diploma thesis

# Adaptive Building Intelligence built on the Open Services Gateway initiative

Simon Gassmann
<sgassman@ini.phys.ethz.ch>

Patrick Brunner
<p1brunne@ini.phys.ethz.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 10
8640 Rapperswil, Switzerland
http://www.hsr.ch

Institute of Neuroinformatics
University and ETH Zurich
Winterthurstrasse 190
8057 Zurich, Switzerland
http://www.ini.unizh.ch

May 18, 2005

Typeset by LaTeX

# OSGi ABI

## Prof Dr Josef M Joller (supervisor)

This diploma thesis is a first step in a new ABI direction: the use of well defined services built on top of a framework (OSGi). Service oriented architectures proved to be very powerful in recent years and the IT world is still exploring the full potential of this new paradigm.

The delegation of tasks to dedicated services is probably a very natural concept, as can be seen in biological systems; but as it turns out, it also allows the developer to easily expand the system and include more functionality or new services and to loosely couple system components.

This diploma thesis opens a whole new 'universe' we will have to explore in forthcoming term projects and further research projects:

- voice driven devices
- mobile devices integration
- fancier graphical representations of the system state
- a more active communication between the user and the system
- and many more exciting things

Several results presented in this diploma thesis give some hints in which direction we will have to make the system more 'autonomic' - self-healing and fault tolerant and more realistic.

The short diploma thesis time frame did not allow to go into too many details. But I believe it points in the right direction. Some results seem to justify the decision to start ABI from scratch using new and quite different ideas than in the 'old' ABI.

I look forward to new exciting work and projects and hope both (now ex-) students will have some exciting projects in their future work life as well. I wish you all the best!

**Abstract**

Nowadays many modern buildings are equipped with a huge variety of sensors and actors. At the Institute of Neuroinformatics in Zurich a framework for sophisticated controlling of such devices was developed - the Adaptive Building Intelligence System. To improve further extension of the framework a redesign is needed. The main goal of this diploma thesis is implementing a new software, which uses the well known OSGi Framework. Stability, reliability, extensibility are keywords for this work. The OSGi Framework provides elaborate concepts that are fully used within the new system.

Additionally, we demonstrate a concept of replaying data using a database connection. This is useful whenever new learning algorithm should take over the control of a building. Since it allows deep testing rather than online tests which usually get annoying for inhabitants.

Extensibility is greatly ensured by the OSGi Framework. We introduce an abstract bus model enabling easy integration of new devices on the fly. Even plug and play mechanism are supported. Moreover, network availability is tracked by a general monitoring unit, increasing the overall reliability.

Finally, we present a new learning approach by using goal directed reinforcement learning. This fits perfectly for the ambient brightness task needed in adaptive building intelligence. Further concepts are shown in the very last section describing future elements.

# Table of Contents

# List of Figures

# List of Tables

# Part I

# Introduction, learning and controlling

# Chapter 1

# Introduction

## 1.1 Overview

The adaptive building intelligence system (ABI System) was developed during previous term works as well as diploma thesis. Modern buildings can actively support and assist their inhabitants. The ABI System was designed to learn dynamic environment changes. Different approaches for learning have been introduced. A major improvement in the learning task was done by the previous work [TZ04] where a long and short term memory were presented.

The main goal of the diploma thesis was to develop a new stable version of the existing ABI System. In our previous term work [BG04] we answered the question how the ABI System can be integrated in the OSGi Framework. We showed the advantages of using a well-known framework, which has the ability of adapting to dynamic changes. During the evaluation in the term work we also encountered several problems. Complexity of dynamic programming is a drawback and very time consuming. All the same it leads to a easy extendable system. The OSGi Framework turned out to be fairly complicated to understand. In this diploma thesis we built the ABI System from scratch, not using any components from the previous releases. Only the abstract bus concept developed in our term work was taken into consideration.
Unlike the term work, which compromises a proof of concept, we implemented here a fully working new ABI System based on the OSGi Framework [Ini03]. We also solved several major insufficiencies of the old ABI System. It lacked from occurring exceptions during run-time, which are believed coming from the underlying LNS Server. To avoid problems of this kind we investigated LONDevice specifications and kept strictly to given instructions and recommendations of [Lon].
Moreover a database connection was implemented as a first step to ensure having descent data used for data mining in the results section and for learning tasks. A concept of replaying situations from the data base is also a part of this diploma thesis, which could play a key role in future works. Since new learning algorithm can be easily tested on recorded data. The OSGi Framework is designed for dynamical changes of any service. Thus the implemented software is a starting point for any following projects.
Learning plays a major role in building intelligence. We therefore implemented a new learning approach in this diploma thesis. A reinforcement learning algorithm which was adapted to the fulfill building intelligence has been used. Another main task was implementing recovery mechanisms. To achieve that goal many new recovery concepts are introduced within this work.
Apart from hardware devices also the concept of virtual software devices is shown. This is particular important if collecting hardware events and upon them producing new output. Implemention of new device types which did not exists within the old ABI System is done to have more input for the learning task, i.e. virtual devices.

## 1.2 This document structure

This diploma thesis contains the following parts and main sections. We start with a short introduction and then give an overview of reinforcement learning. In the controlling section we present a fuzzy control circuit and explain the control unit used. Architecture, design and implementation sections describe the new built ABI System. Finally the results are summarized and discussed in the last section. Project setup, administrative commands, messenger commands, reinforcement implementation details and installations instructions are presented in the appendix section.

1. Introduction
2. Learning
3. Controlling
4. Architecture, design and implementation
5. Results
6. Appendix

## 1.3 Related work

In this section the key parts of all the related works is presented. It is by no means a complete comparison of the previous related term works and diploma thesis. In the following list we describe the different approaches.

- Ueli Rutishauser and Alain Schaefer → Learning without additional user input, online learning algorithm, learning a maximal structure, fuzzy rule base with anytime learning algorithm and no dynamic structure detection.
- Jonas Trindler and Raphael Zwiker → Short and long term memory, parallel architecture of the control unit and multi agent framework.

## 1.4 Acknowledgments

We would like to thank Raphael Zwiker for many crucial hints and illuminating discussions - Hanno Gassmann for reading the manuscript. We would also like to acknowledge Matthias Saenger for the weather information provided 7.1 and reading the manuscript. Finally we would like to thank Robert Brunner for giving helpful information about brightness calculation for the light devices 7.8.

# Chapter 2

# Learning

In this chapter we describe the reinforcement learning algorithm we used in our project. First a short overview is given and then we explain how we implemented our learning approach. Further information about reinforcement learning or machine learning is provided by [RSS98] and [Mit97].

## 2.1 Reinforcement learning

In this section we first give a short introduction to reinforcement learning. Subsequently the elements of reinforcement learning are described.

In general, reinforcement learning is a computational approach to understand goal-directed learning and decision-making. It uses states, actions and rewards to define the interaction between a learning service and its environment. A key feature of reinforcement learning are explicit goals. The learner is not told which actions to take, but must discover itself which action yield the most reward by trying them. Since an action can influence current situations as well as future situations it is important to take both situations into consideration while rewarding an action. This leads to two characteristics:

1. trial and error search
2. delayed reward, a reward that is given after an action has been executed

A reinforcement learning system consists of a policy, a reward function, a value function and optionally a model of the environment. These elements are now briefly introduced.

A policy defines the behaviour of a learning service at a given time. It basically maps from perceived states of the environment to actions. This means whenever a specific state is perceived an action follows (state - action pair). In simple cases the policy may be a lookup table. More complex cases might need extensive computation such as a search process.

A reward function defines the goal in reinforcement learning. It maps the previously explained policy (state - action pair) to a single number, the so called reward. An objective of the reinforcement learning service should be maximizing the total received rewards. In other terms the reward function defines the good and bad events for the learning service.

In contrast to the reward function which indicates somehow what is good in an immediate sense, a value function specifies what is good in the long run. The value function defines the total amount of reward a learning service can expect to accumulate over time when starting in a specific state.

The last mentioned element is the model. Models are used for planning. A model can for example predict the resultant next state and therefore also the next reward.

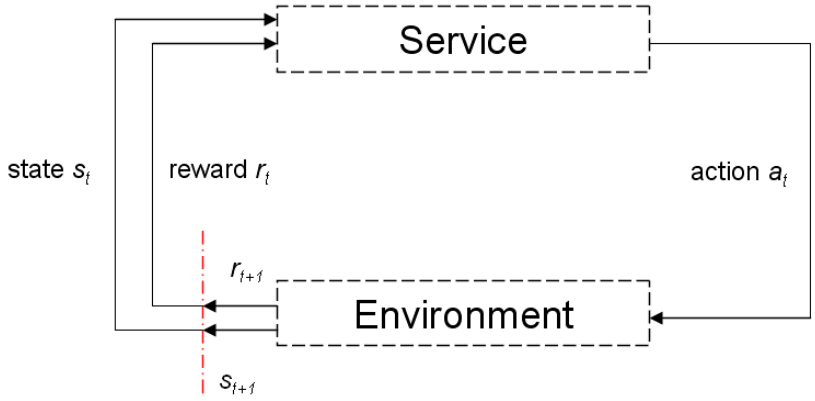


Figure 2.1: Service and environment interaction

As described in figure 2.1 the reinforcement service interacts continually with the environment by selecting actions and receiving rewards for that performed action. Those actions present then new situations (so called states) to the reinforcement learning services. The rewards received are special numerical values that the learning service tries to maximize over time. A complete specification of an environment such as learning service, states, actions, rewards and environment defines a task.

### 2.1.1 Reinforcement learning definitions

We give here the common definitions of reinforcement learning, as well as a description of the activity chain in figure 2.1.

Time steps are defined as:

$$t = 0, 1, 2, 3, ... \text{ in a continuing task: } t \to \infty \tag{2.1}$$

Environment states are defined as:

$$s_t \in S \text{ whereas } S \text{ is the set of possible states} \tag{2.2}$$

Actions are defined as:

$$a_t \in A(s_t) \text{ where } A(s_t) \text{ is the set of possible actions available in state } s_t \tag{2.3}$$

Numerical reward is defined as:

$$r_{t+1} \in \Re \tag{2.4}$$

Estimated action values are defined as:

$$V(s_{t+1}) = V(s_t) + \alpha [V(s_{t+1}) - V(s_t)] \text{ where } \alpha \text{ is a step sizeparameter} \tag{2.5}$$

Estimated action-values are defined as:

$$Q_t(a) = \frac{r_1 + r_2 + r_3 + ... + r_{k_a}}{k_a} \tag{2.6}$$

To specify more clearly here the $t$-th play of a action $a$ has been chosen $k_a$ times prior to $t$, rewards received are then $r_1, r_2, ..., r_{k_a}$.

At each time step $t$, the learning service takes an action $a_t$. As a consequence of its action, the learning service receives a numerical reward $r_{t+1} \in R$ and is then in a new state $s_{t+1}$. To choose an action out of all possible actions $A(s_t)$ a policy $\pi_t$ is used.

A policy is defined as:

$$\pi_t \text{ where } \pi_t(s,a) \text{ is the probability that } a_t = a \text{ if } s_t = s \tag{2.7}$$

A common policy used is the $\varepsilon$-policy. Most of the time this policy chooses an action that has a maximal action value. That means it takes the action with the highest reward so far calculated. With a probability of $\varepsilon$ it chooses an action at random. It follows that all nongreedy actions are given the minimal probability as follows:

$$\frac{\varepsilon}{|A(s)|} \tag{2.8}$$

Further all greedy actions are given the probability:

$$1 - \varepsilon + \frac{\varepsilon}{|A(s)|} \tag{2.9}$$

Such a policy ensures that actions are taken that have a high action value calculated so far (exploit) and from time to time with a probability of $\varepsilon$ it chooses a new action (explore). It balances the learning task between exploit and explore actions. This makes sense since it is important to see if there are actions that might also have reasonable rewards that we don't know so far.

### 2.1.2 R-learning for undiscounted continuing tasks

In cases the service-environment interaction does not break naturally into identifiable episodes, the learning is called a continuing task. In such cases it is not possible to calculate the return $R_t = r_{t+1} + r_{t+2} + r_{t+3} + ...$, as $t \to \infty$. A common approach is then using discounts on the return $R$ as follows:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.10}$$

R-learning is not using such a discount rate of the return value. One tries to obtain the maximum reward per time step. This time step must refer to fixed intervals of real time - they can refer to arbitrary successive stage of decisions making. The value functions for a policy $\pi$ are defined relative to the average expected reward (per time step):

$$p^\pi = \lim_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} E_\pi [r_t] \tag{2.11}$$

**Assumption:** the process is ergodic, meaning there exists an nonzero probability of reaching any state from any other. Thus $p^\pi$ does not depend on the starting state. The value of a state is defined as:

$$\tilde{V}^\pi(s) = \sum_{k=1}^{\infty} E_\pi [r_{t+k} - p^\pi | s_t = s] \tag{2.12}$$

The value of an state-action pair is defined as:

$$\tilde{Q}^\pi(s) = \sum_{k=1}^{\infty} E_\pi [r_{t+k} - p^\pi | s_t = s, a_t = a] \tag{2.13}$$

These values are called relative values since they are relative to the average reward under a chosen policy. R-learning is a Temporal-Difference learning (TD) approach, since it uses differences $V(s_{t+1}) - V(s_t)$ between estimates at two different times. In the R-learning algorithm this is done by calculating differences between $\tilde{Q}(s_{t+1}, a_{t+1}) - \tilde{Q}(s_t, a_t)$.

### 2.1.3 R-learning algorithm

The r-learning algorithm is defined as follows:

```
Initialze p and Q(s,a) for all s,a

Repeat forever:

1. s <- current state
2. Choose action a in s using behaviour policy (e-greedy)
3. Take action a, observe r, s'
4. Q(s,a) = Q(s,a) + a [r-p+max{Q(s',a'),a'} - Q(s,a)]
5. If Q(s,a) = max{Q(s,a),a} then:
      p = p + b[r-p+max{Q(s',a'),a'}-max{Q(s,a),a}]

whereas max{Q(..),a} means seeking for (s,a)-pairs that maximize the Q(..) value.
```

Parameters $a := \alpha$ and $b := \beta$ are step size parameters.These step size parameters have only an influence on the Q(s,a) values and how they are taken into consideration when a state change is performed in terms of $s_t \rightarrow s_{t+1}$ for a specific action $a_t$.

## 2.2 Integration of the r-learning algorithm into building intelligence

Depending on hardware devices many states can be possible. To cut down the amount of states and therefore the computation power used, we decided to use the following devices that define our states:

1. daylight, depends on light given by the environment (sun and electrical light).
2. presence, indicates the attendance of a person.
3. light, electrical light, can be turned on and off.

For further information see also 6. We give here just a general overview.

### 2.2.1 Adapted r-learning algorithm for this project

We first describe our approach by explaining what states we defined for the learning service. We defined states as follows:

$$s_i = [x, y, z] \tag{2.14}$$

$$x = \begin{cases} 1 & \text{daylight range 1} \\ 2 & \text{daylight range 2} \\ 3 & \text{daylight range 3} \end{cases}$$

$$y = \begin{cases} 0 & \text{somebody is present} \\ 1 & \text{nobody is present} \end{cases}$$

$$z = \begin{cases} 0 & \text{light is off} \\ 1 & \text{light is on} \end{cases}$$

To make things clear we give here an overview of all states in table 2.1. First we introduce the state transitions occurring within each row and each column, then the states overview is given, omitting the former introduced transitions. The figure 2.2 shows the row transitions, whereas figure 2.3 shows column transitions. A row is arranged in a way that only the daylight changes are possible, thus the transitions are restricted to wait for these daylight changes. Column transitions described in figure 2.3 though show changes that occur on the presence level. One must bear in mind that in every state there is also the implicit transition to itself possible, the wait self transition. Transitions for the lights are showed in figure 2.4 excluding the above mentioned daylight, presence and wait self transitions.

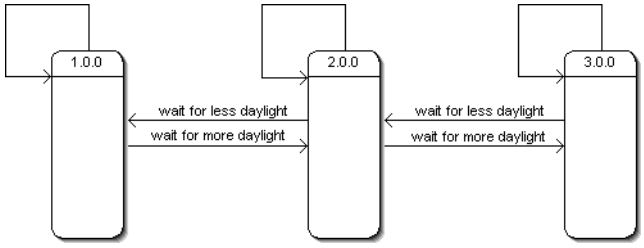


Figure 2.2: State transition within each row

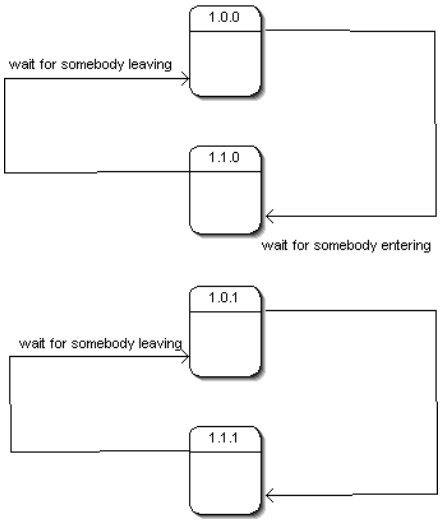


Figure 2.3: State transition within each column

Actions $a_t$ are defined as $a_t = (s_t, s_{t+1})$. The problem we encountered with the r-learning algorithm was that we have only a few actions we can really take such as, turning lights on or off. All other input comes directly from the environment i.e. daylight changes. That means we can not change the daylight value by taking an action. This leads us to change the algorithm and allow sudden state changes invoked by the environment to take place. In such cases we don't reward the action at all, except the wait actions we introduced. Wait actions are different from actions that have an impact on the environment. Generally they are used to learn also to wait for a state change invoked by the environment. For example if we wait for a daylight change $a_t = (s_t = (1, 0, 0), s_{t+1} = (2, 0, 0))$ in a certain state $s_t = (1, 0, 0)$ and the daylight changes to state $s_{t+1} = (2, 0, 0)$ invoked by the environment results in a reward given. In other situations where the action chosen does not match the state we suddenly encounter from the environment, we simply adjust our algorithm and bring it to the appropriate state. For example the current state is $s_t(1, 0, 0)$. Let us assume the action we choose to perform is $a_t = (s_t = (1, 0, 0), s_{t+1} = (1, 1, 0))$. Unfortunately the next state invoked by the environment is $s_t(2, 0, 0)$. In this situation we do not give rewards at all, since the action does not match. We believe this must work as the r-learning algorithm is not depending on its starting state.
To simplify the algorithm's complexity we also decided to have only a small amount of states and actions are not allowed being built on run time (for an overview of states and actions see table 2.1 and figure A.1). In addition states and actions have been chosen to give a simple but effective learning for our ambient room learning task.

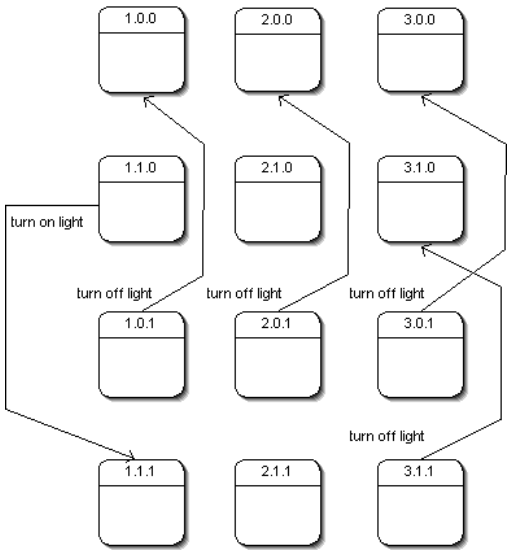


Figure 2.4: States and actions defined for the r-learning algorithm

| State | daylight | presence | light |
|---|---|---|---|
| $s_1 = [1, 0, 0]$ | daylight is in range 1 | nobody is present | the light is off |
| $s_2 = [2, 0, 0]$ | daylight is in range 2 | nobody is present | the light is off |
| $s_3 = [3, 0, 0]$ | daylight is in range 3 | nobody is present | the light is off |
| $s_4 = [1, 1, 0]$ | daylight is in range 1 | somebody is present | the light is off |
| $s_5 = [2, 1, 0]$ | daylight is in range 2 | somebody is present | the light is off |
| $s_6 = [3, 1, 0]$ | daylight is in range 3 | somebody is present | the light is off |
| $s_7 = [1, 0, 1]$ | daylight is in range 1 | nobody is present | the light is on |
| $s_8 = [2, 0, 1]$ | daylight is in range 2 | nobody is present | the light is on |
| $s_9 = [3, 0, 1]$ | daylight is in range 3 | nobody is present | the light is on |
| $s_{10} = [1, 1, 1]$ | daylight is in range 1 | somebody is present | the light is on |
| $s_{11} = [2, 1, 1]$ | daylight is in range 2 | somebody is present | the light is on |
| $s_{12} = [3, 1, 1]$ | daylight is in range 3 | somebody is present | the light is on |

Table 2.1: States used for the r-learning algorithm

### 2.2.2 Testing the adapted r-learning algorithm for this project

We first developed a test environment to ensure our slightly adapted r-learning algorithm works as expected and fulfills the given requirements. We have chosen a random starting state $s_{t=random(1-12)} = s_1$ and then we chose one of the possible actions in this state randomly $a(s_1, s_{t=random(1-12)=s_2*})$, whereas this action was only rewarded if the following state $s_{t=random(1-12)} = s_2$ was the one predicted previously $s_2 = s_2*$. This was repeated for 500'000 times. It is obvious that matching a state predicted by an action is not very likely thus out of these 500'000 trials only 41'672 rewards were given. All $Q(s, a)$ have been initialized with zero.

The following example helps understanding the test set up.

1. choose a starting state randomly $\rightarrow s_{start} = (1, 0, 0)$
2. actions possible in state $s_{start}$ are:
   - $a_{waitself}(100, 100)$
   - $a_{waitdaylight}(100, 200)$
   - $a_{waitdaylight}(100, 300)$
   - $a_{presencechangewait}(100, 110)$
3. choose randomly an action $\rightarrow a_{waitdaylight}(100, 200)$.
4. repeat forever (in this example only one cycle is showed):
   - choose a next state randomly $\rightarrow s_{next} = (2, 0, 0)$
   - if action predicted this as the following state $\rightarrow$ yes $\rightarrow$ do reward
   - else if check was not successful do not reward

Step size parameters were set to the following values:

- step size parameter $\alpha$=0.1
- step size parameter $\beta$=0.0
- $\varepsilon$-greedy = 0.1

The reward function which denotes reward to matched actions is described in the appendix A.2 and A.3. We chose a reward function that gives always the same reward for a matched action. Some of the results are shown in following figures 2.6, 2.6, 2.7, 2.8. An important fact is that all q-values of a certain state-action pair converge to a real number. Even though some actions matched more than others, depending on the reward function and the $\varepsilon$-greedy choosing algorithm, it is shown that all q-values converge. As mentioned earlier (2.1.2) this is exactly what we expected since the r-learning algorithm is not depending on its starting state. Assume that $\pi$ is the estimation policy, then the action-value $Q$ is an approximation of $\tilde{Q}^{\pi}$. It follows from the definition that the average reward $p$ is an approximation of $p^{\pi}$. In this test run we calculated an average reward of $p = 5.875000000000019$.

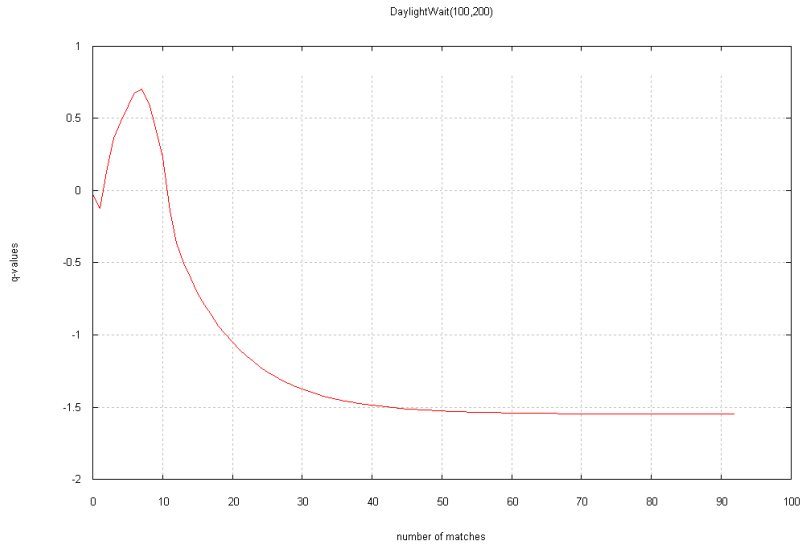


Figure 2.5: Action: daylight wait from state (1,0,0) to (2,0,0). Action $a_t = (s_t = (1,0,0), s_{t+1} = (2,0,0))$ converges with increasing number of matches ($numberofmatches >= 50$).

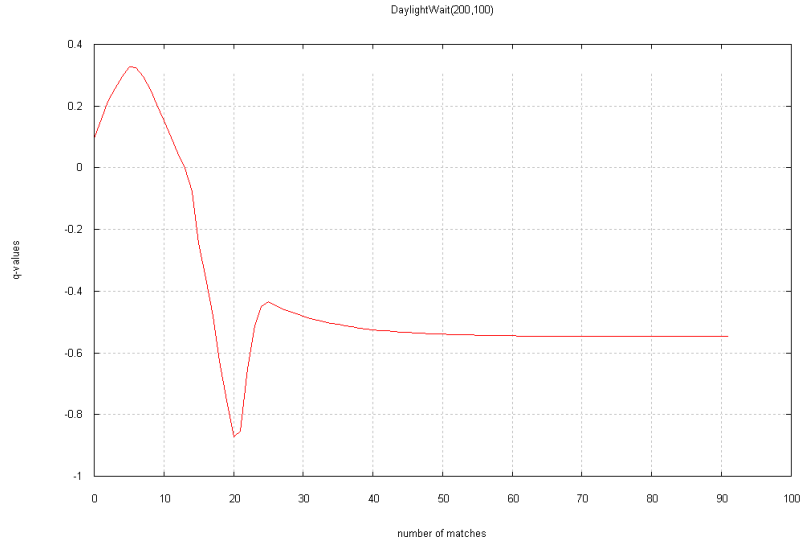


Figure 2.6: Action: daylight wait from state (2,0,0) to (1,0,0). Action $a_t = (s_t = (2,0,0), s_{t+1} = (1,0,0))$ converges with increasing number of matches ($numberofmatches >= 50$).

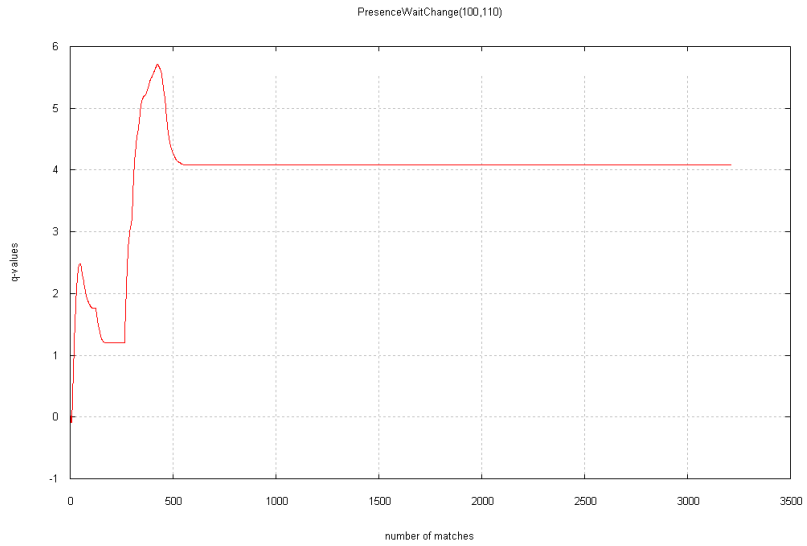


Figure 2.7: Action: presence change wait from state (1,0,0) to (1,1,0). Action $a_t = (s_t = (1,0,0), s_{t+1} = (1,1,0))$ converges with increasing number of matches ($numberofmatches >= 600$).

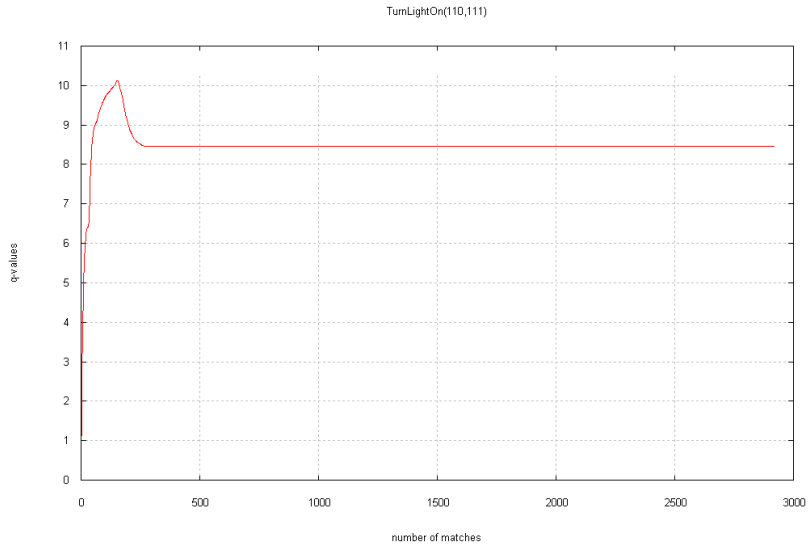


Figure 2.8: Action: turn light on from state (1,1,0) to (1,1,1). Action $a_t = (s_t = (1,1,0), s_{t+1} = (1,1,1))$ converges with increasing number of matches ($numberofmatches >= 300$).

### 2.2.3 Punishment for user interactions integrated in the r-learning algorithm

The problem to solve is answering the question how punishment can be integrated into the r-learning algorithm. A simple but effective approach is to adapt the static reward function to a dynamic reward function. So far we had for any action $a_t$ a constant reward value. For example an action $a_{turnlighton}(110, 111)$ is chosen. Then the light in a certain room (area) is turned on, but the user is not satisfied with this decision and turns the light off immediately. The learning system can detect such user interaction and then adapt its reward for the given action to a smaller value - even negative rewards would be thinkable. To do so a delayed reward has to be implemented, because the reward can only be given correctly if we wait for timeout and then do the reward. Another approach is to change the calculated $Q(s, a)$ value i.e. to a smaller one. This is a more direct way than the first described idea, since it has a more direct influence on the future action chosen.

### 2.2.4 Long and short term memory using the r-learning algorithm

It makes sense having a short and a long term memory in adaptive building intelligence (see related work [TZ04]). Assume that all $Q(s, a)$ values are stored in a table over a certain time period , e.g. Monday morning. The short term memory could be simulated by saying, all $Q(s, a)$ values experienced and calculated so far represent the short term memory. If the time period changes, e.g. to Monday afternoon, a new short term memory period is evoked. The $Q(s, a)$ calculated for the period Monday morning is then merged into the long term memory, which could be also a table of $Q(s, a)$ values. Merging can be done by calculating an average value for each entry in the table. Decision can then be done by taking into consideration both, the short and long term memory.

# Chapter 3

# Controlling

In this section we describe the general controlling mechanism used. Our first approach was using a fuzzy control circuit, which turned out as an inadequate solution see 3.2. The second approach is more sophisticated and works fine, see 3.1 for more details.

## 3.1 Area control of the daylight

The area room service is responsible for recording the current state given by its connected devices. It then sends the perceived state to the area decision controller unit, which in turn asks each connected sub controller units for an action proposal. Based on the received actions it must decide what action to choose and pass this action back to the area room service. Ambient controller or energy controller uses reinforcement learning, i.e. also short and long term memories are thinkable. This way of controlling is strongly goal directed.

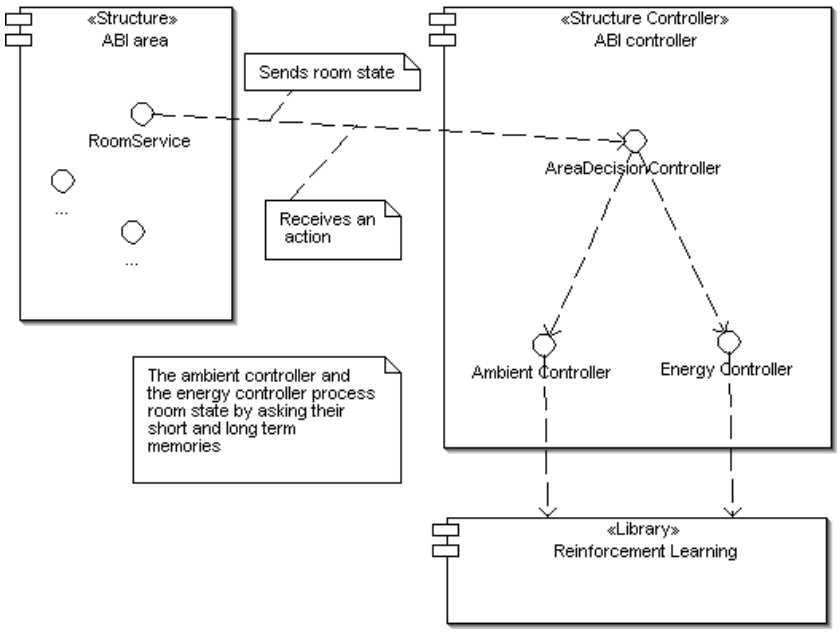


Figure 3.1: Area controlling

## 3.2 Fuzzy control of the daylight

Our first approach was to build a control circuit to achieve a certain luminance in an predefined area. Controlled devices include lights and blinds. We used a fuzzy library for our control circuit. For further information about fuzzy and the used fuzzy library see [Tra02] and [Fuz]. Figure 3.2 shows the control circuit used.

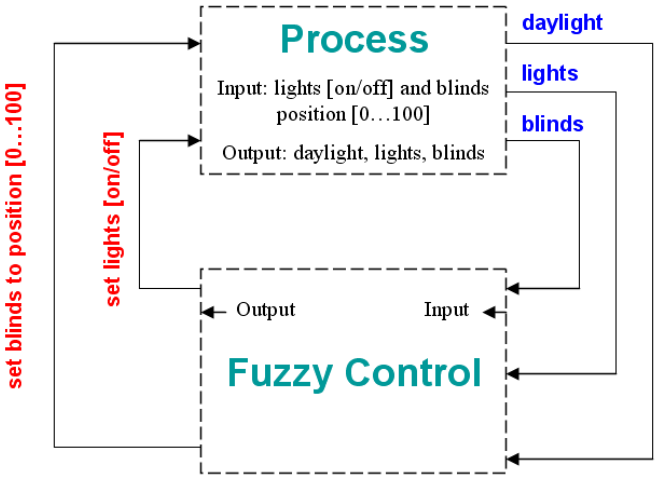


Figure 3.2: Fuzzy control circuit

The fuzzy control unit is feed with light, blind and daylight input from the process (environment). Based on the input values it produces then output for the blind and light devices. Generally this worked fine controlling the daylight in a room, the problem we were faced is that the blinds can get out of sync and return false input for the fuzzy control unit. In such cases the system crashes, i.e. results in unexpected behaviour. Hardware blind issues are discussed further in section 7.3. We decided not to use the fuzzy control circuit for our learning task, since it turned out to be not stable in terms of the wrong behaviour of the blind devices.

# Part II

# Architecture, Design and Implementation

# Chapter 4

# Architecture

This chapter defines the system architecture and its goals. Furthermore we introduce the chosen software components. Whenever possible we were searching and integrating high quality components implementing a standard, and released as open source.

We start with an overview of the system components and how they play together. Later in the chapter we refine the descriptions of the different system components. For a better understanding of the ABI Systems architecture we introduce the OSGi Frameworks concept beforehand. Issues, problems and their solution are discussed in the implementation chapter 6 and the design chapter 5 respectively.

## 4.1 System components overview

The whole considered system, is a heterogeneous distributed system of hardware and software components. From now on we use the term ABI System for the components designed and implemented by ourself, and also for the open source components[1] adapted for our needs. The overall goal of the here presented architecture is to provide an open, flexible, and service oriented architecture. We made an effort for a solid base using standards[2] and applying standard procedures[3]. Most important, the architecture must meet the Adaptive Building Intelligence task requirements. Roughly speaking, these are:

**Bus abstraction:** to interconnect different field buses, and virtual buses feeding information into the ABI System.

**Device abstraction:** separate the real device from its provided information input, and also to allow virtual devices without hardware representation.

**Collecting data:** collect all the raw information by sensing the building and its environment.

**Learn from data:** extract features, learn patterns from the collected data. Moreover classify a new situation, resulting in a decision for an action to take.

First of all there is a hardware network of sensors and effectors in a building. These are interconnected with a hardware field bus. Typically such an installation is delivered with a software to configure the network structure. The problems of such a configuration are stated in the chapter 1. The ABI System starts where the static rule configuration ends.

From the above stated it is clear, that there are software components abstracting the hardware in such a way that different field bus technologies and different kind of devices can be handled by the ABI System.

[1] Namely the Wireadmin Service Bundle, and the Fuzzy Library.
[2] Telnet Protocol, SQL, XMPP, XML, URI, LDAP filter string RFC1960
[3] procedures as described in the [Ini03], knopflerfish programming documentation, software design and patterns

The abstraction is one part, on the other hand, each bus hardware technology requires a specific software component to make it accessible from the ABI System. Besides of the hardware abstraction there is also the need for a structural abstraction. This structure component should be flexible enough to handle both, static[4] and dynamic[5] structures. Or at least it should allow the coexistence of dynamic and static structures. Typically, such components have an administrative[6] side besides the functionality they provide. Thus the demand for an administrative component. As with all distributed heterogeneous software system, it is very important to have a central information or log facility. In fact in a system trying to do machine learning it is of most importance to have a possibility to gather, save, and retrieve information in a structured way. From this follows the claim of an information storage component.

The so far enumerated components do not bring in the adaptive building intelligence part, but they are the needed infrastructure for the machine learning and controlling component.

Our container for the outlined ABI Systems components is the OSGi Framework, a java application container. The key features of the OSGi Framework architecture are introduced in the section 4.2.

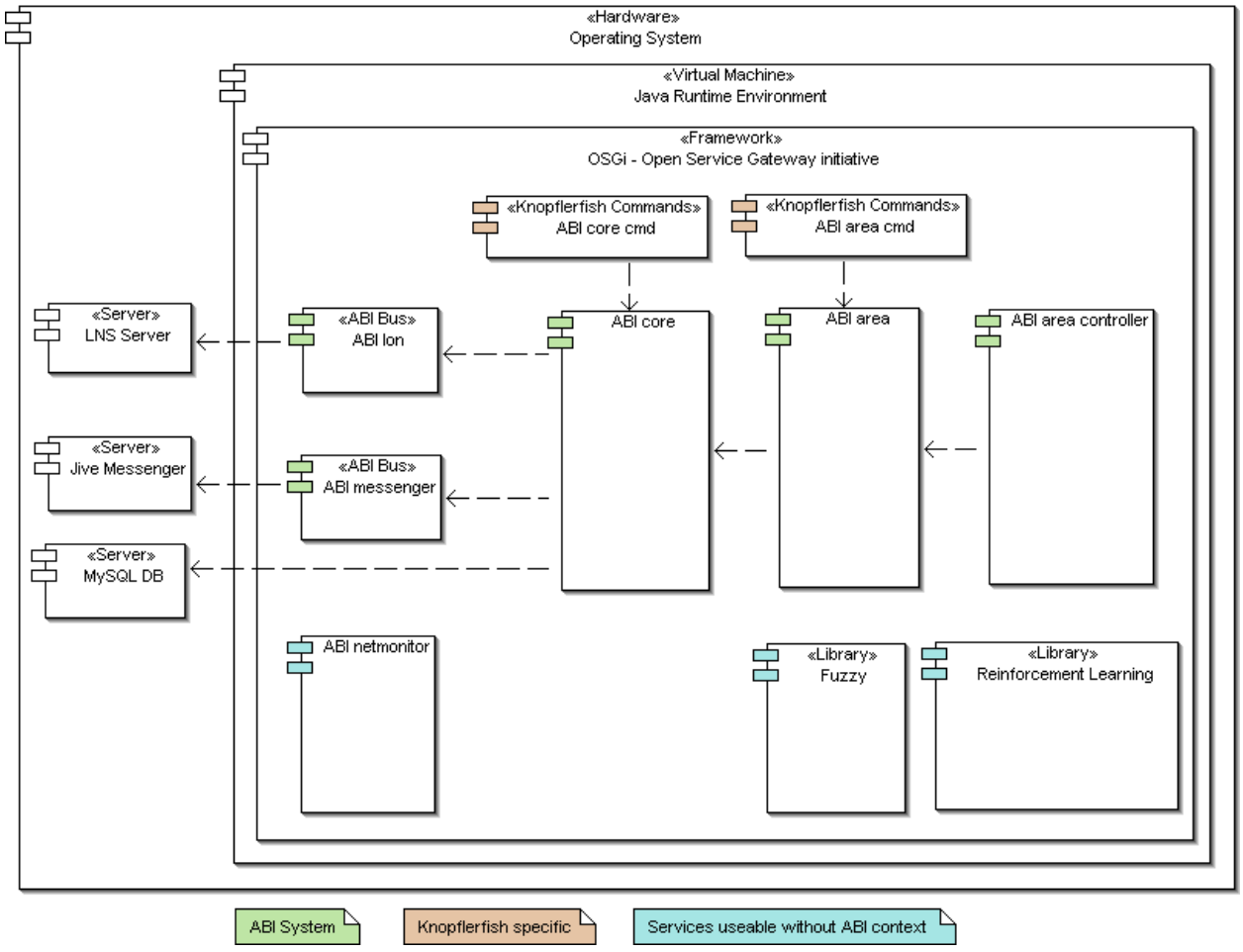


Figure 4.1: System components overview

The figure 4.1 serves as a high level overview of the previous described components, whereas the components have the names according to the ABI System. The following list matches the components to the names. Bundle is the term for component used in the OSGi Framework environment.

**bus and device abstraction** The `ABIcore Bundle` provides an abstract bus and device abstraction for each sensor and effector.

[4] Dependencies like building - floor - room
[5] Discovering related sensors and effectors without prior knowledge of a static structure.
[6] Define configurations, showing an internal status, startup, shutdown

**bus technology component** The `ABIlon Bundle` is a specific bus technology implementation for connecting the ABI System to the LonWorks field bus. It is also responsible for the Lon specific implementations of the sensor and effector devices. The `ABImessenger Bundle` is the bridge for the ABI System to the instant messenger server – a Jive Messenger server in our case. Actually it implements a presence sensor.

**structural abstraction** The `ABIarea Bundle` represents the static building - floor - room structure, and finds devices in a plug and play fashion. Furthermore it coordinates the learning and controlling aspect.

**administrative component** Instead of having one monolithic administrative component, it is divided into the `ABIcoreCmd Bundle` and the `ABIareaCmd Bundle`. Each of these bundles is a command-line providing administrative bundle for its corresponding service bundle.

**central log component** The central log component is placed in the `ABIcore Bundle` as it is seen as core functionality.

**information storage** We decided to use the MySQL database as information storage. But this is not mandatory, as the logged information is written to a file if a database is not available. Although it is highly recommended to use a database.

**learning component** The `ABIlearning Bundle` makes different machine learning approaches accessible to the ABI System.

**controller component** The `ABIcontrolling Bundle` offers controlling mechanisms to the ABI System.

## 4.2 The OSGi Framework

This section introduces the OSGi Framework as far it is needed to understand the resulting ABI System architecture. It explains the OSGi Framework terms bundle, service, service reference, and service registration properties. The section ends with the OSGi Framework concept of filters.

OSGi stands for Open Service Gateway initiative.

The OSGi Framework is a lightweight framework providing the infrastructure for hot plugging[7] software components, while the framework is running. It is designed to be stable robust and long running. The interested reader can find more details about the framework in our term work [BG04]. Here we focus on the key features for a better understanding of the whole architecture.

A software component in the OSGi Framework is understood as a bundle providing one or more services. Other software components use the available services to combine and refine them to a new service. An application is viewed as a, more or less, loose coupled collection of services.

The OSGi Framework provides the infrastructure to install, uninstall, update, start and stop bundles. And the bundles use the OSGi Framework to register and deregister their services, and also to look up and track other bundle's services. One has to bear in mind, that services can appear, disappear and change during runtime. Each bundle must take care to handle this dynamic aspect correctly.

One can see the OSGi Framework as a registry of services, providing the infrastructure which

- fires events when a bundle is installed, deinstalled, updated, started stopped
- fires events when a service is registered, deregistered, changed.
- helps to look up services with the help of a filter facility
- helps to track services

[7] plug in and out software components

As the OSGi Framework is a specification only, there exists different implementations - commercial ones on one hand, but also open source ones on the other hand. Besides the core framework, the specification describes in detail useful bundles in the context of gateways[8].Our term work [BG04] points out the details why the OSGi Framework is a good choice for the Adaptive Building Intelligence application, and which described bundles to take as base pillars.

Technically spoken, application units like the `ABIcore Bundle`, are deployed as bundles. A bundle is a JAR-file[9] containing a MANIFEST.MF file[10]. The manifest describes the contents of the jar file Within the OSGi Framework a bundle has a life cycle as illustrated with the figure 4.2. When a bundle is started it is

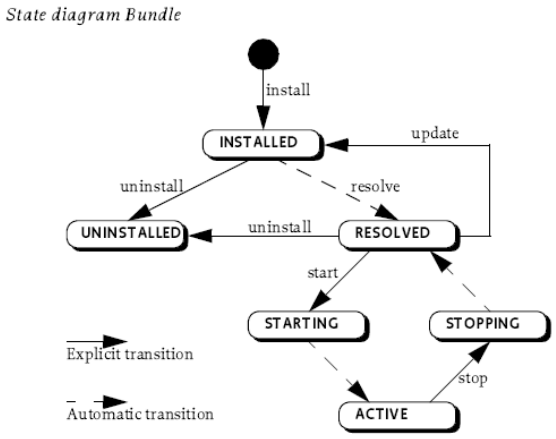


Figure 4.2: bundle states

initialized with a bundle context object which serves as interface from the bundle to the OSGi Framework. It allows the bundle to access methods from the framework. And the OSGi Framework uses the bundle context for book keeping various things concerning the started bundle.

A closer look on how the OSGi Framework service concept works, is as follows. It is recommended to define an interface describing the services. This interface is exported to the OSGi Framework and imported by other bundles. The bundle providing the service, creates one or more instances of classes implementing this interface. It makes this service implementation instances available by registering them within the OSGi Framework under the interface's fully qualified class name. Other bundles query the OSGi Framework with a filter interface provided by the framework. Applying a filter to the frameworks registry does not return the requested services, instead it returns the descriptive portion of the service - the service reference. A service reference holds a dictionary describing the requested service. If the description satisfies the needs of the requesting bundle, it can obtain the service through the bundle context and the service reference. Potentially a service reference can exist without an available service, this case must always be handled appropriate by the requesting bundle. It is up to service registering bundle to provide a useful set of properties in the service registration.

The service descriptions, or more formally the service registration properties, are mission critical. These descriptions act as the glue between services, and bundles respectively. Viewing an application as a collection of related services, the registration properties are responsible for relating the services. Thus these properties are part of the applications architectural design, or the applications complexity. The service registration properties are stored as key - value tuples, whereas key is a string constant and value is any object, or an array of objects.

Last but not least the OSGi Framework overview is completed with the important filter feature. As the OSGi Framework is mainly a registry for services, along with their service registration properties, it is of utmost importance to have an easy way to filter for services. The filter syntax string is based on the RFC 1960. This RFC defines a representation of LDAP search strings with the following grammar:

[8] the primary target was the gateway in the broadband connected home.
[9] Java archive file, a zip compressed form of a file structure
[10] it is a configuration file holding key,value pairs.

```
filter ::= ´(´ filter-comp ´)´
filter-comp ::= and | or | not | item
and ::= ´&´ filter-list
or ::= ´|´ filter-list
not ::= ´!´ filter
filter-list ::= filter | filter filter-list
item ::= simple | present | substring
simple ::= attr filter-type value
filter-type ::= equal | approx | greater | less
equal ::= ´=´
approx ::= ´~=´
greater ::= ´>=´
less ::= ´<=´
present ::= attr ´=*´
substring ::= attr ´=´ initial any final
inital ::= () | value
any ::= ´*´ star-value
star-value ::= () | value ´*´ star-value
final ::= () | value
```

The `attr` is a key in service registration properties and the `value` the extracted value object. If the value object is of type array of objects, or vector, the filter is applied recursively. Also numeric and string objects are handled according to their types. More details can be found in [Ini03].

## 4.3 The ABI System

This section covers our ABI Systems architecture within the OSGi Framework. Choosing the OSGi Framework as application framework implied breaking down the Adaptive Building Intelligence application into bundles and services. The section cuts also in the two main aspects learning and controlling (4.3.2) and infrastructure (4.3).

### 4.3.1 Infrastructure

The infrastructure part has two major responsibilities:

- abstraction of real buses and devices
- collecting information

The figure 4.3 shows the part of the ABI System providing the infrastructure for the Adaptive Building Intelligence application.

On one hand the abstraction is achieved by specifying an `ABI bus service` which describes how a bus is accessed in the ABI way. On the other hand there are abstractions for the most common devices used in the building automation. For each technology bus we want to connect with the ABI System, we have to provide a bundle with the specific bus access technology. In most building automation cases the devices are tightly coupled with the used bus. Hence the implementation for a bus will also carry the implementations for the devices. But the architecture does not prescribe to have the bus and device implementations in one bundle. One can argue having the bus and its devices in one bundle helps structuring the software, but we see it as an unnecessary restriction. As it is too specific to field buses. If we extend the ABI System beyond the field bus we can take office buses into account. Office bus examples are the Ethernet, or the USB. And it is clear that there are far more possible devices for these office buses, than we can think of now.

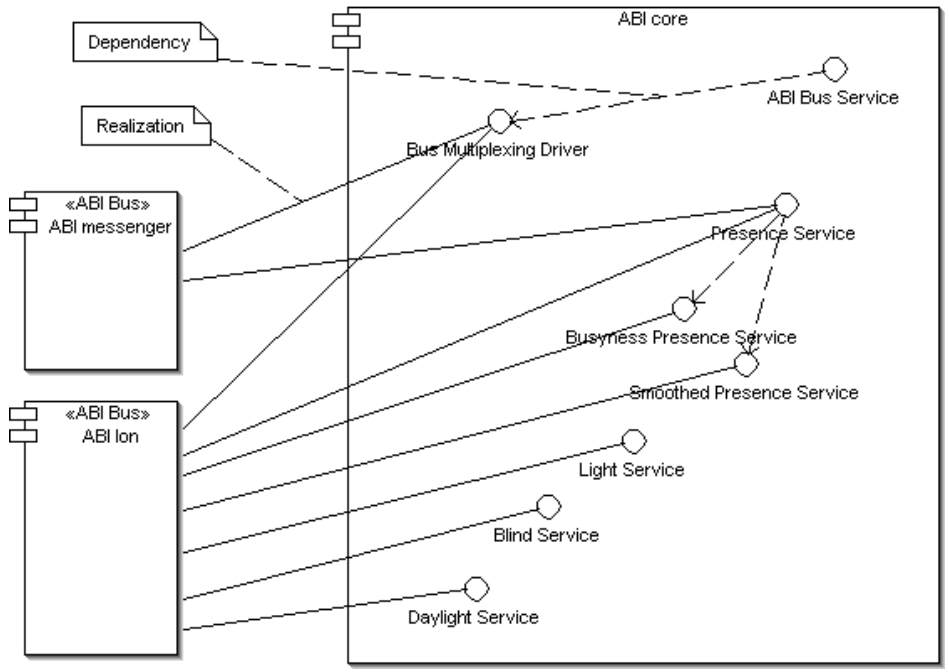


Figure 4.3: Overview of the ABI base

In principle we see the bus as the administrative component to the network of devices. As a consequence the `ABI bus service` is the administrative bus to all connected buses, thus it is multiplexing requests to the ABI infrastructure. We are using the OSGi Framework infrastructure of devices and drivers. In our system we have the `ABI Bus Multiplexing Service` acting as a driver. The different specific bus technologies are devices and get collected by the `ABI Bus Multiplexing Service`.

If we look from above to the ABI infrastructure we see only a cloud of device services ready to be used, but we are not aware that they are connected to different bus technologies. To choose now the right device services from this cloud we need additional information. This information is stored in the different service registration properties. For example table 4.1 shows the service registration properties of a busyness device service.

Although the devices are abstract, the information they provide is not. It is very important that the collected information is structured, and stored in the most raw form possible. To meet this requirements we use the OSGi Framework log service facility in combination with a database. Which information is written to the database is decided by each service. Logging is not restricted in any way. A structure of the log entries must be defined application wide as a convention.

### 4.3.2 Learning and controlling

The learning and controlling part uses the infrastructure part as the source of information. In building automation we want to learn and control a specific area. This area service or structure abstraction is the interface between the learning and controlling and the underlying infrastructure.

Figure 4.4 gives an overview of the Adaptive Building Intelligence application, which resides on top of the ABI System infrastructure part.

The `ABIarea` bundle contains the services related to the static structure abstraction. It is responsible to manage the appearing and disappearing of services related to an area. The area then produces an area state information from its sources. This area state information is broadcasted to interested area controller resulting in an action command for the area. If more then one action command is returned, as more controller are connected, the area must provide some logic to decide which action to take. This decision can be done

| Key | Value | Remarks |
|---|---|---|
| service.pid | BusynessService@55.G.74_ | a `String` object identifying the service within the OSGi Framework instance. |
| wireadmin.producer.flavors | {`String.class`} | an array of `Class` objects defining the formats, e.g. classes, supported by the service to distribute its value. This service support `String` values only. |
| ABILOGSERIVCE_ENABLED | empty `String[]` | if this key is present, log messages sent to the OSGi Framework log service are also specially logged within the ABI System, e.g. written to the database. |
| STATIC_LOCATION | 55.G.74_ | a `String` object specifying a location of the service |
| objectClass | `Producer.class.getName()`, `ABIVirtualDevice.class.getName()`, `BusynessService.class.getName()` | an array of `String` objects containing fully qualified class names under which the service is registered. |

Table 4.1: Example device service registration

with a decision controller.

## 4.4 Other software, and software libraries

Besides the OSGi Framework and ABI System parts, figure 4.1 shows the server services MySQL, Jive Messenger and LNS Server. This sections covers the functionality of the different server services and how they are included into the ABI System

### 4.4.1 MySQL database

The need for a database was quite clear from the beginning of the project. First of all we need to collect the data from the sensors and actuators in a structured way. Being able to access the data with SQL is very powerful, as it is easy to try out different views to the data within a short time period.

We are using MySQL as database server software as it is available on most operating systems and open source. Furthermore we had already experience using a MySQL database from Java.

The `ABIcore Bundle` exhibits a log reader service which catches all the ABI System relevant data. This log reader uses the database to save the log events in a structured and easy queryable way.

### 4.4.2 Jive messenger server

The Jive messenger server provides instant messaging server facilities. An instant messaging server allows clients to register and connect. Connected clients can exchange instant messages or chat with each other. More important for our application is, that the clients can have different presence states, i.e. on line, off line, away, extended away. This presence states are set automatically by the client software after some timeout, or can be set manually.

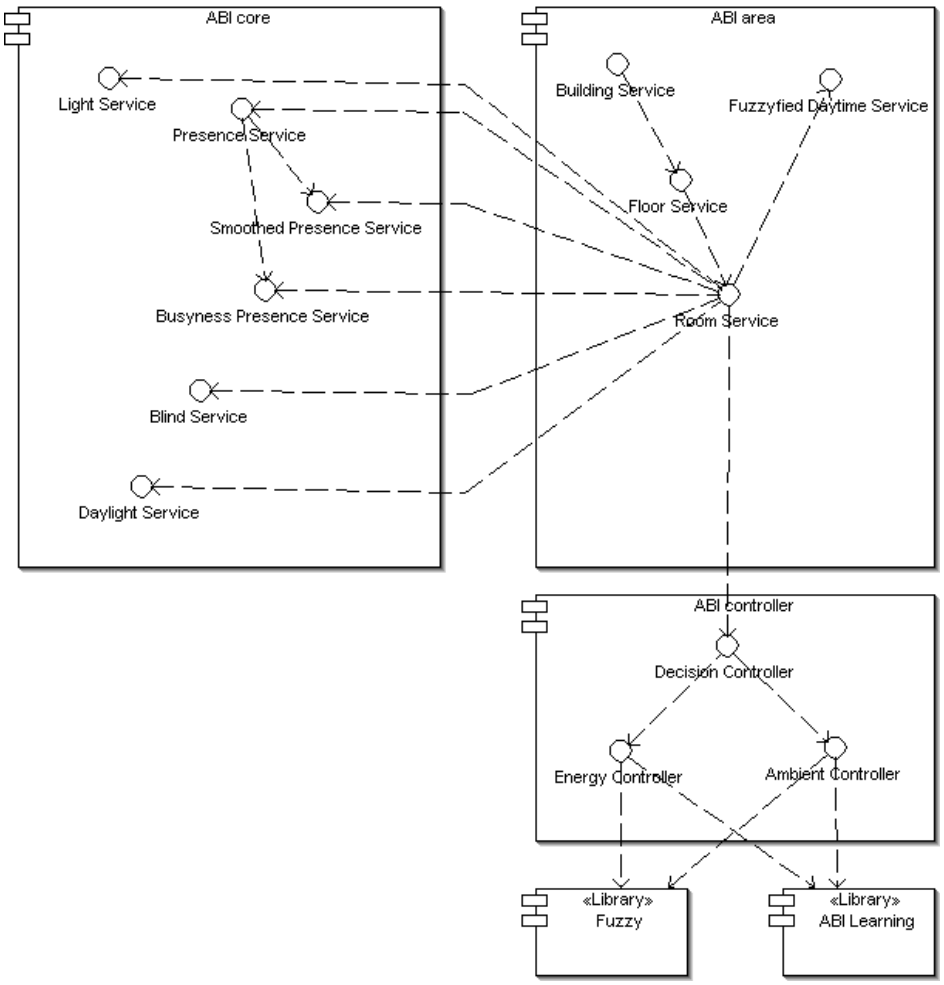


Figure 4.4: Overview of the ABI application

The instant messenger server is seen as additional presence sensor and also as a user interface (GUI) to the lights and blinds in room. This is accomplished by sending a chat message with commands to a specialized user, who is part of the ABI System.

We are using the Jive messenger server as it is open source and implements the XMPP protocol. The XMPP protocol is standardized by the IETF, and also open and freely available.

The `ABImessenger Bundle` is seen as a bus from the ABI System and registers presence devices, representing the connected users. It connects the ABI System to a running Jive messenger server.

### 4.4.3 Fuzzy Library

Fuzzy is a mathematical construct allowing to deal with the fuzziness of language terms like hot, cold, warm. The idea is to be able to calculate with such fuzzy terms. Our predecessor were using the fuzzy architecture from the ABLE framework. Our system also needs a fuzzy library as it is not part of the OSGi Framework. We were using the fuzzy construct by defining a fuzzy daylight and a fuzzy daytime service.

A search on the internet for existing fuzzy libraries revealed lots of different implementations. We have chosen open source fuzzy inference engine for java library [Fuz]. Its key features are the clear and concise way of representing rules, fuzzifying values, and a working fuzzy or-operation.

# Chapter 5

# Design

The architecture chapter 4 provides the high level aspects of the ABI System, whereas the implementation chapter 6 reveals the most detailed view of the ABI System. This chapter is situated in the middle, thus not dealing with all details, but being more specific than the architecture description. The ABI System is divided up into bundles, and each bundle is a collection of related services. A service in turn is preferably an interface. Hence the following sections are bundles and the subsections services, in most cases.

## 5.1 ABI core

Viewed from the hardware side, the core of the ABI System is the mapping of the field bus and devices to software counterparts. But it goes a step further in abstracting the field bus and the real devices into an API[1]. It clearly separates the hardware aspect from the information aspect of devices and buses.

Furthermore we are using the concept of connecting devices with drivers, as it is described in the OSGi Framework specification. The benefits are automated discovery and connectivity management of devices with a corresponding driver. This mechanism is used to attach a field bus, implementing the `BusCategory` interface, to the bus multiplexing driver installed by the ABI core bundle.

Besides of the bus there is also the `ABIBaseDevice` interface, which is used to define the greatest common divisor among devices providing services for the ABI System. The ABI core also defines the most common used sensors and effectors in the context of building automation. It would be possible to move the automation specific devices out of the core to achieve an even more general view. We decided to have them within the core, as they are core services in our application, and it is without restriction of any kind possible to have different devices defined from other bundles.

### 5.1.1 Bus Category

Each service registered as `BusCategory` (5.1) gets automatically collected by our ABI bus multiplexing driver. This automated attachment works only if, the bus service is setting the `BusCategory.DEVICE_CATEGORY_NAME` in its properties. Each service acting as a bus for the ABI System has its own way to `connect()` and `disconnect()` respectively.

As a bus normally has a number of devices connected, it must also provide a mean to identify these devices with ABI System wide unique device identifiers. The `getDeviceidentifiers()` must return these identifiers. Once the identifiers are known in the system, the bus can be asked to register a device with a given identifier as a service. Thus a service appears within the OSGi Framework and provides access to the device information

[1] Application Program Interface

after a call to `registerDeviceAsService()`. Symmetric to this is the `deregisterDeviceAsService()` method call, removing a device from the service space.

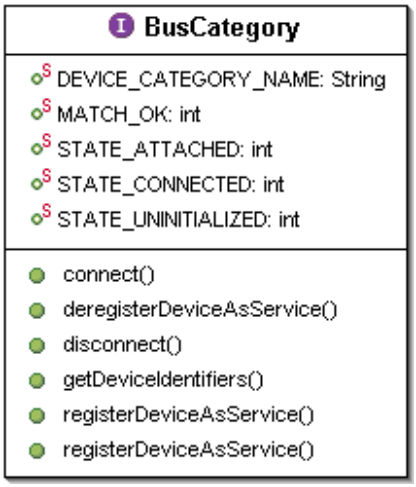


Figure 5.1: Interface bus category

### 5.1.2 ABI Bus Service

The `ABIBusService` interface (5.2) extends the `BusCategory` (5.1) with more administrative methods. This special bus is not attached to the ABI multiplexing driver, to the contrary it is very tightly coupled with the multiplexing driver.

The idea of the `ABIBusService` is, that the clients of the ABI System see only one bus. Thus the `connect()` and `disconnect()` method have now the meaning of connecting, disconnecting all bus devices attached to the multiplexing driver. The `registerDeviceAsService()` and `deregisterDeviceAsService()` are multiplexing the call to the correct bus. To see the registered devices, interesting for the ABI System, one can list them with the `showRegisteredDevices()`. Once having the unique identifier of a device, it is possible to send it commands through the `sendCommandToDevice()`, or `sendCommandToDeviceWithoutExistingWire()`.

As the `ABIBusService` is an administrative service it provides some extra methods to address the different attached bus technologies separately, `connectToBus()` and `disconnectFromBus()`, and `getStateForBus()`. Thus the need for each registered bus to have a unique identifier. The current active bus identifiers can be listed with the `getDeviceIdentifiers()`

The methods `addMonitorToHost()` and `removeMonitorToHost()` allow the creation of a host monitoring tool. Typically this is used for monitoring the availability of a bus technology.

### 5.1.3 ABI Base Device

The `ABIBaseDevice` (5.3) is, as it is stated in the name, the base for all devices which want to play a role in the ABI System. Mainly it demands for the `getLocationString()` as it is a structural valuable information. The corresponding keys expected in the service registration properties are `STATIC_LOCATION` and `STATIC_DETAILED_LOCATON`. The `deregisterDevice()` assumes, that the implementing device knows best how to unregister itself.


Figure 5.2: Interface ABI bus service

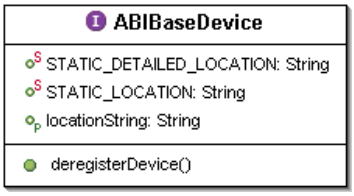


Figure 5.3: Interface ABI base device

### 5.1.4 ABI Log Service

The interface `ABILogService` with the concrete class `ABIdbLogMessageDAO` (5.4) and together with the OSGi Framework's log service are the powerful ABI System log mechanism. Each service in the ABI System can log to the central database or log file by adding the `ABILOGSERVICE_ENABLED` constant as key in its service registration properties. And then use the default log service from the OSGi Framework.

As the log service only allows `String` object to pass as log message, we created the `ABIdbLogMessageDAO` to wrap the logged information. It provides several getter and setter methods of property fields. Then having created and initialized such an instance, it can be serialized, e.g. by calling `toDBLogString()`, into a `String` representation, which can be passed to the default log service. The ABI core bundle registers a log reader capturing all these messages, and deserializes them, e.g. by calling `fromDBLogString()`, for writing to a data base or file entry.

### 5.1.5 Building automation sensors and effectors

As already stated in section 5.1, the following sensor and effector interfaces could also be placed in a separate bundle.

Figure 5.5 gives an overview of the effector interfaces. Blinds can move up and down and may have the possibility to be set a certain position. The most simple effector for the lights are switching them on and off.

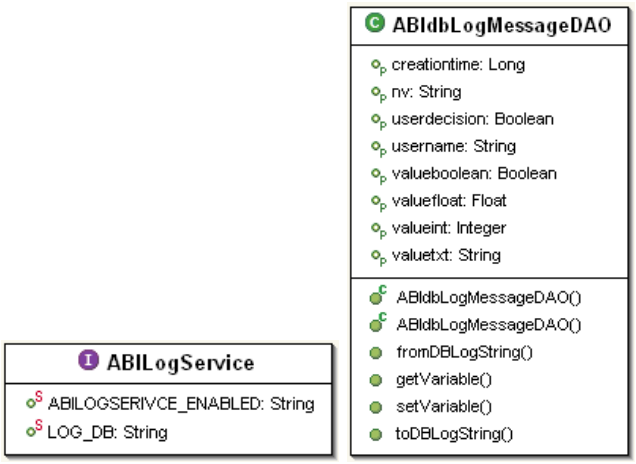


Figure 5.4: Interfaces of the ABI log service

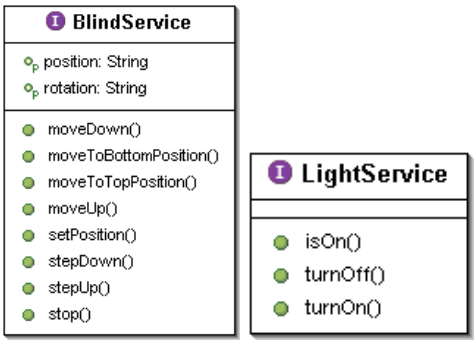


Figure 5.5: Interfaces of effectors

Figure 5.6 gives an overview of the sensor interfaces. The ambient daylight sensor and the room daylight sensor could use the the same interface, but we decided to have them separated as they have different light ranges. The presence sensor provides a signal if somebody is present.


Figure 5.6: Interfaces of sensors

The presented sensor and effector interfaces are very simple and intuitive from the design point of view. Issues that arises when dealing with the underlying real devices must not be underestimated. Problems that we encountered are described in more details in the results chapter in the section 7.3

### 5.1.6 Virtual devices services

Virtual devices are the ones which do not directly map to a hardware counterpart. Some of them represent a different meaning of an underlying sensor, such as a the `BusynessService`.

A good example for the busyness service is the measurement of the busyness of a room. This can be accomplished by refining a presence service with the busyness service. The output of such a busyness device can then be interpreted as the busyness of a room. But it is also possible to refine other device services with a busyness service, the important thing is a valuable meaning of the output. The results chapter discusses

further details in the section 7.2.


Figure 5.7: Interfaces of virtual devices

## 5.2 ABI netmonitor

The `ABInetmonitor` (5.8) interfaces show the factory pattern. This bundle offers the possibility to monitor a host on some port. The `NetMonitorFactory` is used to create `NetMonitor` services. From the interfaces the intention should be clear, that per (host,port) tuple only one `NetMonitor` should be created. The factory pattern controlls the created services. The `ABInetmonitor` provides a service to the OSGi Framework which is totally independent from the ABI System.

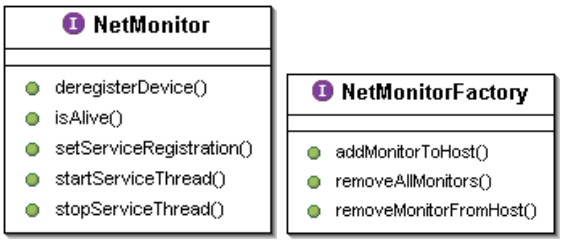


Figure 5.8: Interfaces of ABI netmonitor

## 5.3 Dependencies between the ABI bundles

Up to this section we have described the different bundles comprising the ABI System. It is clear that there are dependencies between the bundles which can be seen from the figure 4.1. We distinguish different types of dependencies:

- library dependency
- OSGi Framework implementation dependency
- feature dependency
- functional dependency

In this section we want to explore these dependencies from the design point of view.

The simplest dependencies are between the library bundles and the rest of the ABI System. These dependency consists of exporting and importing packages. There is also no dynamic involved in this kind of dependency. The fuzzy and reinforcement learning libraries provide data structures and computational services which can be compared to a mathematical library. From this it is clear, that they are not services in the OSGi Framework sense.

As a next level we want to state the OSGi Framework implementation specific dependency, and also how we decoupled this dependency in a way that the reusability, in an other OSGi Framework implementation, for most parts of the ABI System is given. The OSGi Framework specification does not specify the user

interfaces. But it is clear, that every OSGi Framework implementation needs a kind of user interface. Our chosen Knopflerfish OSGi comes with a diversity of different user interfaces, i.e. a graphical swing UI[2], telnet console, terminal console. Our ABI System is dependent on the Knopflerfish OSGi way of providing commands to such consoles. To avoid unnecessary dependencies of all ABI System bundles, we decided to move the commands into an `ABI <somebundle> cmd` bundle for each bundle having commands. Hence it must be possible to install the ABI System on a different OSGi Framework implementation without the Knopflerfish OSGi specific command bundles. But the commands must be reprogrammed for the different platform implementation.

Besides the Knopflerfish OSGi specific issues, we have also an implementation dependency with the `WireAdmin` bundle. This bundle is specified in the OSGi Framework. As it is not mandatory for an implementation to provide such a bundle, it can not be expected to have one in a specific implementation. This is the case with the Knopflerfish OSGi implementation so far. Fortunately we found a `WireAdmin` bundle for the OSCAR OSGi implementation. This was also a prove of the bundle concept as after installing the `WireAdmin` bundle in the Knopflerfish OSGi it worked without problems.

The `ABInetmonitor` bundle is an example for a feature dependency. It can be used in any OSGi Framework implementation completely independent from the ABI System context. And the ABI System is working without the existence of the `ABInetmonitor`, but then lacks a stability feature.

As one may expect, the functional dependency is given between the main ABI bundles. The main ABI application consists of the `ABIcore`, `ABIarea`, `ABIareacontroller`. It is clear that the `ABIcore` does not provide any useful service unless a technology bus, i.e. `ABIlon`, is available in the ABI System. Furthermore if no devices are around, the `ABIarea` and the `ABIareacontroller` are also useless.

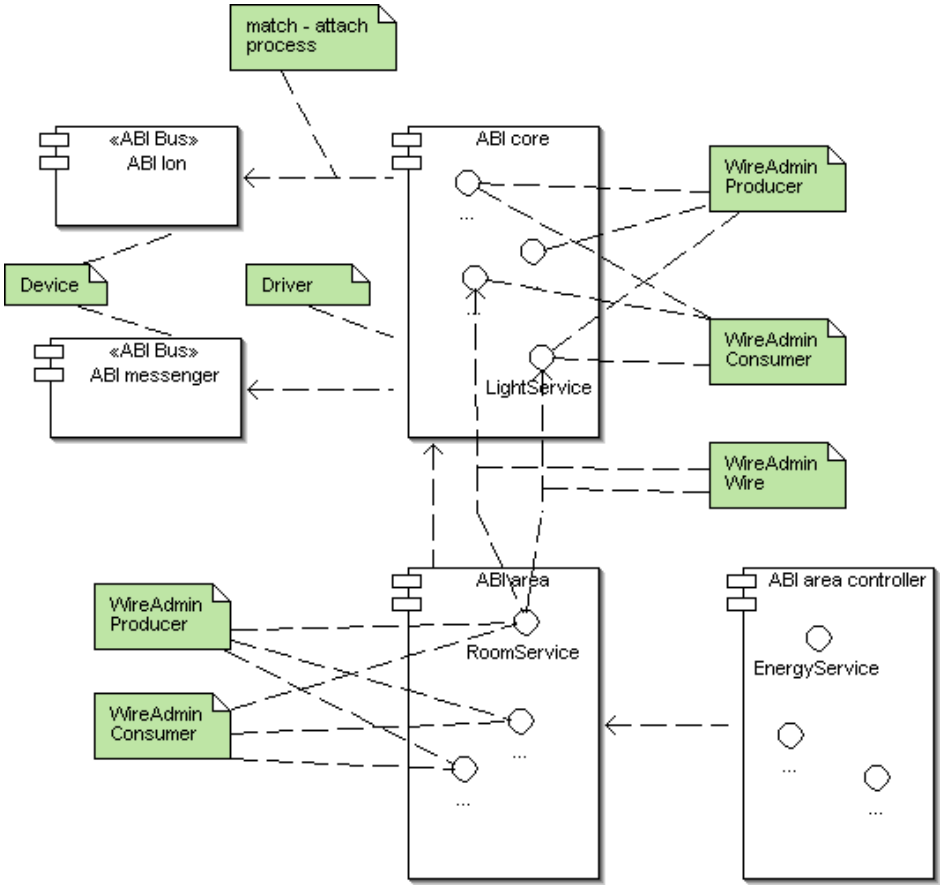


Figure 5.9: Dependencies of the main ABI bundles

[2]user interface

An overview of the dependencies between the main ABI bundles is given in figure 5.9. The green colored remarks indicate the usage of wires and services being producers, consumers respectively. In addition to the producer - wire - consumer dependency, figure 5.9 shows also the dependency with the device - driver connection.

## 5.4 Device - driver concept

The device - driver concept as it is described in the OSGi Framework specification was very well suited for our abstract bus concept. We decided to define a multiplexing driver and the different bus technology software counterparts as devices. With this setup we are able to make use of the device - driver concept to its full extent. As a consequence bringing up a new bus technology requires four steps:

- bus access bundle, registering a service as a ABI Bus device
- install and start the bundle.
- issue a connect command, either manually or automatic
- register devices as services, either manually or automatic

By installing and starting such a bundle it is automatically detected by our multiplexing driver, and the bus is available in the ABI System without stopping, or restarting it.

Additionally the device - driver concept can also be used to refine device services. This concept is very powerful as we can see from the following example. Let us consider the presence service, the software representation of the movement detector. The most raw form of this service brings in all peeks from movement updates. Whereas a peek is a recognized movement resulting in a message carrying a 1. In a busy room this results in a lot of movement peeks during a short time. To extract the presence information from this movement detector a refinement is needed in a way to smooth the information. This can be accomplished by defining a driver refining the raw presence service to a smoothed presence service with a configurable time out. Thus the driver gets attached to the presence service device, and in turn registers a smoothed presence service.

## 5.5 Producer - wire - consumer concept

Application sensors are producers generating values which are consumed by interested services. Effectors are clearly consumers, as they receive values initiating an action, i.e. setting a light level, or moving blinds in a position. But they are also producers in the sense that they generate status messages.

Producers and consumers are connected with a wire object which is managed by the `WireAdmin` bundle. Using wires instead of the observer pattern has several advantages:

**weaker coupling** producer and consumer are not coupled by having references to each other, instead they share the wire object.

**central managed persistence** creation, deletion and changes on wires are managed by the wire admin, also responsible for the persistence of the wires.

**hand shaking for the best supported value exchange format** producer and consumer have to agree on a format how to exchange data. The handshaking is established during the set up of the wire.

**long living connection** the wire is even surviving if both of the wire ends, producer and consumer, vanish. But the wire comes back with the appearing of the producer and consumer.

## 5.6 Asynchronous parts

As already stated in the architecture chapters section 4.1, the whole ABI System is a highly heterogeneous distributed system. Now we are bringing together all the described parts, e.g. the bundles and the infrastructure connecting the different bundles, services, and parts. The OSGi Framework specification is pointing out that care must be taken with the asynchronous nature of the problem domain. But it does not prescribe or specify asynchronous concepts. As a consequence there must be a concept to handle asynchronous messages at various places in the ABI System.

**raw updates** are the events from real hardware or virtual devices

**wire updates** are the update calls to the wires from the producers.

**log messages** are captured and written to a database or file.

Potentially the ABI System contains a few hundreds of devices which are receiving updates totally independent from each other. They process the update and generate a new update to be propagated in the ABI System. The figure 5.10 clarifies these different update sources and sinks.

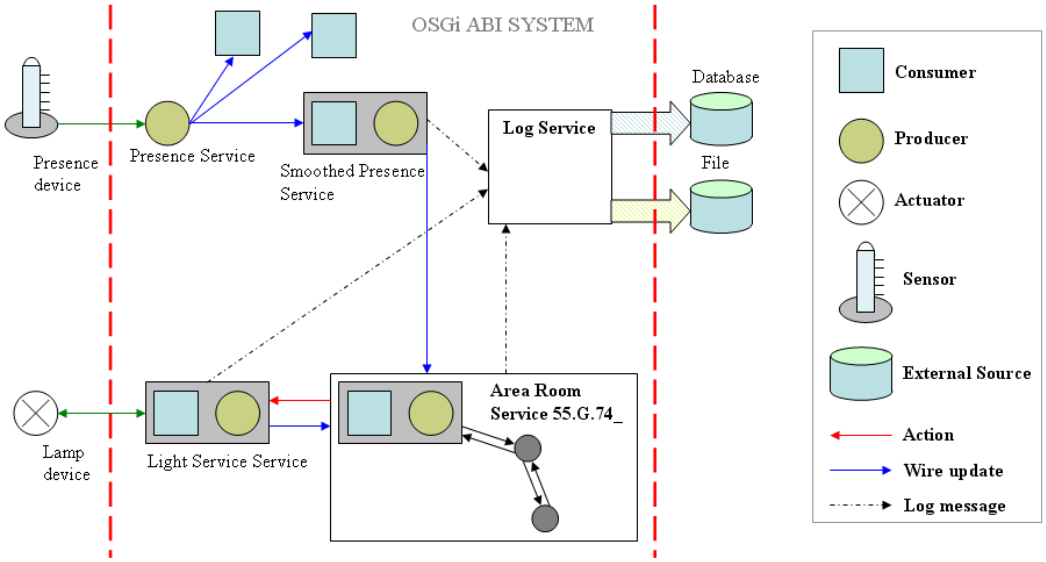


Figure 5.10: Overview of the different system updates

We postulate in the ABI System that the update receiving part is responsible to return immediate from the update method call. Hence the receiving parts fetch the information from the update, putting it on an intermediate stack. With this approach we want to improve system response time and avoid the lost updates problem encountered with earlier versions of the ABI System. A drawback is the lots of threads running. Future works can focus on optimizing the number of threads.

# Chapter 6

# Implementation

This chapter shows parts of interesting implementation details in the ABI System. It should serve as additional documentation to the javadoc[1].

## 6.1 Wireadmin service

Its responsability is the administration of wires, e.g. creation, persistency, and handling connectivity. A wire is used to connect a consumer with a producer: parties can negotiate the best way of exchanging values. The producer then uses the wire to update its connected consumers.

Thus having a network of sensors and effectors, we are using the wireadmin service bundle for connecting the sensors with the effectors as producer - consumer pairs. Moreover there is the possibility of creating converters which act as consumers and producers at the same time.

Originally designed for the OSCAR OSGi implementation, the wireadmin service bundle was succesfully deployed in the Knopflerfish OSGi implementation. The availability of the wireadmin service bundle was mission critical, because otherwise we would had to implement it ourself. During the development of the system we had to fix some bugs and inconsistency in the existing implementation. It is no longer a third-party-only library as we were hard working to fix the critical problems:

- zombie threads after stopping, restarting
- inconsistency with services registered as `WireAdmin.Producers` and `WireAdmin.Consumers` at the same time.

We shortly present our patches here. The thread problem is solved in the inner class `AsyncMethodCaller` of the class `WireAdminImpl`. The `stop` method was adapted to set a stop flag and send an `InterruptedException` to terminate the thread.

The problem with services registered as `WireAdmin.Producers` and `WireAdmin.Consumers` at the same time, was showing up in the following case. The situation was having a wire created and connected from a producer only with a service registered as consumer - producer. When we tried to send updates over that wire it was not working. In short, the problem was an `if --- then --- else ---` statement. The solution was to transform the former into two `if --- then ---` statements, with all the consequences to related code.

The wrong `if --- then --- else ---` was found in the `WireAdminImpl.serviceChanged()` method checking first if a registered service is a consumer, e.g. the `if (m_consumerFilter.match(serviceRef))`. In the else clause it was checking for the producer, e.g. `else if m_producerFilter().match(serviceRef))`. We

[1] generated java code documentation

changed it in such a way, that a service is checked for its consumer property, and the consumer specific actions are taken. Next in the control flow the producer property is checked, and the producer specific actions are taken. These changes also had an impact on the inner class `AsyncMethodCaller`. Before the change there was one `m_methodCallStack` which we had to split up in a `m_methodProducerCallStack` and `m_methodConsumerCallStack`.

## 6.2 Lon bus

The LON specific bus and device implementation is found in the `ABIlon` bundle. Several libraries concerning the LON accessibility are used. These libraries are not above suspicious. We want to point out our implementation of user input detection in the case of turning on and off the light. And also how to bring up the devices automatically when connecting the OSGi Framework to the LON bus.

### 6.2.1 Detection of manually switched lights

It is of utmost importance to know if a light is switched on by the system or if it was a real user interaction. User interaction means pushing a the button on the wall socket.

### 6.2.2 Register Lon devices as services upon startup

Registering devices as service is done with the `registerDeviceAsService()` method from the `LonServiceImpl`. Either this is called from another service in the OSGi Framework or as it is normally the case, from the `ABIbus` service, the abstact bus.

One special case is the startup phase. Whereas startup is seen as the moment, when the ABI System connects to the LON bus. We have a xml configuration file containing all the information to create the services for the different available devices. This file is read by a thread registering each of the encountered device as a service in the OSGi Framework. An example entry in the configuration file for the room 74_ looks as follows:

```
<area type="room" name="INI Room 74" location="74">
    <area type="room" name="INI Room 74_" location="_">
        <device type="occup">
            <nv>Y355G-4174_-HTS_ECO.nvoOccup.2</nv>
        </device>
        <device type="light" location="corridor">
            <nv>Y355G-4174_-PHI_LRC_A.nvo0102LampVal.26</nv>
            <nv>Y355G-4174_-PHI_LRC_A.nvi0102irSetting.14    </nv>
            <nv>Y355G-4174_-UPL_LT_A.nvoSetting[2].2</nv>
        </device>
        <device type="light" location="window">
            <nv>Y355G-4174_-PHI_LRC_A.nvo0102LampVal.27</nv>
            <nv>Y355G-4174_-PHI_LRC_A.nvi0102irSetting.15</nv>
            <nv>Y355G-4174_-UPL_LT_A.nvoSetting[3].3</nv>
        </device>
        <device type="daylight">
            <nv>Y355G-4174_-HTS_ECO.nvoDaylightLux.5</nv>
        </device>
        <device type="blind" model="big" location="middle" function="single">
            <nv>Y455G-4174_-MS_STA_G19-20.nvoPositionFb.14</nv>
            <nv>Y455G-4174_-MS_STA_G19-20.nviSettingLcl.12</nv>
        </device>
        <device type="blind" model="big" location="left" function="single">
```

```
            <nv>Y455G-4174_-MS_STA_G19-20.nvoPositionFb.15</nv>
            <nv>Y455G-4174_-MS_STA_G19-20.nviSettingLcl.13</nv>
        </device>
        <device type="blind" model="big" location="right" function="single">
            <nv>Y455G-4174_-MS_STA_G18.nvoPositionFb.14</nv>
            <nv>Y455G-4174_-MS_STA_G18.nviSettingLcl.12</nv>
        </device>
    </area>
</area>
```

As we can see, some devices have more then one network variable[2], this is typical for effectors. An effector has input and output ports which are represented by the various network variables.

## 6.3 Messenger bus

The ABI System must have an instant messenger account on the messenger server to which it should be connected. In fact, the messenger bus is logged in ABI System having the building's people as contacts in its contact list. The messenger bus then registers the contacts as presence device services with a location information. As the contact list is organized by having group names which correspond to the room identifiers. It is an easy task for the messenger bus to use the group name as location information. Moreover this gives also an easy administration utility. If people move from one room to another, this can be on line administrated in the messenger bus by moving the contact from one group to another group, e.g. move the person from one room to another.

In addition to the presence information which is extracted with the help of the instant messenger setup it also gives the possibility to send commands. For example people can turn on their room lights, or moving down the blinds without touching the wall sockets. The people are also aware whether the service is not available, either because the messenger bus sends them back an instant message, or they are aware that the ABI System is not on line.

## 6.4 Fault Tolerance, Stability and Recovery

From the discussions with our predecessors and people from the institute it was clear that it is a must for such a system to ensure stability. Stability in the sense of a system which is not turning off the lights and moving down the blinds, in the case where no more inputs are detected. And also to bring the system back, once inputs are detected again.

Fault tolerance is expected from such a distributed system, as we had in mind the well known *Eight Fallacies of Distributed Computing by L. Peter Deutsch*

> The Eight Fallacies of Distributed Computing
> Peter Deutsch
> Essentially everyone, when they first build a distributed application, makes the following eight assumptions. All prove to be false in the long run and all cause big trouble and painful learning experiences.
>
> - The network is reliable
> - Latency is zero
> - Bandwidth is infinite
> - The network is secure
> - Topology does not change

[2] LON specific way of addressing ports on installed devices.

- There is one administrator
- Transport cost is zero
- The network is homogeneous

### 6.4.1 Netmonitor

Indeed we expected to have a reliable local area network, which turned out to be false when one of our two switches had a failure. This failure lead us to design the netmonitor as described in the section 5.2.

The netmonitor is implemented as thread. It has a heartbeat at which it checks the availability of a (host,port) tuple. If a socket can be opened it sends out a `true` message on its wires. Otherwise the creation of the socket fails with an exception and a `false` message is sent out.

Consumers of the netmonitor information can take countermeasures when a host goes down and react appropriate if it is available again. Netmonitor clients defer the polling of a host to the netmonitor. If there are more services interested in the same (host,port) tuple the system connects them automatically with the singleton instance.

# Part III

# Results

# Chapter 7

# Results and Discussion

In this Chapter we present the main results of this diploma work.

## 7.1 Data interpretation

Data was collected right after the start of the diploma work. But not all this data can be evaluated as a stable version of the software had to be worked out first. In this section we are focusing on the data collected during the last three weeks of our diploma work (Monday $18^{th}$ April 2005 till Wednesday $4^{th}$ May 2005). Data mining and interpretation takes place in the Institute of Neuroinformatics, building 55 floor G, situated as shown in the following figure 7.1.

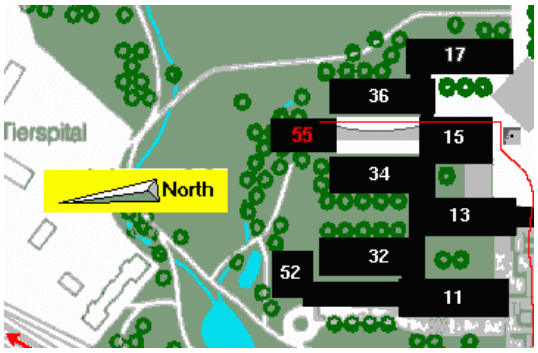


Figure 7.1: Neuroinformatic Institute situation on the campus

The weather conditions during this time changed from cloudy and rainy to sunny. If appropriate, the subsections describe the particular meteorological situation in more details, see 7.1.

We have chosen the rooms 75, 70, 84, 27 and 54 for the further discussion. Room 74 is used for testing purposes. Hence it is not reflecting the needs of inhabitants. The following reasons influenced the choice of rooms:

1. purpose (working, meeting storage)

2. orientation (west, east)

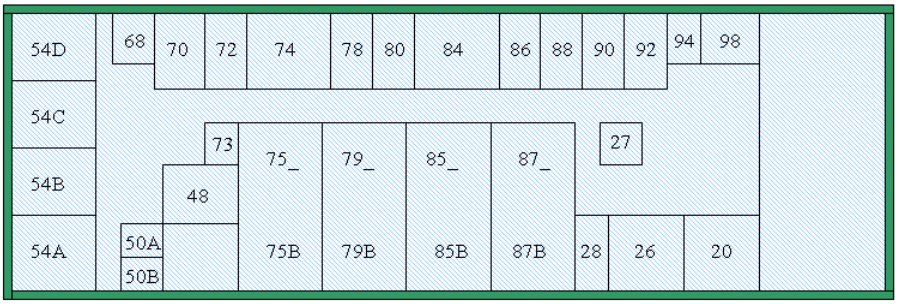


Figure 7.2: Building 55, floor G, Institute of Neuroinformatics

### 7.1.1 Weather condition influence the daylight

The brightness of a room is very subjective in terms of its users. In figure 7.3 we show the difference between a rainy and a sunny day. It makes sense to integrate the weather information into the new ABI System. This can improve the learning task once more for follow up works.

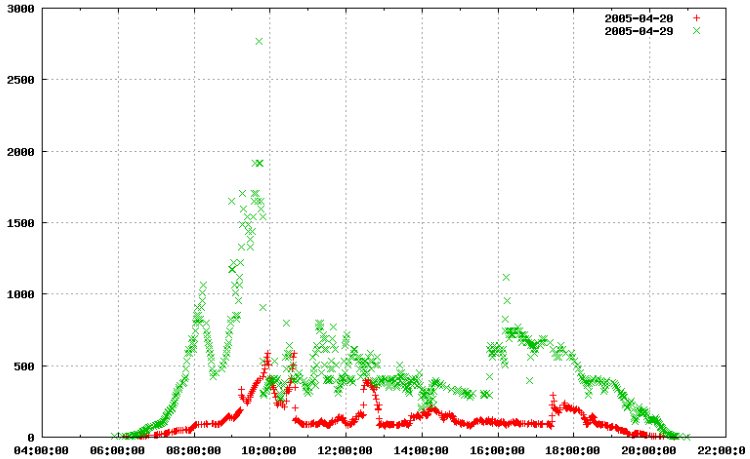


Figure 7.3: Weather influence on the daylight recorded on the $20^{th}$ and $29^{th}$ April 2005

### 7.1.2 Daylight

One problem we encountered at the start of the work were reliable definitions for the daylight's membership functions. This fuzzy variable plays a major role in our whole ABI System, because it defines the darkness, or brightness in an area. Thus the main task is to decide whether to turn on or off the light devices. The blinds were not taken into account, due to the hardware issues not solved till the end of our diploma work, see 7.3 for further information. The values for the membership functions are shown in figure 7.4 and have been evaluated on previous collected data.

## 7.2 Virtual devices

Beside the hardware devices which are represented by software services, we defined also virtual services. The term virtual service is used for a specific service type. For example one can think of a virtual service which summarizes all attendant persons in a building and displays the occupied rooms on a screen. Such a service is using different presence sensors. In other words, a direct mapping from a hardware device to any software

| date | min. temp.$[C^o]$ | max. temp.$[C^o]$ | wind $[km/s]$ | rain $[l/m^2]$ | sunshine $[min]$ | daylight $[lux]$ |
|---|---|---|---|---|---|---|
| 2005-04-19 | 6.1 | 10.9 | 39.66 | 3.8 | 17 | 1400 |
| 2005-04-20 | 5.2 | 6.5 | 31.0 | 16.4 | 0 | 430 |
| 2005-04-21 | 3.0 | 11.6 | 29.9 | 0.6 | 303 | 5620 |
| 2005-04-22 | 0.3 | 15.1 | 22.3 | 0.0 | 746 | 3520 |
| 2005-04-23 | 5.4 | 17.1 | 48.6 | 6.6 | 232 | 5370 |
| 2005-04-24 | 8.5 | 13.2 | 25.9 | 6.3 | 24 | 1090 |
| 2005-04-25 | 7.6 | 11.4 | 47.9 | 5.2 | 9 | 890 |
| 2005-04-26 | 8.1 | 16.5 | 38.9 | 0.0 | 335 | 3440 |
| 2005-04-27 | 8.26 | 10.3 | 30.2 | 8.6 | 0 | 400 |
| 2005-04-28 | 8.2 | 18.2 | 19.1 | 0.1 | 259 | 5620 |
| 2005-04-29 | 11.6 | 23.6 | 28.1 | 0.0 | 526 | 3600 |
| 2005-04-30 | 11.0 | 25.2 | 22.0 | 0.0 | 768 | 3600 |
| 2005-05-01 | 12.8 | 26.9 | 20.5 | 0.0 | 767 | 3520 |
| 2005-05-02 | 13.7 | 26.9 | 62.6 | 0.0 | 560 | 3370 |
| 2005-05-03 | 9.7 | 20.1 | 67.0 | 14.9 | 69 | 3930 |

Table 7.1: Weather information provided by MeteoNews Zurich

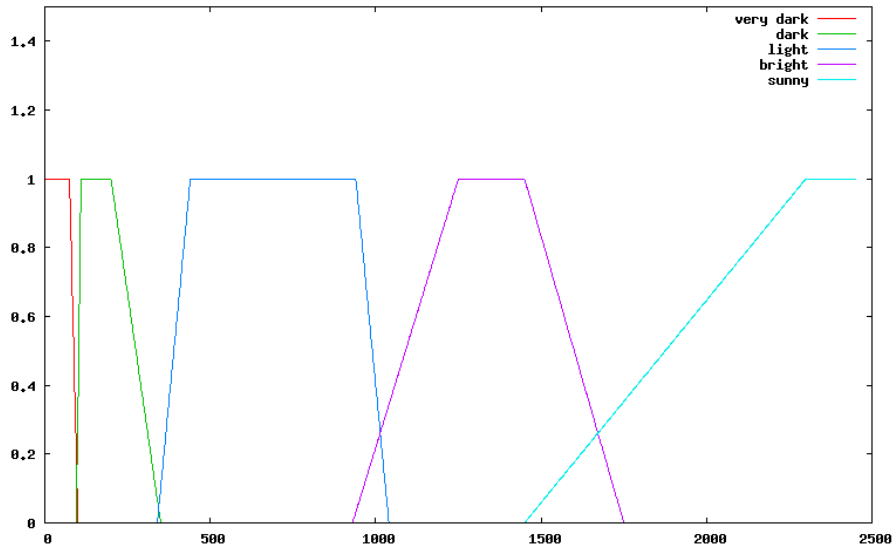


Figure 7.4: Membership functions of the fuzzy daylight linguistic variable

is called a device service and depending services which make use of them are called virtual device services, see figure 7.5.

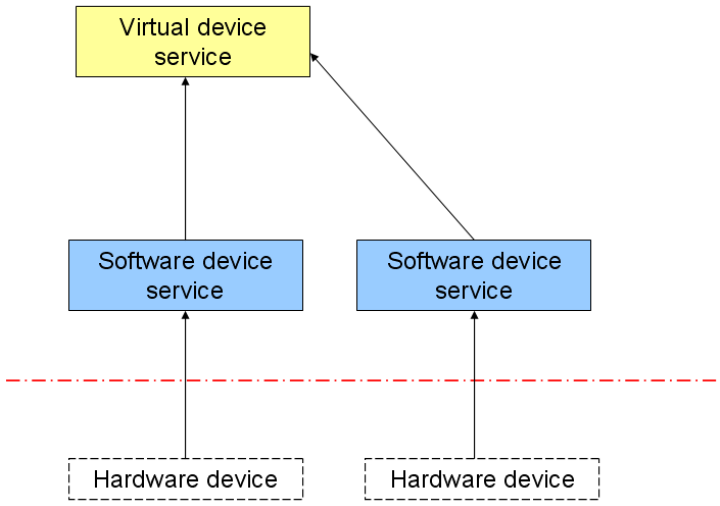


Figure 7.5: Mapping from hardware to software and then to virtual device services.

### 7.2.1 Busyness service and smoothed presence service

A common problem with raw data is that it has to be adapted to the needs of a service. Raw data provided by the hardware presence device is a set of $(true, false)$ values indicating a persons attendance. Mapping the hardware presence sensor into software leads to a presence device service. Based on values provided by the presence device service we created two virtual devices:

1. a busyness device, which tells us how busy a room is, see figure 7.6.
2. a smoothed presence device, which indicates whether a person is present or absent using a certain timeout, see figure 7.6.

We show in figure 7.7 a data sample provided by the busyness device recorded in room 55.G.28 during approximately one hour. The busyness service counts the number of movements within a adjustable time limit and logs this number to the database. In our example a lot of movements were counted so we can state this room as busy.

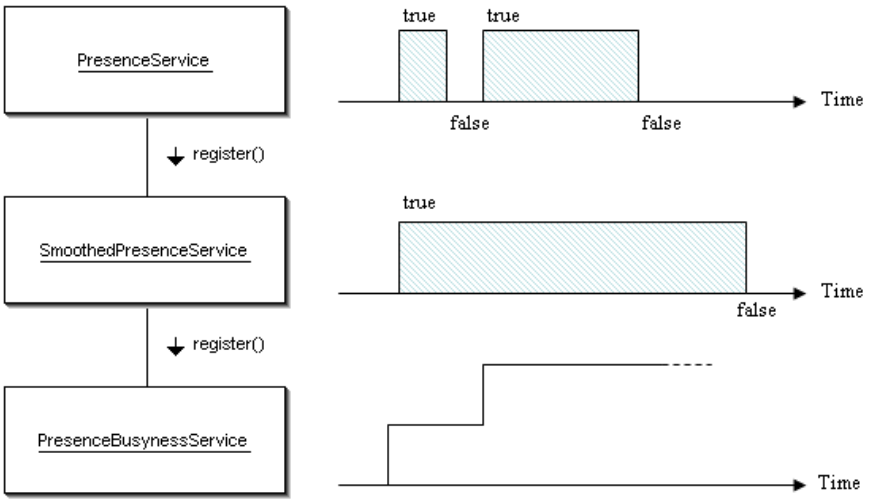


Figure 7.6: Presence, smoothed presence and busyness service

The major problem with the presence sensor is that in some cases it is not able to detect the attendance of a person correctly. Mainly if the sensor's view on the person is hidden by any furniture or if the movement of a person is very little. In order to get more reliable presence detection we used a common messenger and integrated this presence value in our system.

### 7.2.2 Messenger

A messenger per user has to be installed to improve our presence detection. Once this is done we then have a more reliable input for our learning module, since the messenger detects any keystrokes or movements invoked by the attached mouse. This messenger device can be seen as a virtual device service, although it is attached via a bus connected to our new developed ABI System see 7.5 and 6.3 for further information.

### 7.2.3 Fuzzified daytime device

The fuzzified daytime device service divides the day into episodes. This can be useful for the learning task since it is then possible to learn for different time periods. That way the daylight is not one precisely defined value of brightness. It is depending on sun azimuth and elevation, as well as cloud cover.

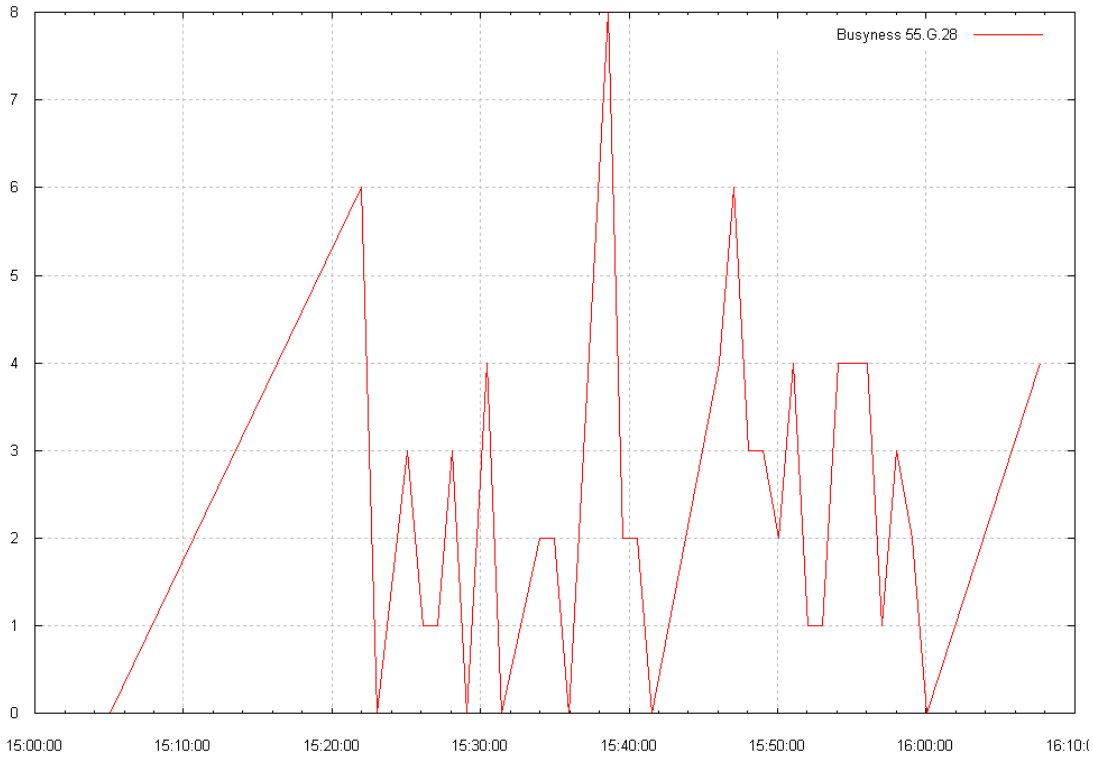


Figure 7.7: Virtual busyness service, recording time approximately 1 hour, room 55.G.28.

For example daylight situations are certainly different in terms of early morning and late afternoon. The learning task could adapt these information and result in a more precise learning. This daytime device is also considered as a virtual device.

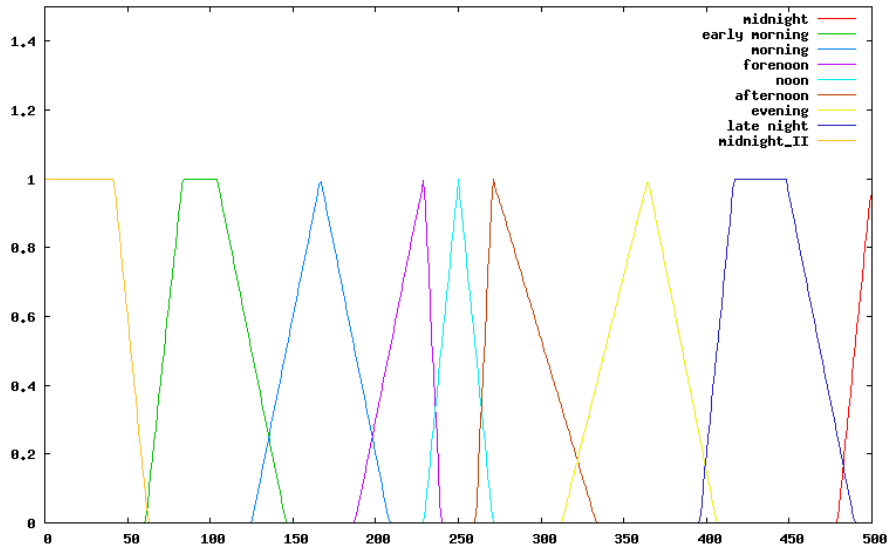


Figure 7.8: Membership functions for the fuzzified daytime device..

## 7.3 Hardware issues

In this section we discuss the hardware issues encountered with several hardware devices and state what impact results for the learning task.

### 7.3.1 Blind device

The blind hardware devices cause a lot of problems, mainly because they can get out of sync, which is very hard to detect. If a command is sent to a blind device that is out of sync , it results in a unpredictable behaviour such as moving blinds down or up randomly. This leads to another problem with the blind device. Whenever getting out of sync the current position of the blinds is returned wrong by the device. It is pretty obvious that such a device is not adequate for learning since it causes wrong input to the system. We decided not to use the blind device for our learning task. All the same it is possible to send commands to the blinds. We integrate a work around for the blind problem as we defined, that it is only possible to move blinds up or down, and stop moving - in those cases we are neither interested in the correct position of the blinds nor do we care about wether the movement is working correctly, as the user can stop the issued command at any time. This is similar to the wall switches used. In other words, we mapped the behaviour of the wall switches into software, although our software blind service is able to handle many additional commands such, stepping up or down, move to a exact position and many more. We considered this hardware issue as a drawback for the overall goal of learning, as it is not possible to actively control daylight in a room. Actions that can be issued by the learning system are cut down to the light device, which can be turned on and off. All other environment changes are then issued by the user, since for example he can move down the blinds completely even if this might be not adequate to the outer brightness conditions at all. Hence it makes no sense to use the ambient daylight neither, unless the hardware issue on blind devices is solved. Imagine a situation with sunny weather but the blinds are completely down and the light is turned on in the considered room. In this case the indoor and outdoor daylight do not correlate at all. Therefore it is important to know the correct position of the blinds at any time.

### 7.3.2 Daylight device

Figure 7.9 shows the daylight values in the period of $19^{th}$ April to $29^{th}$ April 2005. At first glance it leads to the conclusion that this is a quite reasonable behaviour of the daylight. Without worrying about details

one can see that the values raise in the morning and then slowly decrease in the late afternoon, except for some spikes. Taking a closer look to the daylight values, shows a surprising behaviour of the sensor, see figure 7.10. Within minutes the daylight value increases and decreases from values 13333 to 2916, see table 7.2. It goes without saying that such value oscillating is very hard to handle. Smoothing the daylight values would be a simple but not very effective solution, since we have experienced oscillation in many different situations. It seems that reflecting material outside of the building can have a great influence on the sensor.

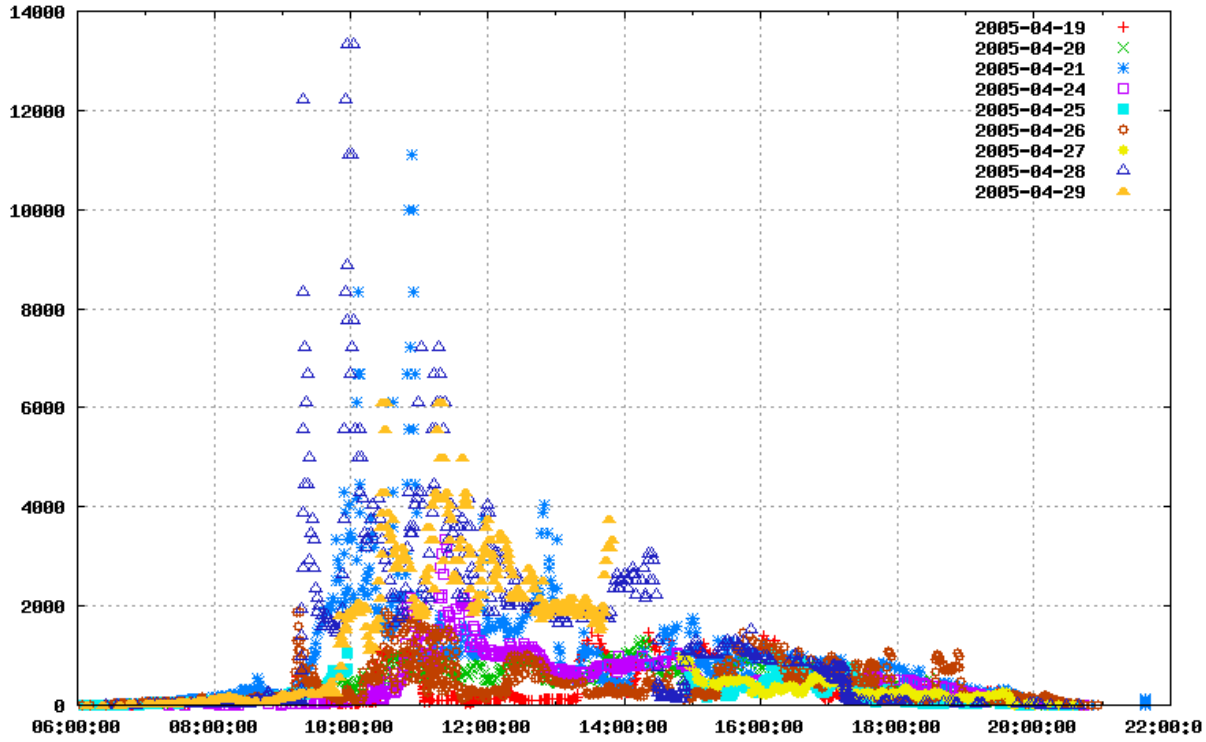


Figure 7.9: Daylight recorded on $19^{th}$ April to $29^{th}$ April 2005 in room 55.G.74_.

| time | daylight value |
|---|---|
| 2005-05-01 07:54:31 | 13333 |
| 2005-05-01 07:55:31 | 8888 |
| 2005-05-01 07:56:25 | 5000 |
| 2005-05-01 07:57:23 | 6111 |
| 2005-05-01 07:57:38 | 8888 |
| 2005-05-01 07:58:36 | 5000 |
| 2005-05-01 07:58:54 | 2916 |
| 2005-05-01 07:59:23 | 4166 |
| 2005-05-01 07:59:30 | 6111 |
| 2005-05-01 07:59:45 | 8888 |
| 2005-05-01 08:00:10 | 13333 |
| 2005-05-01 08:08:09 | 11111 |

Table 7.2: Toggling of daylight values in room 55.G.74_

Better performance of the sensor has been recorded in ares where the sensor is located far away from the windows as shown in figure 7.11 and 7.12. Using the equation 7.1 we calculated the difference between the maximal and minimal value. The room *55.G.75_* is equipped with a daylight sensor which is located far away from any window, in contrast to room *55.G.75B* where the daylight sensor is located approximately 3 meters away from the window front.

$$\delta_{daylight} = |max_{daylight}(t) - min_{daylight}(t)| \text{ t} \in time\{7:30 - 8:15\} \quad (7.1)$$

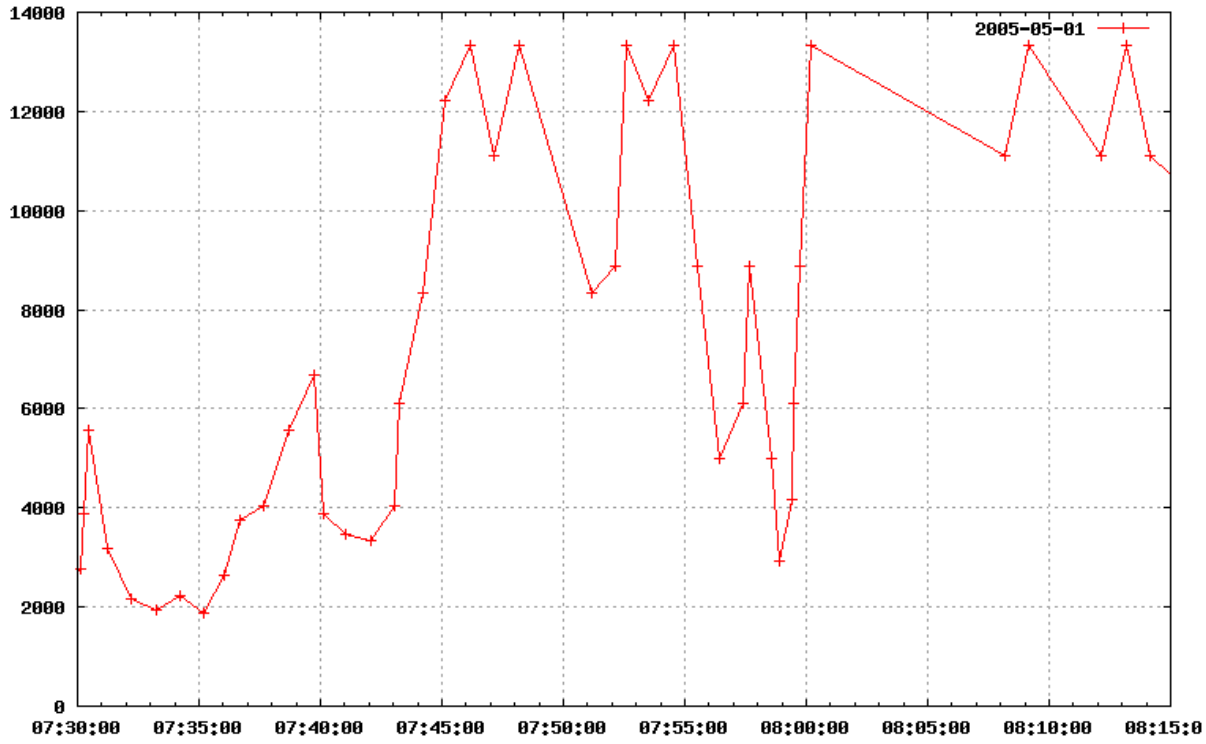


Figure 7.10: Daylight recorded on $1^{st}$ April in room 55.G.74_ between 7:30 a.m. - 8:15 a.m.

$$\delta_{daylight_{55.G.75B}} = 3122 \text{ room } 55.G.75B \tag{7.2}$$

$$\delta_{daylight_{55.G.75\_}} = 206 \text{ room } 55.G.75\_ \tag{7.3}$$

The daylight sensor is very important in the learning task, since turning on and off the lights is mostly based on the daylight input.

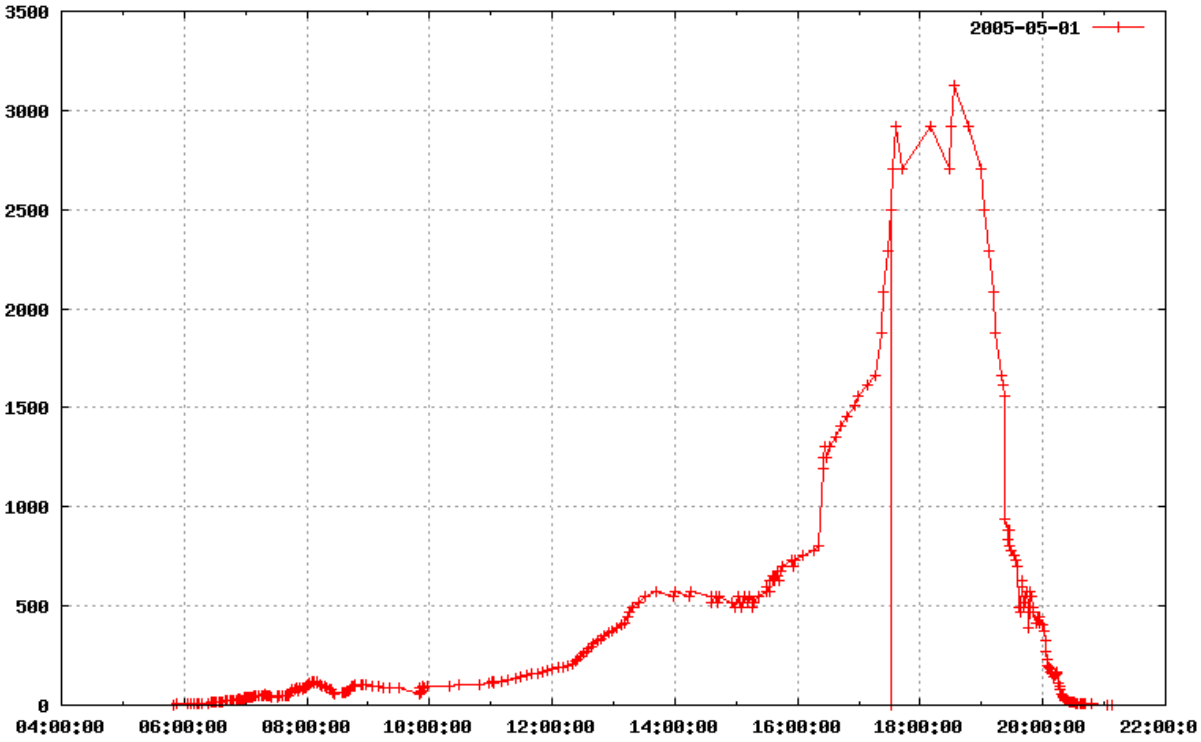


Figure 7.11: Daylight recorded on $1^{st}$ April in room 55.G.75B between 5:50 a.m. - 9:07 p.m.

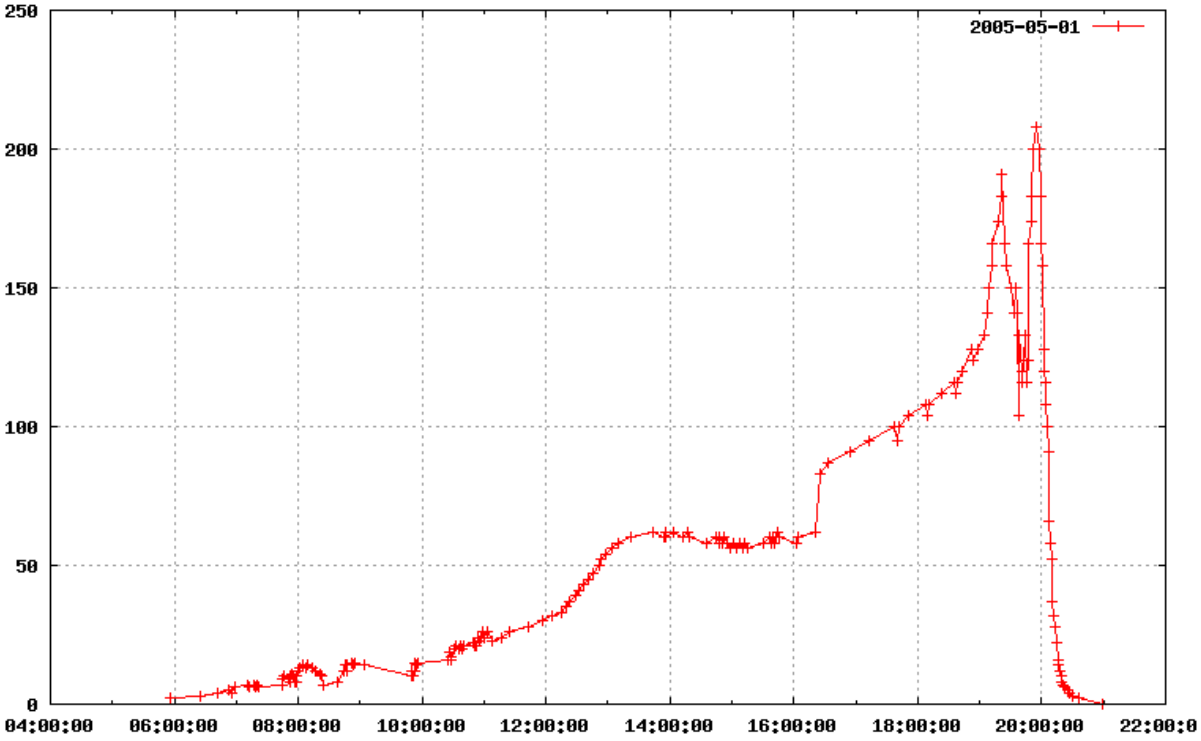


Figure 7.12: Daylight recorded on $1^{st}$ April in room 55.G.75_ between 5:50 a.m. - 9:07 p.m.

### 7.3.3 Presence device

As described in the previous section 7.2 the presence sensor should be seen more as a movement detector than as a presence detecting device. It actually detects only movements of a person in a certain range. Problems occur if movements are very little or the sensor has no intervisibility with the person. Different settings are possible on the hardware, we decided not to smooth the movements detection by the sensor itself. We even created different software services which do smoothing and busyness detection. It is then possible to extract more information about the attendance of persons in that way. To ensure further inquiries we recorded the raw data as well as all extracted information in the database.

### 7.3.4 LNS Server setup

Another hardware problem we found was the setup of the LNS Server and the LON gateway. The current network connections are as follows. The LON gateway is connected to a public IP switch. From this network switch a connection is laid to the LNS Server. Other network hosts (clients) are connected to the before mentioned switch as well, see figure 7.13. This raises a network traffic problem, since any updates issued by devices connected to the LON gateway are sent via a network connection first to the switch and then to the LNS Server. In general all communication between the LNS Server and the LON gateway is done via a network switch. Since a lot of updates and messages are created within the LNS Server and the LON gateway this setup is completely wrong. An improved setup is presented in figure 7.14, where the LON gateway is directly connected to the LNS Server using a secondary network interface in the server. The network traffic would be reduced significantly and the overall speed increases in such a setup, resulting in a more reliable system. A common problem with asynchronous update messages is that messages not necessarily arrive in the same order than they where sent. Some update messages we received by the system, had a large delay in comparison with the event causing them, but this is rather a network issue than a LNS Server problem. We strongly recommend to change the network setup (see 7.14).

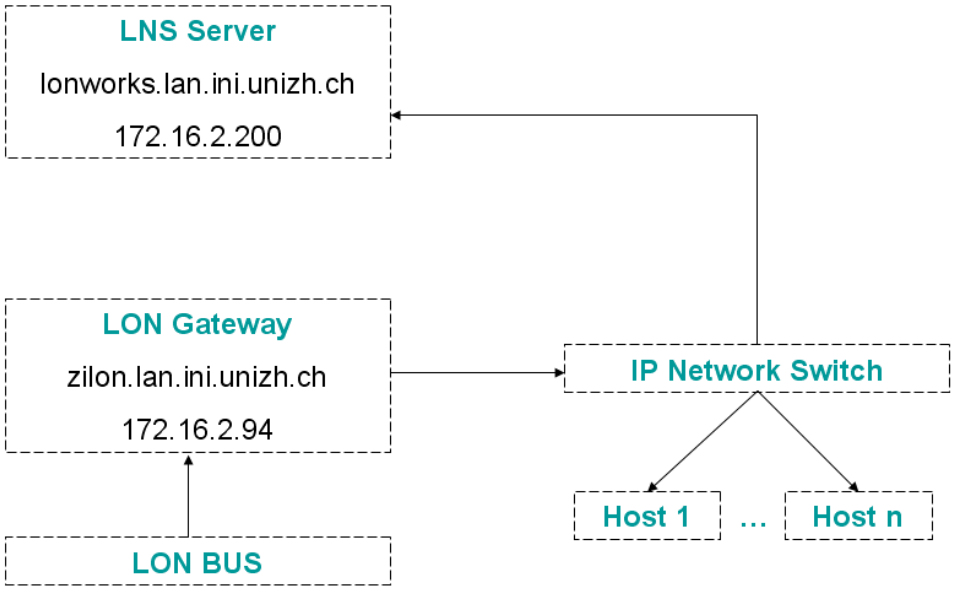


Figure 7.13: Current LNS Server, LON gateway and host network setup.

### 7.3.5 Conclusion

Reliable data input and output from the sensors and actuators is important. It makes no sense to use a device which gives unreliable input, in particular within a learning task. Wrong data input can lead to

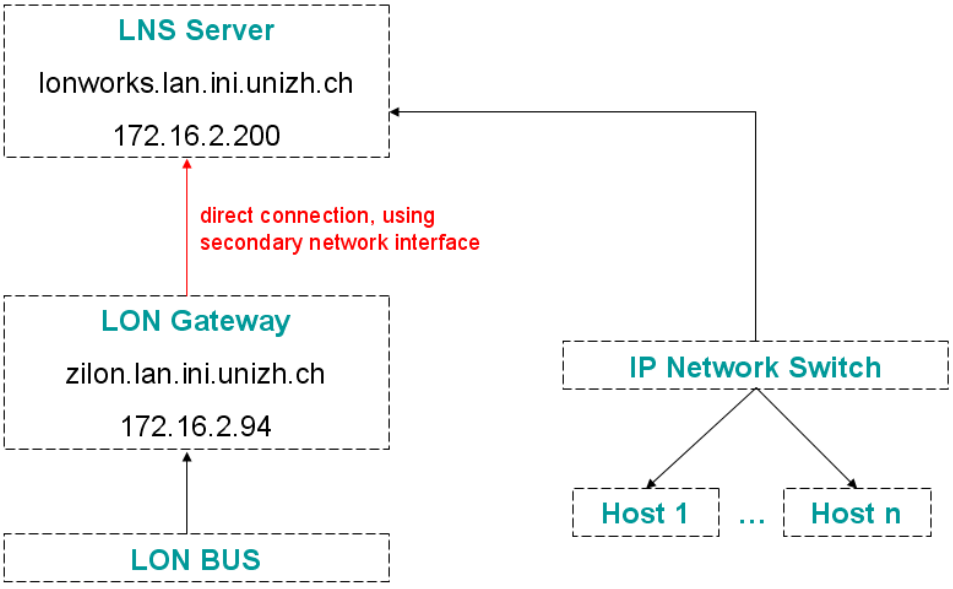


Figure 7.14: Improved LNS Server, LON gateway and host network setup.

incorrect learning. A filter mechanism which gets rid of suspicious data or a rating mechanism for all devices would be very useful, see future work 8.

## 7.4 Detecting user interaction

In this section we describe our approach of detecting user interaction with the system. There exist two different ways a user can interact with the system. First, at any time the user can press the manual switch for light and blind changes, second it is possible to change the actual state by giving commands through software, see 7.5.

### 7.4.1 Detection of manual switch changes

Manual light switches are integrated in every room. The problem we encountered is the way those switches interact with the LNS Server. In general an update is created whenever a switch is pressed manually. In case any software interacts directly with the light device through the LNS Server this update is not created. Updates are created only if the manual switch device and the light device have a different states, i.e. manual switch 'state = light is on' and light device 'state = light is off'.
We introduce the concept of having more than one LNS Server network variable for a software device service, allowing the detection of manual switch user interactions. On any light change we compare the values given by the manual switches and light device. More than one manual switch can be connected to a single light device. After comparing we know then whether the user or the system has invoked the light change.

### 7.4.2 Software changes

Software changes concerning the light device are easy to detect since the LNS Server generates always an update on which a listener can be applied. Moreover if the light change is invoked by our software, it is even easier to detect (the source of the event is created internally). We found that using timeouts for detection of manual user changes is not an adequate way, since it is only an approximation. Using the concept of the above mentioned multiple LNS Server network variables result in a more reliable data input for the learning

task. However, what still can happen is that two invoked changes for example a user interaction and a software interaction can clash with each other. This is also recognized by our new ABI System generating a race condition event within our framework. Once more this supports the reliability of the system.

## 7.5 Chatting with devices

Instead of having some additional software for controlling light and blind devices we integrated a messenger for that purpose. The idea is that messengers are used quite often for communication anyway. Connecting from almost any messenger to our system is realizable and enables the user to send commands to the software devices directly. Security issues are solved with this concept as well as communication with our system in general. The administrator of the messenger server can add and remove users to any defined areas. Therefore no user is able to send commands to devices which do not belong to the area he is attached to. Since messengers use SSL for a secure connection we can ensure high encrypted message sending to our system. The messenger user interface is easy to manage and minimizes the administrative outlay. Commands like 'move blinds up' as well as 'move left blind up' are handled. For a complete list of all commands see A.4. The messenger service is not only a command dialog it acts also as a presence device service, see 7.2.2. It detects movements of the mouse and keystrokes of attached devices to the computer running a instant messenger.

## 7.6 Wireadmin

As described in section 6.1 we had to deal with incorrect implementation of the Wireadmin provided by Oscar OSGi [OSC] (a different implementation of the OSGi Framework than Knopflerfish OSGi [KNO]. The main problem was that the provided Wireadmin bundle did not support a service which was registered within the framework as producer and consumer. Only single service registration such as producer or consumer was supported. Moreover, the asynchronous event manager did not terminate its threads correctly, this resulted in creating more and more threads when restarting the bundle (resource exhausting). We worked hard to fix those problem because the Wireadmin is an essential service in our new ABI System. Without a correct implementation of the Wireadmin we could not create wires for communication between producer and consumer services, see also 6.1. Smaller problems not worth mentioning here were fixed during reviews of the service. Attention must be paid when importing the Wireadmin service into the framework, current fixed version is 1.0.1, the API version was not changed and is still 1.0.0, ensuring other versions of the Wireadmin service being compatible with our version.

## 7.7 Long and short term memories and reinforcement learning

Our approach of having two different memories has been implemented with two reinforcement learning instances. These two reinforcement units are connected in parallel, i.e. they have the same input given by the environment. Independent from each other they calculate the next action to take if the environment invokes a state change, see figure 7.15.

The performed action is then chosen by calculating the difference between the optimal learned policy and the q-value of the proposed action for each learning unit. The proposed action with the smaller difference wins the competition and is performed. As explained in section 2 an earlier proposed action can match with the current state. If only one learning unit has a matched action-state pair this is chosen to perform the next action. In cases where no action matches the long term memory decides what action to choose next. In figure 7.16 the situation for the long term q-values is shown and in figure 7.17 the short term q-values recorded during two days in May. Not all states have been visited yet, and though rewards have been given only to certain states and actions.

The optimal policy learnt so far is denoted in the 'Rho-Value' in figure 7.16 and 7.17. It is obvious that

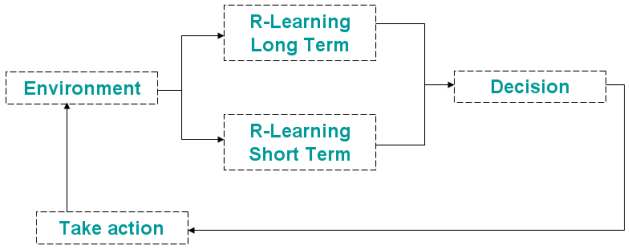


Figure 7.15: Long and short term memory overview

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| # Q-Table | | | | | | | | | | |
| # | | | | | | | | | | |
| 100 | DaylightWait(100,200) | -0.014347523 | DaylightWait(100,300) | 0 | WaitSelf(100,100) | -0.24309465 | WaitPresenceChange(100,110) | 3.397261686 | (0,0) | 0 |
| 200 | DaylightWait(200,100) | 0 | DaylightWait(200,300) | 0.338544798 | WaitSelf(200,200) | 0.098484593 | WaitPresenceChange(200,210) | 0.056678487 | (0,0) | 0 |
| 300 | DaylightWait(300,100) | 0 | DaylightWait(300,200) | -0.100007692 | WaitSelf(300,300) | -0.129093455 | WaitPresenceChange(300,310) | -0.07519182 | (0,0) | 0 |
| 110 | DaylightWait(110,210) | 0 | DaylightWait(110,310) | 0 | WaitsSlf(110,110) | 0 | WaitPresenceChange(110,100) | 0 | TurnLightOn(110,111) | 9.575474 |
| 210 | DaylightWait(210,110) | 4.043840952 | DaylightWait(210,310) | 0 | WaitSelf(210,210) | 0.462212872 | WaitPresenceChange(210,200) | 0 | (0,0) | 0 |
| 310 | DaylightWait(310,110) | 0 | DaylightWait(310,210) | 0.627617068 | WaitSelf(310,310) | 0.032875089 | WaitPresenceChange(310,300) | 0.378920899 | (0,0) | 0 |
| 101 | DaylightWait(101,201) | 0 | DaylightWait(101,301) | 0 | WaitSelf(101,101) | 0 | WaitPresenceChange(101,111) | 0 | TurnLightOff(101,100) | 7.420084 |
| 201 | DaylightWait(201,101) | 0 | DaylightWait(201,301) | 0 | WaitSelf(201,201) | 0.450962798 | WaitPresenceChange(201,211) | 0 | TurnLightOff(201,200) | 6.161261 |
| 301 | DaylightWait(301,101) | 0 | DaylightWait(301,201) | 0 | WaitSelf(301,301) | 0 | WaitPresenceChange(301,311) | 0 | TurnLightOff(301,300) | 2 |
| 111 | DaylightWait(111,211) | 0 | DaylightWait(111,311) | 0 | WaitSelf(111,111) | 0.699391378 | WaitPresenceChange(111,101) | 3.118008916 | (0,0) | 0 |
| 211 | DaylightWait(211,111) | 0 | DaylightWait(211,311) | 0 | WaitSelf(211,211) | -0.006786703 | WaitPresenceChange(211,201) | 1.872967792 | (0,0) | 0 |
| 311 | DaylightWait(311,111) | 0 | DaylightWait(311,211) | 0 | WaitSelf(311,311) | 0 | WaitPresenceChange(311,301) | 0 | TurnLightOff(311,310) | 2.752108 |
| # Rho-Value | | | | | | | | | | |
| # | | | | | | | | | | |
| rho | 2.840525744 | | | | | | | | | |
| # Date and Time | | | | | | | | | | |
| # | | | | | | | | | | |
| TimeAr | 4 May 2005 10:32:52 GMT | | | | | | | | | |

Figure 7.16: Long term memory q-table recorded in room 55.G.84_ from $2^{nd}$ May - $4^{th}$ Mai 2005.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| # Q-Table | | | | | | | | | | |
| # | | | | | | | | | | |
| 100 | DaylightWait(100,200) | 0.019969018 | DaylightWait(100,300) | 0 | WaitSelf(100,100) | -0.585471004 | WaitPresenceChange(100,110) | 0 | (0,0) | 0 |
| 200 | DaylightWait(200,100) | -0.115430802 | DaylightWait(200,300) | 0 | WaitSelf(200,200) | 0.010385982 | WaitPresenceChange(200,210) | 0.798018648 | (0,0) | 0 |
| 300 | DaylightWait(300,100) | 0 | DaylightWait(300,200) | 0 | WaitSelf(300,300) | 0.307961813 | WaitPresenceChange(300,310) | 0.276957539 | (0,0) | 0 |
| 110 | DaylightWait(110,210) | 0 | DaylightWait(110,310) | 0 | WaitsSlf(110,110) | 0 | WaitPresenceChange(110,100) | 0 | TurnLightOn(110,111) | 9.872202 |
| 210 | DaylightWait(210,110) | 2.619538185 | DaylightWait(210,310) | 0 | WaitSelf(210,210) | 0.329216072 | WaitPresenceChange(210,200) | 0.33160492 | (0,0) | 0 |
| 310 | DaylightWait(310,110) | 0 | DaylightWait(310,210) | 0 | WaitSelf(310,310) | -0.257634395 | WaitPresenceChange(310,300) | 0.824879901 | (0,0) | 0 |
| 101 | DaylightWait(101,201) | 0 | DaylightWait(101,301) | 0 | WaitSelf(101,101) | 0.516268318 | WaitPresenceChange(101,111) | 0 | TurnLightOff(101,100) | 6.783898 |
| 201 | DaylightWait(201,101) | 0 | DaylightWait(201,301) | 0 | WaitSelf(201,201) | 0 | WaitPresenceChange(201,211) | 0 | TurnLightOff(201,200) | 6.010992 |
| 301 | DaylightWait(301,101) | 0 | DaylightWait(301,201) | 0 | WaitSelf(301,301) | 0 | WaitPresenceChange(301,311) | 0 | TurnLightOff(301,300) | 2 |
| 111 | DaylightWait(111,211) | -0.015843812 | DaylightWait(111,311) | 0 | WaitSelf(111,111) | 0.642520507 | WaitPresenceChange(111,101) | 3.973939788 | (0,0) | 0 |
| 211 | DaylightWait(211,111) | 0 | DaylightWait(211,311) | 0 | WaitSelf(211,211) | -0.016643019 | WaitPresenceChange(211,201) | 1.910808628 | (0,0) | 0 |
| 311 | DaylightWait(311,111) | 0 | DaylightWait(311,211) | 0 | WaitSelf(311,311) | 0 | WaitPresenceChange(311,301) | 0 | TurnLightOff(311,310) | 2.818171 |
| # Rho-Value | | | | | | | | | | |
| # | | | | | | | | | | |
| rho | 2.562976806 | | | | | | | | | |
| # Date and Time | | | | | | | | | | |
| # | | | | | | | | | | |
| TimeAr | 4 May 2005 10:33:03 GMT | | | | | | | | | |

Figure 7.17: Short term memory q-table recorded in room 55.G.84_ from $2^{nd}$ May - $4^{th}$ Mai 2005.

the long and short term memory learning unit have different optimal policies, since only matched proposed actions lead to a reward and matches are totally independent from the long and short term learning unit. The long and short term memory described above can be reseted or replaced within different areas. We have implemented a store and load mechanism to ensure memory replacement.
The reward function used is described in the appendix section A.3. Both learning units have so far learnt to turn off the lights if nobody is present in the room, see action $a_{TurnLightOff(101,100)}$, $a_{TurnLightOff(201,200)}$ and $a_{TurnLightOff(301,300)}$.

## 7.8 Brithness estimation during night

Very low *lux* levels have been encountered during the night: lux values within the range of 30-60 *lux* have been recorded even though the lights were on in room. We therefore used an estimation as follows to validate the daylight device performance.

Assumptions:

- luminous flux 3350 [lm]
- light efficiency 50%
- room area 100 $m^2$

In the considered room 55.G.74_ 3 light rows each have 4 lamps. Thus the luminous flux is $12 * 3350[lm] = 40200[lm]$. The light efficiency is estimated to 50% → effective luminous flux:

$$luminousflux = 0.5 * 40200[lm] = 20100[lm] \tag{7.4}$$

$$brightness_{12Lamps} = \frac{20100[lm]}{100m^2} = 201[lx] \tag{7.5}$$

$$brightness_{3rows} = \frac{201[lx]}{3} = 67[lx] \tag{7.6}$$

The age of the lamps has also an influence on their luminous flux. Therefore lux values in the range of 30-40 are expected. Estimations and calculation has been provided by R. Brunner 1.4.

# Chapter 8

# Future work

The diploma thesis time is restricted to eight weeks. Six weeks were used to get the described ABI System designed implemented and tested. Most of the time in the remaining two weeks was used for writing, some minor part was used for bug fixing. The result is now a stable base for future work. There is an elaborated project setup giving the possibility to create new projects within a very short time. New created or adapted software pieces can be easily installed and uninstalled while the ABI System is running.

In this chapter we try to outline all the ideas and possible enhancements we see, having explored the OSGi Framework and implemented the ABI System.

## 8.1 Data interpretations

The ABI System is collecting a lot of raw environmental data. This data must now carefully be interpreted to gain valuable information for the adaptive building intelligence application. As the data is stored in a structured way in the MySQL database, it is very easy and fast to create different views to the data.

It must be possible to extract very interesting information, which in turn lead to ideas for new virtual device services to be implemented. We think also that trying to extract information reveals more requirements to the logged information. Hence we clearly see one future work in bringing in more logging details.

With the fuzzified daytime device one idea is already on its way. The collected data has always the time stamp associated, but this information is not yet taken into account. The daytime service implemented divides the day in different periods like morning, noon, midnight. The idea to take the data and enhance it with a new attribute denoting the fuzzified daytime. Plotting theses aggregated values is expected to show patterns describing how the inhabitants of the building are using it. The kitchen has surely some peeks for certain day times.

## 8.2 Real devices

It is clear that an ABI System is always placed in an environment with real devices and not with the complete reliable devices. A clear concept must be worked out to do machine learning within such an environment. One approach we had always in mind was to learn about the reliability of devices and take this into account during the machine learning. This has to be defined in each machine learning context. In our reinforcement learning approach this clearly would mean to have a dynamic reward function which is also dependent on the reliability.

Another idea to deal with the issue of the real devices of hardware is to improve their functionality by combining different hardware devices. To clarify this idea, let us take the example of the non functional

blinds. Some of the blinds are not usable as they do not provide a position information. Even worse if we send a up command it moves up to the top, but after hitting the topmost position it moves down to the bottom. As the stop command always worked, we had the idea to connect a USB camera to one of these ever running computers in the room. The camera could then be used as a position sensor for the blinds.

The presence detection is also a harder task than it may look at a first glance. The very best would be being able to count people which entered the room, or left the room. Hence always knowing how many are in room at that very moment. The cheap movement detectors which are playing the role of a presence sensor are far away from even reliably giving the presence information. The assumption of the producers of presence sensors, that there must be free sight to the sensor can not be hold. One enhancement to the present sensor was made by introducing the messenger bus with its messenger presence service. Another idea is to have a USB camera installed, to check the area around the door.

## 8.3 Virtual devices

In a next step we propose to expand the ideas like the busyness device to the other devices besides the presence device service.

Bringing the pictures of an USB camera into the ABI System opens the field for different refining device services:

- position of blinds
- open windows or doors
- presence in a room.
- counting people enter, leaving a room

The good thing is, that such a service can be developed completely independent from the ABI System. Once it works as a stand alone solution it is easy integrated.

Most of the computers in a room are equipped with a microphone. Like the instant messenger running on computers with people working on it, it may be possible to have a software running which is always listening through the microphone. On one hand this could bring in additional presence signal, on the other hand this could be used for speech recognition.

## 8.4 Reinforcement Learning

The reinforcement learning as it is implemented so far, lacks the possibility of dynamic states. As consequence all the states have to be statically defined before starting and remain fixed. For an adaptive building intelligence application it is a clear demand to have these states dynamic. A nice future work is to turn the presented reinforcement learning into a framework. The internet search at the beginning of this work did not bring up a java framework for reinforcement learning. There are libraries around for C and C++ but not for java.

## 8.5 New structures

The presented static structure of building, floor, and room is needed for proving the concept, and also to test and compare different learning algorithms. A project of its own should deal with the question of the structure. Taking the ABI System with its device and data base logging can be a good starting point and data source for structure discussions. It is a stable base for fast prototyping different structure approaches.

## 8.6 Extend ABI System flexibility

In the OSGi Framework are a lot more concepts described as were used in the ABI System so far. Adding the configuration management as it is specified in the OSGi Framework and implemented in the Knopflerfish OSGi would increase the flexibility of the ABI System drastically. Each of the presented bundles needs configuration information which is now fixed wired in the program code or resides in a configuration file. Enabling the configuration management as it is defined within the OSGi Framework brings in the possibility to configure the bundles during run time.

# Part IV

# Appendix, Glossary and Bibliography

# Appendix A

# Appendix

## A.1 Technologies and Software

- Java JSDK 1.4 as programming language and platform for the Knopflerfish OSGi implementation.
- LON the fieldbus technology connecting the sensors and actuators on the Instiute of Neurinformatics G floor.
- LonMaker 3 software for LON network browsing and configuration tasks
- LNS Server 3.01 server software providing access to the LON network through TCP/IP
- echelon.jar Java software library defining an API for accessing the LON network.
- lonmark.jar Java software library defining an API for accessing sensors, actuators and controller devices in the LON network.
- stdtypes.jar Java software library defining an API for the standard LON enabled devices.
- OSGi a consortium providing the platform specificaton — a service oriented technology
- Knopflerfish an open source implementation of the OSGi specification in the Java programming language.
- fuzzy.jar an open source fuzzy library
- MySQL an open source database.
- Eclipse an open source integrated development environment.
- Ant an open source Java-based build tool.
- Jive a jabber instant messenger server
- smack.jar Java library for connection to a jabber instant messenger server

## A.2 Actions defined for the reinforcement learning

In this appendix section we show the actions defined for our reinforcement learning approach.

| action | source state | target state | id |
|---|---|---|---|
| $a_{DaylightWait}$ | $s_{[1,0,0]}$ | $s_{[2,0,0]}$ | DaylightWait(100,200) |
| $a_{DaylightWait}$ | $s_{[1,0,0]}$ | $s_{[3,0,0]}$ | DaylightWait(100,300) |
| $a_{PresenceChangeWait}$ | $s_{[1,0,0]}$ | $s_{[1,1,0]}$ | PresenceChangeWait(100,110) |
| $a_{WaitSelf}$ | $s_{[1,0,0]}$ | $s_{[1,0,0]}$ | WaitSelf(100,100) |
| $a_{DaylightWait}$ | $s_{[2,0,0]}$ | $s_{[1,0,0]}$ | DaylightWait(200,100) |
| $a_{DaylightWait}$ | $s_{[2,0,0]}$ | $s_{[3,0,0]}$ | DaylightWait(200,300) |
| $a_{PresenceChangeWait}$ | $s_{[2,0,0]}$ | $s_{[2,1,0]}$ | PresenceChangeWait(200,210) |
| $a_{WaitSelf}$ | $s_{[2,0,0]}$ | $s_{[2,0,0]}$ | WaitSelf(200,200) |
| $a_{DaylightWait}$ | $s_{[3,0,0]}$ | $s_{[2,0,0]}$ | DaylightWait(300,200) |
| $a_{DaylightWait}$ | $s_{[3,0,0]}$ | $s_{[1,0,0]}$ | DaylightWait(300,100) |
| $a_{PresenceChangeWait}$ | $s_{[3,0,0]}$ | $s_{[3,1,0]}$ | PresenceChangeWait(300,310) |
| $a_{WaitSelf}$ | $s_{[3,0,0]}$ | $s_{[3,0,0]}$ | WaitSelf(300,300) |
| $a_{DaylightWait}$ | $s_{[1,1,0]}$ | $s_{[2,1,0]}$ | DaylightWait(110,210) |
| $a_{DaylightWait}$ | $s_{[1,1,0]}$ | $s_{[3,1,0]}$ | DaylightWait(110,310) |
| $a_{PresenceChangeWait}$ | $s_{[1,1,0]}$ | $s_{[1,0,0]}$ | PresenceChangeWait(110,100) |
| $a_{WaitSelf}$ | $s_{[1,1,0]}$ | $s_{[1,1,0]}$ | WaitSelf(110,110) |
| $a_{TurnLightOn}$ | $s_{[1,1,0]}$ | $s_{[1,1,1]}$ | TurnLightOn(110,111) |
| $a_{DaylightWait}$ | $s_{[2,1,0]}$ | $s_{[1,1,0]}$ | DaylightWait(210,110) |
| $a_{DaylightWait}$ | $s_{[2,1,0]}$ | $s_{[3,1,0]}$ | DaylightWait(210,310) |
| $a_{PresenceChangeWait}$ | $s_{[2,1,0]}$ | $s_{[2,0,0]}$ | PresenceChangeWait(210,200) |
| $a_{WaitSelf}$ | $s_{[2,1,0]}$ | $s_{[2,1,0]}$ | WaitSelf(210,210) |
| $a_{DaylightWait}$ | $s_{[3,1,0]}$ | $s_{[2,1,0]}$ | DaylightWait(310,210) |
| $a_{DaylightWait}$ | $s_{[3,1,0]}$ | $s_{[1,1,0]}$ | DaylightWait(310,210) |
| $a_{PresenceChangeWait}$ | $s_{[3,1,0]}$ | $s_{[3,0,0]}$ | PresenceChangeWait(310,300) |
| $a_{WaitSelf}$ | $s_{[3,1,0]}$ | $s_{[3,1,0]}$ | WaitSelf(310,310) |
| $a_{DaylightWait}$ | $s_{[1,0,1]}$ | $s_{[2,0,1]}$ | DaylightWait(101,201) |
| $a_{DaylightWait}$ | $s_{[1,0,1]}$ | $s_{[3,0,1]}$ | DaylightWait(101,301) |
| $a_{WaitSelf}$ | $s_{[1,0,1]}$ | $s_{[1,0,1]}$ | WaitSelf(101,101) |
| $a_{TurnLightOff}$ | $s_{[1,0,1]}$ | $s_{[1,0,0]}$ | TurnLightOff(101,100) |
| $a_{DaylightWait}$ | $s_{[2,0,1]}$ | $s_{[1,0,1]}$ | DaylightWait(201,101) |
| $a_{DaylightWait}$ | $s_{[2,0,1]}$ | $s_{[3,0,1]}$ | DaylightWait(201,301) |
| $a_{WaitSelf}$ | $s_{[2,0,1]}$ | $s_{[2,0,1]}$ | WaitSelf(201,201) |
| $a_{TurnLightOff}$ | $s_{[2,0,1]}$ | $s_{[2,0,0]}$ | TurnLightOff(201,200) |
| $a_{DaylightWait}$ | $s_{[3,0,1]}$ | $s_{[2,0,1]}$ | DaylightWait(301,201) |
| $a_{DaylightWait}$ | $s_{[3,0,1]}$ | $s_{[1,0,1]}$ | DaylightWait(301,101) |
| $a_{WaitSelf}$ | $s_{[3,0,1]}$ | $s_{[3,0,1]}$ | WaitSelf(301,301) |
| $a_{TurnLightOff}$ | $s_{[3,0,1]}$ | $s_{[3,0,0]}$ | TurnLightOff(301,300) |
| $a_{DaylightWait}$ | $s_{[1,1,1]}$ | $s_{[2,1,1]}$ | DaylightWait(111,211) |
| $a_{DaylightWait}$ | $s_{[1,1,1]}$ | $s_{[3,1,1]}$ | DaylightWait(111,311) |
| $a_{WaitSelf}$ | $s_{[1,1,1]}$ | $s_{[1,1,1]}$ | WaitSelf(111,111) |
| $a_{PresenceChangeWait}$ | $s_{[1,1,1]}$ | $s_{[1,0,1]}$ | PresenceChangeWait(111,101) |
| $a_{DaylightWait}$ | $s_{[2,1,1]}$ | $s_{[1,1,1]}$ | DaylightWait(211,111) |
| $a_{DaylightWait}$ | $s_{[2,1,1]}$ | $s_{[3,1,1]}$ | DaylightWait(211,311) |
| $a_{WaitSelf}$ | $s_{[2,1,1]}$ | $s_{[2,1,1]}$ | WaitSelf(211,211) |
| $a_{PresenceChangeWait}$ | $s_{[2,1,1]}$ | $s_{[2,0,1]}$ | PresenceChangeWait(211,201) |
| $a_{DaylightWait}$ | $s_{[3,1,1]}$ | $s_{[2,1,1]}$ | DaylightWait(311,211) |
| $a_{DaylightWait}$ | $s_{[3,1,1]}$ | $s_{[1,1,1]}$ | DaylightWait(311,111) |
| $a_{WaitSelf}$ | $s_{[3,1,1]}$ | $s_{[3,1,1]}$ | WaitSelf(311,311) |
| $a_{PresenceChangeWait}$ | $s_{[3,1,1]}$ | $s_{[3,0,1]}$ | PresenceChangeWait(311,301) |
| $a_{TurnLightOff}$ | $s_{[3,1,1]}$ | $s_{[3,1,0]}$ | TurnLightOff(311,310) |

Table A.1: States used for the r-learning algorithm

## A.3 Reward function defined for the reinforcement learning

In this appendix section we show the rewards defined for our reinforcement learning approach.

| state | action | reward |
|---|---|---|
| $s_{[1,0,0]}$ | WaitSelf(100,100) | 1.0 |
| $s_{[1,0,0]}$ | DaylightWait(100,200) | 1.25 |
| $s_{[1,0,0]}$ | DaylightWait(100,300) | 1.0 |
| $s_{[1,0,0]}$ | PresenceChangeWait(100,110) | 1.5 |
| $s_{[2,0,0]}$ | WaitSelf(200,200) | 1.0 |
| $s_{[2,0,0]}$ | DaylightWait(200,300) | 1.25 |
| $s_{[2,0,0]}$ | DaylightWait(200,100) | 1.25 |
| $s_{[2,0,0]}$ | PresenceChangeWait(100,110) | 1.5 |
| $s_{[3,0,0]}$ | WaitSelf(300,300) | 1.0 |
| $s_{[3,0,0]}$ | DaylightWait(300,100) | 1.25 |
| $s_{[3,0,0]}$ | DaylightWait(300,200) | 1.5 |
| $s_{[3,0,0]}$ | PresenceChangeWait(100,110) | 1.5 |
| $s_{[1,1,0]}$ | WaitSelf(110,110) | 1.00 |
| $s_{[1,1,0]}$ | DaylightWait(110,210) | 1.5 |
| $s_{[1,1,0]}$ | DaylightWait(110,310) | 1.25 |
| $s_{[1,1,0]}$ | PresenceChangeWait(110,100) | 1.5 |
| $s_{[1,1,0]}$ | TurnLightOn(110,111) | 10.0 |
| $s_{[2,1,0]}$ | WaitSelf(210,210) | 1.00 |
| $s_{[2,1,0]}$ | DaylightWait(210,110) | 1.25 |
| $s_{[2,1,0]}$ | DaylightWait(210,210) | 1.25 |
| $s_{[2,1,0]}$ | PresenceChangeWait(210,200) | 1.5 |
| $s_{[2,1,0]}$ | TurnLightOn(210,211) | 10.0 |
| $s_{[3,1,0]}$ | WaitSelf(310,310) | 1.00 |
| $s_{[3,1,0]}$ | DaylightWait(310,210) | 1.5 |
| $s_{[3,1,0]}$ | DaylightWait(310,110) | 1.25 |
| $s_{[3,1,0]}$ | PresenceChangeWait(310,300) | 1.5 |
| $s_{[1,0,1]}$ | WaitSelf(101,101) | 1.00 |
| $s_{[1,0,1]}$ | DaylightWait(101,201) | 1.5 |
| $s_{[1,0,1]}$ | DaylightWait(101,301) | 1.25 |
| $s_{[1,0,1]}$ | PresenceChangeWait(101,111) | 0.25 |
| $s_{[1,0,1]}$ | TurnLightOff(101,100) | 10.0 |
| $s_{[2,0,1]}$ | WaitSelf(201,201) | 1.00 |
| $s_{[2,0,1]}$ | DaylightWait(201,301) | 1.25 |
| $s_{[2,0,1]}$ | DaylightWait(201,101) | 1.25 |
| $s_{[2,0,1]}$ | PresenceChangeWait(201,211) | 0.25 |
| $s_{[2,0,1]}$ | TurnLightOff(201,200) | 10.0 |
| $s_{[3,0,1]}$ | WaitSelf(301,301) | 1.00 |

Table A.2: Rewards used in states $s_{[1,0,0]}$, $s_{[2,0,0]}$, $s_{[3,0,0]}$, $s_{[1,1,0]}$, $s_{[2,1,0]}$, $s_{[3,1,0]}$, $s_{[1,0,1]}$, $s_{[2,0,1]}$ and $s_{[3,0,1]}$ for the r-learning algorithm

| state | action | reward |
|---|---|---|
| $s_{[3,0,1]}$ | DaylightWait(301,201) | 1.5 |
| $s_{[3,0,1]}$ | DaylightWait(301,101) | 1.25 |
| $s_{[3,0,1]}$ | PresenceChangeWait(301,311) | 0.25 |
| $s_{[3,0,1]}$ | TurnLightOff(301,300) | 10.0 |
| $s_{[1,1,1]}$ | WaitSelf(111,111) | 1.00 |
| $s_{[1,1,1]}$ | DaylightWait(111,211) | 1.5 |
| $s_{[1,1,1]}$ | DaylightWait(111,311) | 1.25 |
| $s_{[1,1,1]}$ | PresenceChangeWait(111,101) | 2.00 |
| $s_{[2,1,1]}$ | WaitSelf(211,211) | 1.00 |
| $s_{[2,1,1]}$ | DaylightWait(211,311) | 1.25 |
| $s_{[2,1,1]}$ | DaylightWait(211,111) | 1.25 |
| $s_{[2,1,1]}$ | PresenceChangeWait(211,201) | 2.00 |
| $s_{[3,1,1]}$ | WaitSelf(311,311) | 1.00 |
| $s_{[3,1,1]}$ | DaylightWait(311,211) | 1.5 |
| $s_{[3,1,1]}$ | DaylightWait(311,111) | 1.25 |
| $s_{[3,1,1]}$ | PresenceChangeWait(311,301) | 2.00 |
| $s_{[3,1,1]}$ | TurnLightOff(311,310) | 10.0 |

Table A.3: Rewards used in states $s_{[3,0,1]}$,$s_{[1,1,1]}$, $s_{[2,1,1]}$ and $s_{[3,1,1]}$ for the r-learning algorithm

## A.4 Messenger commands

The following commands can be used for 'chatting' with the light and blind devices. Commands for the blind device:

- move up blinds
- move down blinds
- stop blinds
- move [left, middle, right] blind up
- move [left, middle, right] blind down
- stop [left, middle, right] blind

Commands for the light device:

- turn lights on
- turn lights off
- turn light [on,off] corridor
- turn light [on,off] window

Combination of of light and blind commands can be used as follows:

- turn lights off and move blinds down
- turn light off corridor and move left blind down
- 'please turn the light on immediately and move up all blinds now'!

A short description of all commands is available by typing 'help' in messenger chat box, see figure A.2 and A.1. The main window of the messenger client used is shown in figure A.3.

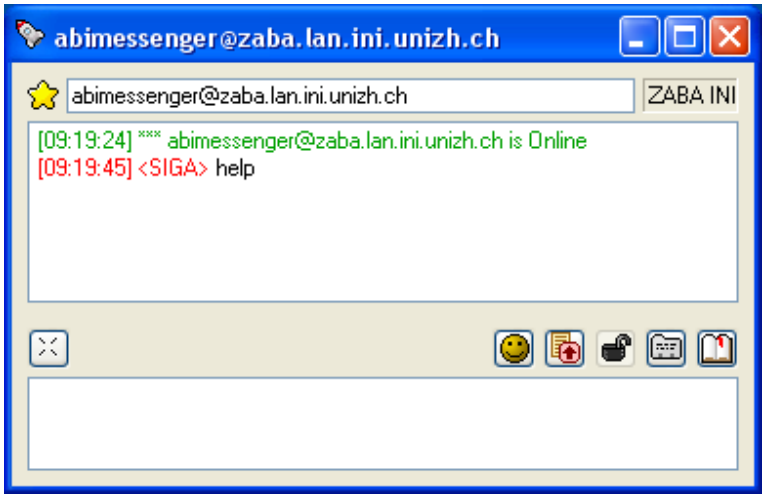


Figure A.1: Type 'help' for all available commands.

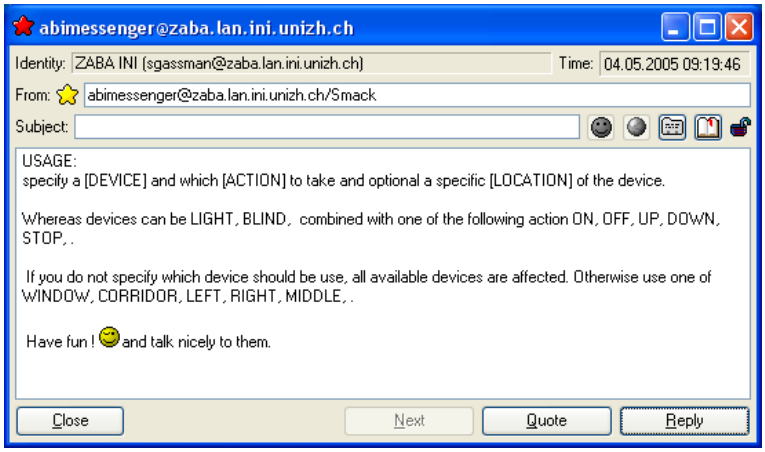


Figure A.2: Messenger help dialog


Figure A.3: Messenger main window

## A.5 ABI commands

Several administrative commands have been implemented for the ABI core system. The following list gives an overview of all available commands.

- showbuses [-help] [-f <pattern>] ... → lists all bus identifiers attached to the abi bus using the following argument:
  1. NOT YET IMPLEMENTED -f <pattern> filters for ABI bus identifiers matching the given expression

- showdevices [-help] [-f <pattern>] ... → lists all devices attached to the abi bus using the following argument:
  1. NOT YET IMPLEMENTED -f <pattern> filters for ABI bus devices identifiers matching the given expression

- connectto [-help] <busid> → connects to an attached bus, given by its <busid> using the following argument:
  1. <busid> connect to an attached bus, given by its <busid>, for example connectto lon.bus-1.0

- disconnectfrom [-help] <busid> → disconnectfroms an attached bus, given by its <busid> using the following argument:
  1. <busid> disconnect from an attached bus, given by its <busid>, for example disconnectfrom lon.bus-1.0

- regdevserv [-help] <deviceURI> → REGister DEVice as SERVice. using the following arguments:
  1. <deviceURI> a valid URI of an available device with its parameter separated by '?', for example:

     ```
     http://172.16.2.200:2540/lon.bus-1.0/DaylightService
        ?nv1=Y355G-2188\_-HTS\_ECO.nvoDaylightLux.5
        ^fqcn=ch.unizh.ini.osgi.service.device.sensor.daylight.DaylightService
     ```

- deregdevserv [-help] <deviceURI> → DEREGister DEVice as SERVice. using the following arguments:
  1. <deviceURI> a valid URI of an available device with its parameter separated by '?', for example:

     ```
     http://172.16.2.200:2540/lon.bus-1.0/DaylightService
        ?nv1=Y355G-2188\_-HTS\_ECO.nvoDaylightLux.5
        ^fqcn=ch.unizh.ini.osgi.service.device.sensor.daylight.DaylightService
     ```

- startnetmonitor [-help] <ipaddress> <port> <timeout> → start monitoring network connection availability for given host using the following arguments:
  1. <ipaddress> ip address for example: 172.16.2.200
  2. <port> port for example: 3232
  3. <timeout> check interval must be greater than 5 sec.

- stopnetmonitor [-help] <ipaddress> <port> <timeout> → stop monitoring network connection availability for given host using the following arguments:
  1. <ipaddress> ip address for example: 172.16.2.200

2. <port> port for example: 3232

- send [-help] <wirePID> <type> <value> → send a new value (command) to a device using an existing wire using the following arguments:
  1. <wirePID> wire PID for example: wire.16-14560200878
  2. <type> flavour of the wire for example: string
  3. <value> message to send, for example: TURN_LIGHT_OFF,USER

- senddevice [-help] <devicePID> <type> <value> → send a new value (command) to a device not using a existing wire using the following arguments:
  1. <devicePID> receiver PID example:

```
http://172.16.2.200:2540/lon.bus-1.0/LightService
    ?nv1=Y355G-2187_-PHI_LRC_B.nvo0102LampVal.26
    ^nv2=Y355G-2187_-PHI_LRC_B.nvi0102irSetting.14
    ^nv3=Y355G-2187_-UPL_LT_A.nvoSetting[2].2
    ^nv4=Y355G-2187_-UPL_LT_B.nvoSetting[2].2
    ^nv5=Y355G-2187_-UPL_LT_D.nvoSetting[2].2
    ^fqcn=ch.unizh.ini.osgi.service.device.actuator.light.LightService
```

  2. <type> flavour of the wire for example: string
  3. <value> message to send, for example: TURN_LIGHT_OFF,USER

## A.6 ABI area commands

Several administrative commands have been implemented for the ABI area. The following list gives an overview of all available commands.

- create [-help] <structureCategroy> <uniqueID> <location> → creates a new area services using the following arguments:
  1. <structureCategroy> a valid structure category either 'room', 'floor' and 'building'
  2. <uniqueID> area will be registered with this ID, for example 55.G.74_
  3. <location> well defined location, for example 55.G.74_

- remove [-help] <uniqueID> → removes area services using the following argument:
  1. <uniqueID> the area service has been registered with this ID

- show [-help] [-f <pattern>] ... → shows all registered area services optional argument:
  1. NOT YET IMPLEMENTED -f <pattern> filters for area identifiers matching the given expression

- printqvalues [-help] <areaID> <type> → prints all so far calculated q-values for given area using the following arguments argument:
  1. <areaID> the area service has been registered with this ID
  2. <type> either 'long' for long term memory or 'short' for short term memory.

- memsave [-help] <areaID> <filename> <type> → saves for an area the long term m., short term m. to a file using the following arguments:
  1. <areaID> the area service has been registered with this ID
  2. <filename> specifies any filename and path
  3. <type> either 'long' for long term memory or 'short' for short term memory.

- memload [-help] <areaID> <filename> <type> → loads for an area the long term m., short term m. from a file using the following arguments:
  1. <areaID> the area service has been registered with this ID
  2. <filename> specifies any filename and path
  3. <type> either 'long' for long term memory or 'short' for short term memory.

## A.7 Installation and startup

There are three mandatory installed software packages:

- Java
- Knopflerfish binary, or source distribution
- ABI System related packages must be available in jars folder of the Knopflerfish installation.

The following optional software packages are recommended:

- MySQL data base
- Jive instant messenger

The Knopflerfish framework is started by using the start script which comes with the Knopflerfish distribution. When the Knopflerfish framework is started there are several possibilities how it can be accessed. The two essential ways to interact are described here:

- Graphical user interface, with console
- Telnet console

Installing the ABI System with the graphical user interface is done by choosing the ABI System bundles with the help of the file dialog. The bundles are then installed and started. The recommended order of installing the bundles is as follows:

- `ABInetmonitor`
- `ABIutil`
- `ABIfuzzyutil`
- `ABIreinforcementlearning`
- `ABIcoreapi`
- `ABIcorecmd`
- `ABIare`
- `ABIarecmd`
- `ABIlon`
- `ABImessenger`

Installing the ABI System on the telnet console is done by typing the following on the command line:

```
install file:jars/ABInetmonitor/ABInetmonitor_all-1.0.0.jar
 file:jars/ABIutil/ABIutil-1.0.0.jar file:jars/fuzzyutil/fuzzyutil-1.0.0.jar
 file:jars/ABIcoreapi/ABIcoreapi_all-1.0.0.jar file:jars/ABIcmd/ABIcmd-1.0.0.jar
 file:jars/ABIlon/ABIlon-1.0.0.jar file:jars/ABIarea/ABIarea_all-1.0.0.jar
 file:jars/ABIareacmd/ABIareacmd-1.0.0.jar
 file:jars/reinforcmentlearningutil/reinforcmentlearningutil-1.0.0.jar
 file:jars/ABImessenger/ABImessenger-1.0.0.jar
```

Installing bundles on the command line does not start them automatically as it is the case with using the graphical user interface. In order to start the previously installed bundles the command line:

```
start "ABI Net Bundle"
 "ABI Core Bundle"
 "ABI Area Bundle"
 "ABI Area CMD Bundle"
 "ABI CMD Bundle"
 "ABI Lon Bundle"
 "ABI Messenger Bus"
```

The ABI System is now ready and it can be brought in action by entering the `ABI` command group and connecting to different buses available. Try also the `help` command or `command -help` to get specific help for a command. Check also the appendixes A.5 and A.6 for the `ABI` commands.

An example output of the `showbuses` command follows:

```
>  > enter ABI

ABI> ABI> showbuses

----=== ............... ===---
----=== Attached Buses ===---
----=== jive.bus-1.0 [attached & connected(6)]
----=== lon.bus-1.0 [attached & connected(6)]
----=== ================ ===---
----=== ================ ===---
ABI> ABI>

ABI> ABI>
```

# Appendix B

# Glossary

| Abbreviation | Explanation / Comment |
|---|---|
| ABI | Adaptive Building Intelligence |
| LNS | Lon Network Server |
| LonWorks | Field bus network protocol / standard |
| Bundles | Java JAR archive |
| Manifest | OSGi configuration file included in any Bundles |
| WireAmin | Bundle originally developed for Oscar OSGi |
| Wire | Connection object used for producer and consumer concepts |
| Service | Interface exportet by the service provider bundle |
| OSGi | Open Service Gateway initiative |
| OSCAR | Open source OSGi implementation |
| JSDK | Java JSDK 1.4 |
| Knopflerfish OSGi | Open source OSGi implementation |
| LonMaker | Software for LON network allowing browsing and configuring tasks |
| SMF | IBM Service Management Framework |
| USB | Universal Serial Bus |
| J2ME | Java 2 Platform, Micro Edition |
| CLDC | Connected Limited Device Configuration |
| SR | Service reference |
| SREG | Service registration |
| DA | Device Access |
| SIENA | Scalable Internet Event Notification Architectures |
| JMS | Java Message Service |
| ABI | Adaptive Building Intelligence |
| JAR | Java Archive |
| NV | Network Variable |
| Eclipse | Open source development environment |
| MySQL | Open source database |
| RL | Reinforcement learning |
| Jive | Instant messenger server |
| IM | Instant message |

Table B.1: Glossary.

# Bibliography


[BG04] Patrick Brunner and Simon Gassmann. Adaptive building intelligence based on the open services gateway initiative, term work. Technical report, University of Applied Sciences Rapperswil, Switzerland and Institute of Neuroinformatics, Swiss Federal Institute of Technology, Zurich, Switzerland, 2004.

[Fuz] Fuzzy Engine by Edward Sazonov. http://people.clarkson.edu/ esazonov/FuzzyEngine.htm.

[Ini03] Open Service Gateway Initiative. *OSGi Service Platform Release 3.* IOS Press, 2003. ISBN : 1-58603-311-5.

[KNO] Knopflerfish. http://www.knopflerfish.org/index.html.

[Lon] LonMark Profile. http://www.lonmark.org/products/fprofile.ht.

[Mit97] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, 1997. ISBN : 0-07-042807-7.

[OSC] OSCAR an Open Source OSGi Framework Implementation. http://oscar.objectweb.org/.

[RSS98] Andrew G. Barto Richard S. Suttom. *Reinforcement Learning.* MIT Press, 1998. ISBN : 0-262-19398-1.

[Tra02] Dirk H. Traeger. *Einfuehrung in die Fuzzy-Logik.* Teubner Verlag, 2002. ISBN : 3-519-16162-1.

[TZ04] Jonas Trindler and Raphael Zwiker. Adaptive building intelligence – parallel fuzzy controlling and learning architecture based on a temporary and long-term memory. Technical report, University of Applied Sciences Rapperswil, Switzerland and Institute of Neuroinformatics, Swiss Federal Institute of Technology, Zurich, Switzerland, 2004.

# Index