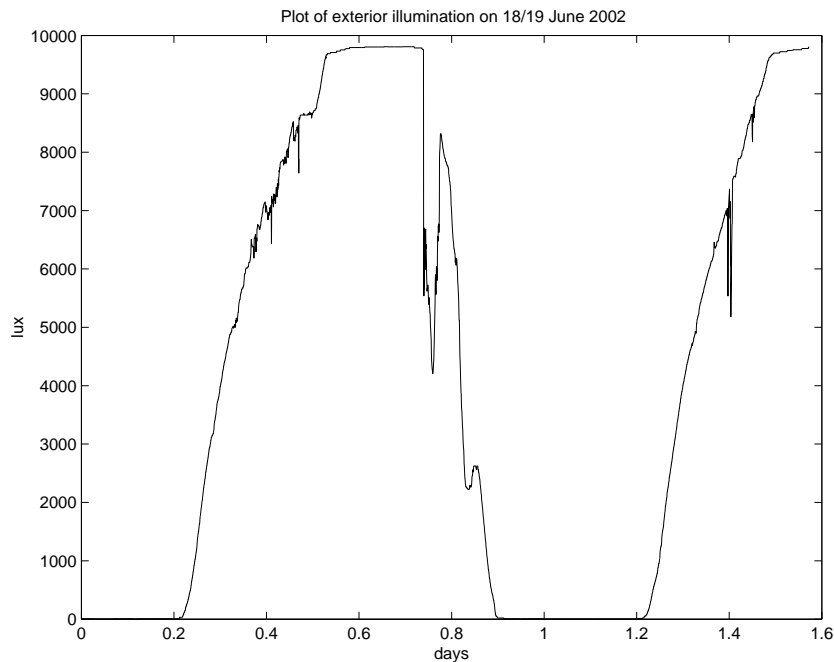


Figure 9.1: Plot of Exterior illumination 18/19 June 2002

Figures 9.7, 9.8, 9.9 are all plots of sensors inside the laboratory which was used for our experiments. This rooms windows are oriented toward north. This can clearly be seen in Figure 9.8. It shows a peek of the daylight sensor in the morning (ca. 0.3) and later in the the afternoon (ca. 0.53). This is because, for a short moment, the sun shines directly through the window. Between these two peaks, the window is in the shadow of the building.

Figure 9.9 shows how AHA controls the blends. The Fuzzy Rules are designed to protect the room from beeing heated up by direct sun light. During the peak in the morning as well as during the longer period of direct sunlight in the afternoon, the blends are brought down.

Figure 9.7 shows when the control agent switches the neon light on and off. On June 18th, during the first 3 periods when presence detector is positive the light is not switched on. This is because of DayLight sensor reporting enough lightness in the room. On the other side the next two periods when presence detector is positive the light is switched on. A presence detector is a very sensitive sensor. As soon as a persons starts making small movements it turns negative. Control agent compensates this by only switching off the light after a 5 minute period of presence detector reporting negative. Our plot does not compensate for that, this can be seen by the plot filling the area under the presence line. See Figure 8.5 for a plot that shows measured input signals from a presence detector together with the result of the smoothening process. This is due to the line oscillating between max and min.

9.2 Implementation Issues

Following are the most important problems we discovered during implementation of the system and the corresponding solutions found are given.

9.2.1 LNS

Unfortunately the documentation of [LNS] is not very accurate. The only documentation available is the API Javadoc. Only a handful of methods are actually documented, but even worse many features are present in

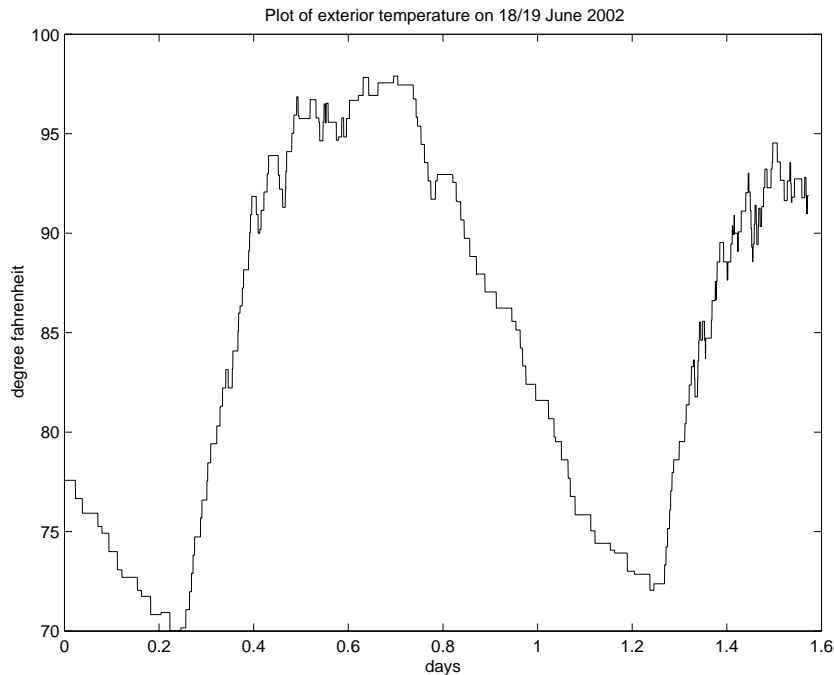


Figure 9.2: Plot of Exterior Temperatur 18/19 June 2002

the javadoc documentation but are not implemented in the actual implementation. One example which we stumbled upon is the *VariableChangeListener*. One can subscribe for these events, but they are never fired. This is because firing of these events does presently not work. *LNSRawUpdate* was used instead.

9.2.2 Presence Detectors

It turned out that the output of the presence detector, as described in detail in section 8.3.2, is not suitable as a direct input to the decision making process. The presence detector sends a number of spikes to the system according to the movement in the room. From this spikes the actual value (presence yes/no, lots of movement, little movement) needs to be computed by the control agent itself before it is forward to the decision making process. Figure 8.5 shows an example – the fast changing line is the raw input from the sensor, the line that seldomly changes is the output after preprocessing.

9.2.3 Pre-processing of outputs

There are certain specific states of the system in which it tends to start to oscillate between to different states of one output variable. This happens when the system is changing a output value that is directly forwarded to a actor like a light switch. If the action of this actor ensuingly affects the inputs to the system (in the case of the light this could be the dayLight input for example) the system may reach a state where it decides to turn off the light again. This starts a never ending cycle of fastly changing input/output values of the system.

To prevent this, additional processing takes places before a decision taken by the system is actually forwarded to the actors (execution). This process constantly re-evaluates past output values, update frequency and the special characteristics of every variable to make sure that the system can't reach a state where it starts to oscillate. This method to prevent the system of becoming unstable has proven to be very successfull and non-computational intensive at the same time.

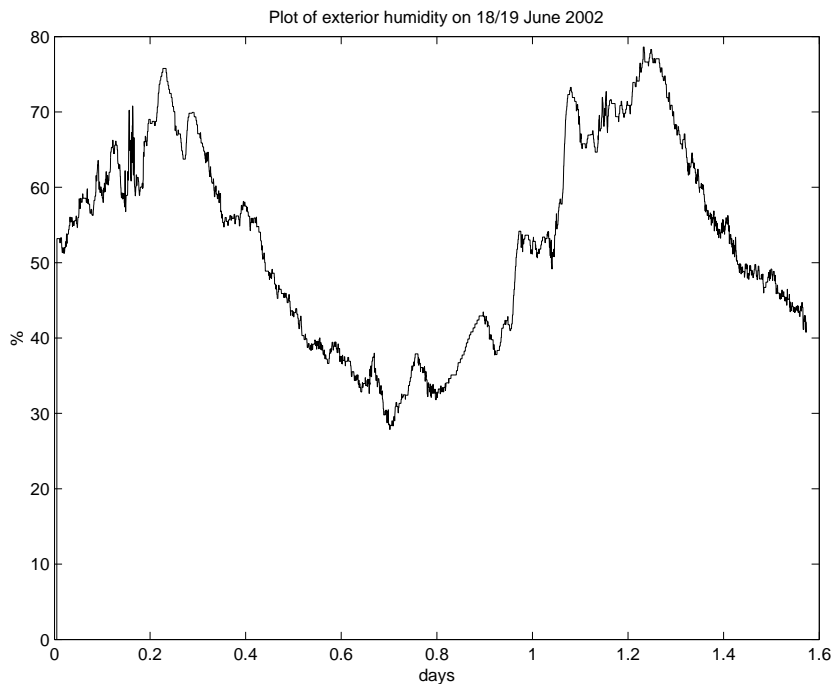


Figure 9.3: Plot of Exterior Humidity 18/19 June 2002

9.2.4 Intra-agent communication

Exchanging messages itself between agents didn't provide particular difficulties. What proved to be much more problematic is the content of these messages. All agents in a system need to agree on the content of every message with which they ever have to deal. These definitions, particularly during development, tend to be highly volatile. With fixed-coded definitions every change in the message syntax or semantics itself requires a complete re-deployment of all agents.

In the future considerations need to be made how this situation can be improved because it is clearly not satisfying. One possibility would be to use one of the standardised agent communication protocols like KQML or KIF which can be used to specify the syntax and semantics of data exchanged between agents.

9.2.5 Concurrency

This is a java specific issue: It proved to be rather tricky to deal with iterators in a highly multi-threaded environment where different iterators which are used in different threads point to the same collection. The problem is that a iterator becomes invalid as soon as the underlying collection gets modified in any way. As soon as this is the case usage of a iterator pointing to it will result in a *ConcurrentModification* exception. Carefull handling, encapsulation and synchronisation makes it possible to handle this situations.

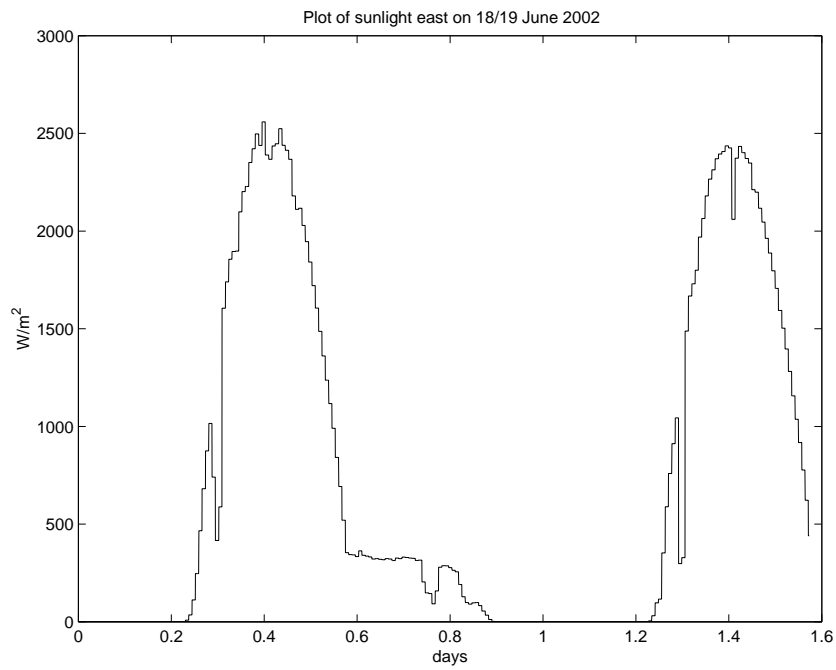


Figure 9.4: Plot of Exterior Sunlight East 18/19 June 2002

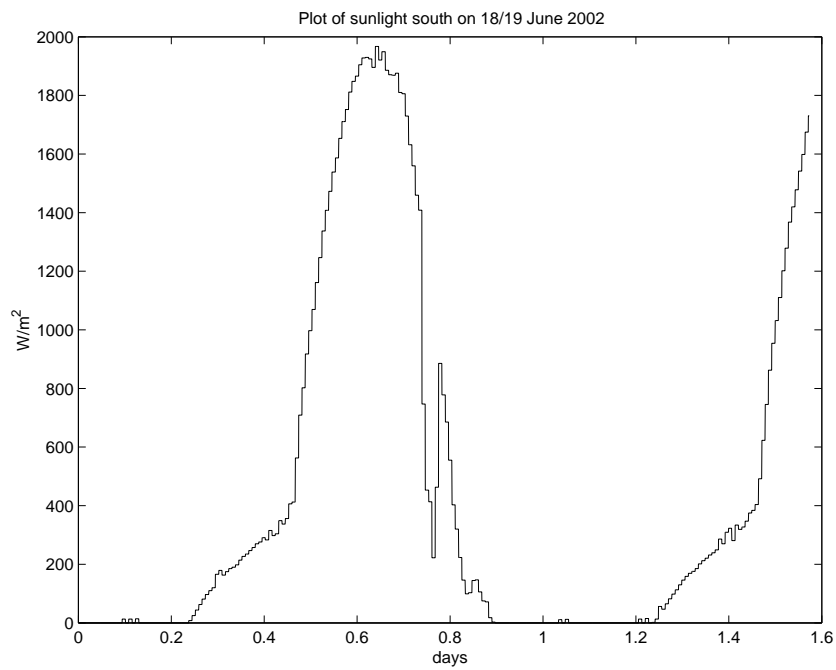


Figure 9.5: Plot of Exterior Sunlight South 18/19 June 2002

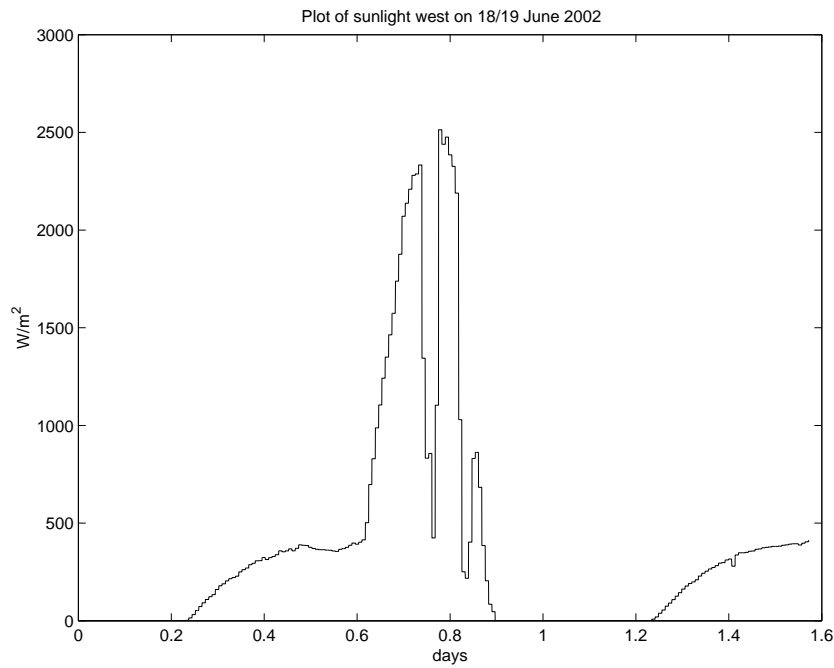


Figure 9.6: Plot of Exterior Sunlight West 18/19 June 2002

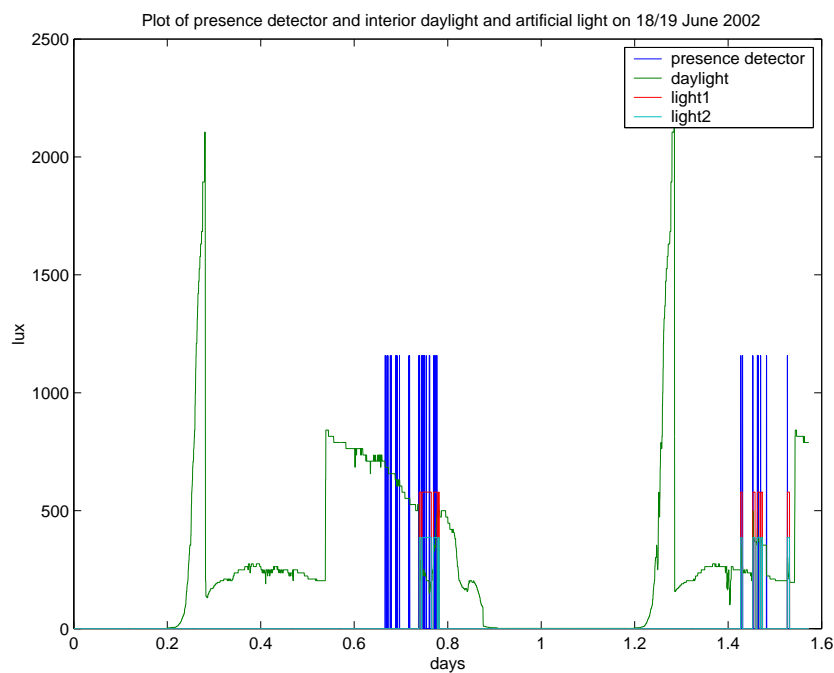


Figure 9.7: Plot of interior Presence with DayLight and Artificial Light for 18/19 June 2002

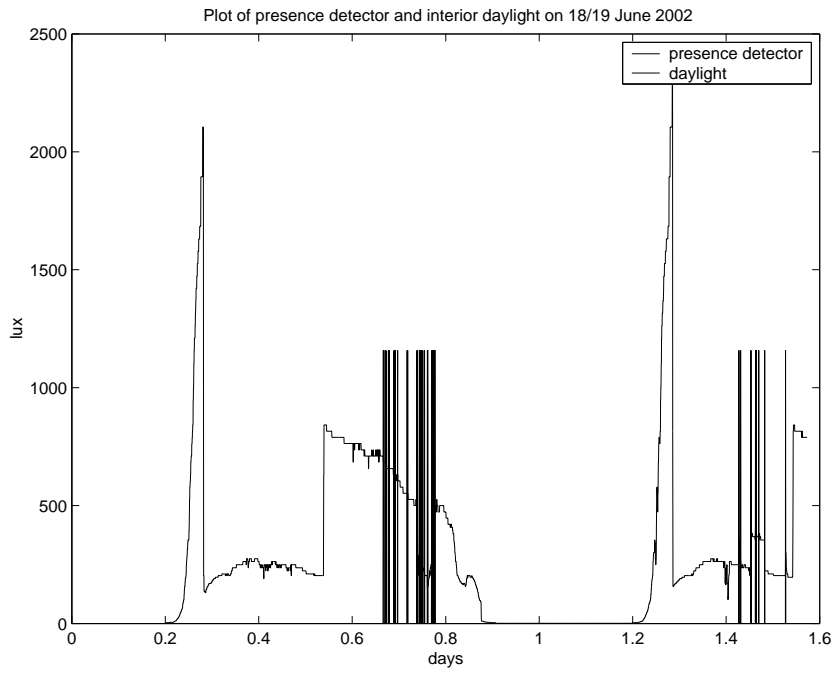


Figure 9.8: Plot of interior Presence with DayLight for 18/19 June 2002

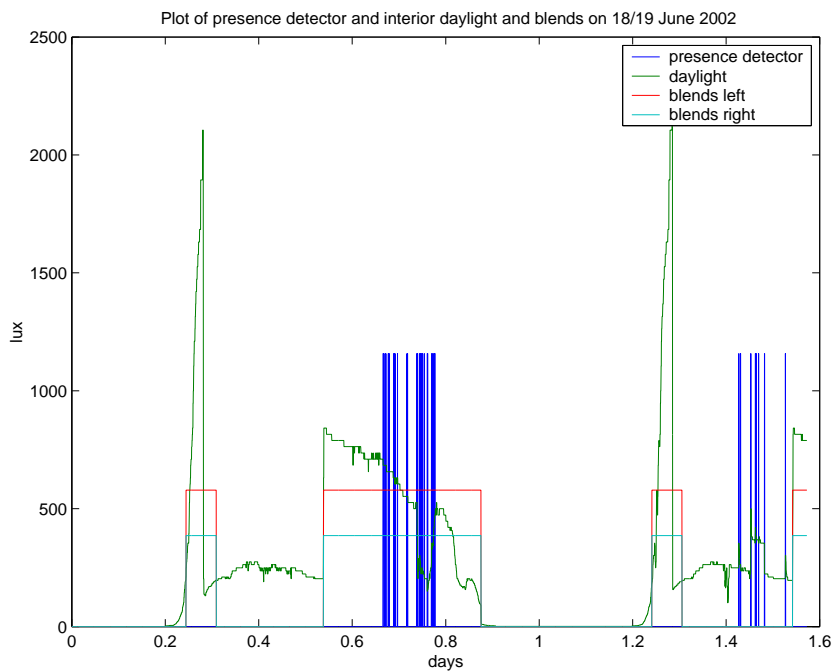


Figure 9.9: Plot of interior Presence with DayLight and Blends status for 18/19 June 2002

Chapter 10

Conclusions and Future work

The final goal for the first part of the AHA project was to establish a stable base for further experiments and enhancements of the system. We established a basic system architecture, agents that provide basic services like logging, bus abstraction and dynamic configuration and an agent structure that takes decisions based on a fixed fuzzy logic rule base. We conducted several long-term experiments which offered a tremendous amount of new knowledge about what works, what doesn't work and what needs to be improved (see chapter 9).

We plan to enhance AHA in the next step of the project (which will be our diploma thesis) such that AHA will be adaptive to its environment. AHA should react to individual feedback of users which communicate with the system by interacting with the sensors of the building. AHA should learn the individual preferences of the inhabitants of the building such that, in the long run, no manual interactions are necessary.

To achieve this AHA needs to be enhanced such that it is learning how users react to decisions taken by the system. AHA needs to adapt its internal database of rules of behavior to continually adopt itself to changing conditions. An intelligent building is different from conventional learning system where there is an optimal solution to the problem provided. There is no such thing as "the optimal solution" for the control of an intelligent building. This is because such a building is a moving target. Preferences of inhabitants, the inhabitants itself, the structure of the building and the distribution of the rooms are things that continually change over time.

AHA needs to be a continually learning system to constantly adapt to these changing conditions. We plan to incorporate several machine learning algorithms to achieve this. This makes it possible to compare the performance of such algorithms in a real-world environment like a intelligent building. It is planned to particularly focus on reinforcement learning algorithms ([RS00]). Other options that could be pursued would be distributed learning in a multi agent environment. This is particularly attractive because it is unlikely that the task at hand can be solved satisfactorily with a single, centralized, learning system that covers everything.

Other possibilities for achieving adaptive behavior would be the usage of evolutionary computing approaches like genetic algorithms or genetic programming. These systems are well suited for continuous adaption with no optimal state so pursuing this options further would give interesting insights of the performance of evolutionary methods in such a "moving target" environment.

An other interesting point where further research could be conducted is the integration of additional input/output devices into the system. A particularly interesting option is the integration of personalized wireless devices like PDA's or mobile phones. In the near term future almost every advanced mobile device will be bluetooth capable. A single bluetooth access point only covers a very small area where its signal is available which makes it possible to pin-point single individuals with great accuracy. This would allow it to further personalize the system in a way not imaginable before.

AHA's focus is on the building as a whole. A building for AHA consists of a number of rooms which contain

sensors and actors. The combination of AHA with a system that focuses on a single room like the intelligent space Ada ([EBB⁺02]) or the MIT intelligent meeting room project ([Bro97]) would provide interesting possibilities for further enhancements of the system.

Glossary

ABLE	Agent building and learning network
Agent	An <i>agent</i> is a computer system that is situated in some environment, and that is capable of <i>autonomous action</i> in this environment in order to meet its design objectives ([WJ94]).
AHA	Adaptive Home Automation
AI	Artificial intelligence
ARL	Agent rule language
DAI	Distributed artificial intelligence
Defuzzification	The process of converting a fuzzy set to a crisp number (FLC output)
DGC	Distributed garbage collection
EIB	European installation bus
ETH	Swiss Federal Institute of Technology
FLC	Fuzzy logic controller
Fuzzyfication	The process of converting a crisp number to a fuzzy set
IB	Intelligent Building
IE	Intelligent Environment
INI	Institute for Neuroinformatics, Switzerland
LNS	Lon Network Server
LonWorks	Field bus network protocol / standard
MAS	Multi agent system
MIMO	Multi-input-multi-output (classification for a FLC)
MISO	Multi-input-single-output (classification for a FLC)
ML	Machine learning
SISO	single-input-single-output (classification for a FLC)
UNIZH	University of zurich

Bibliography

- [ABL] Agent building and learning environment (able). <http://www.alphaworks.ibm.com/tech/able>.
- [AJD] Aha javadoc documentation. <http://www.easc.ch/aha/javadoc/index.html>.
- [ANT] Antlr, another tool for language recognition. <http://www.antlr.org>.
- [BDY99] Magnus Boman, Paul Davidsson, and Håkan L. Younes. Artificial decision making under uncertainty in intelligent buildings. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 65–70, 1999. <ftp://ftp.dsv.su.se/users/mab/uai99.ps>.
- [Bro86] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, Vol 2(Nr. 1):pp14–23, 1986.
- [Bro97] R. Brooks. The intelligent room project. In *Proceedings of the second International Cognitive Technology Conference (ICT'97)*, Aizu, Japan, 1997.
- [COP] Can open protocol. <http://www.canopen.org/canopen/>.
- [EBB⁺02] Kynan Eng, Andreas Bähler, Ulysses Bernardet, Mark Blanchard, Adam Briska, Jörg Conradt, Marcio Costa, Tobi Delbrück, Rodney J Douglas, Klaus Hepp, David Klein, Jonatas Manzolli, Matti Mintz, Thomas Netter, Fabian Roth, Ueli Rutishauser, Klaus Wassermann, Adrian Whately, Aaron Wittmann, Reto Wyss, and Paul F M J Verschure. Ada: Constructing a synthetic organism. In *Proceedings of IROS 2002, IEEE/RSJ International Conference on Intelligent Robots and Systems, Lausanne, Switzerland*. Institute of Neuroinformatics ETH/University of Zurich, 2002.
- [EIB] European installation bus. <http://www.eiba.com>.
- [Ful00] Robert Fullr. Neural fuzzy systems. In *Advances in Soft Computing Series*. Springer-Verlag, Berlin/Heidelberg, 2000. ISBN : 3-7908-1256-0.
- [HS99] Michael N. Huhns and Larry M. Stephens. Multiagent systems and societies of agents. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 79–120. The MIT Press, Cambridge, MA, USA, 1999.
- [IBM] IBM alphaworks. *ABLE Team, Able Rule Language (ARL) Documentation*.
- [JCO] Jcommon, general java library with supporting classes for various purposes. <http://www.object-refinery.com/jcommon/index.html>.
- [JFR] Jfreechart, a library for producing dynamic plots out of java. <http://www.object-refinery.com/jfreechart/index.html>.
- [L4J] Log4j logging framework. <http://jakarta.apache.org/ant>.
- [LNS] Lns hmi developer's kit for the java platform. <http://www.echelon.com/Products/lns/lnsJava.htm>.
- [MLF] Aha matlab functions. <http://www.easc.ch/aha/matlab>.
- [Oza] Nikunj Oza. A survey of robot architectures.

- [RS00] Andrew G. Barto Richard S. Sutton. *Reinforcement Learning, An Introduction*. MIT Press, 2000. ISBN : 0-262-19398-1.
- [VC00] Graham Clarke Victor Callaghan. A soft-computing dai architecture for intelligent buildings. Technical report, Department of Computer Science, University of Essex and Department of Computer Science, University of Hull, 2000. <http://cswww.essex.ac.uk/intelligent-buildings/publications/springerverlag.pdf>.
- [Wei99] Gerhard Weiss, editor. *MULTIAGENT SYSTEMS, A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, Massachusetts, 1999. ISBN : 0-262-23203-0.
- [WJ94] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. <HTTP://www.doc.mmu.ac.uk/STAFF/mike/ker95/ker95-html.h> (Hypertext version of Knowledge Engineering Review paper), 1994.
- [Woo99] Michael Wooldridge. Intelligent agents. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 27–78. The MIT Press, Cambridge, MA, USA, 1999.