Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Master's of Computer Science

An Approach to a Sensor-Adaptive and Biomedical-Sensitive Smart Home

A master's thesis submitted in fulfillment of the requirements for the degree Master's of Computer Science

Author: Bojan Janisch
bjanis2s@smail.inf.h-brs.de

Supervisors: Prof. Dr. Gerhard K. Kraetzschmar
Prof. Dr. Karl Jonas

Date of Submission: 06-22-2016

Declaration of Authorship

I, Bojan Janisch, declare that this document titled, 'An Approach to a Sensor-Adaptive and Biomedical-Sensitive Smart Home' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this report has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the report is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ________________________________

Date: ________________________________

Abstract

One of the major drawbacks in current Smart Home solutions is, that they are not smart. A casual use-case for a Smart Home is to provide simple remote access to every sensor or actuator that is installed in the apartment, moving the switch for a light from the wall, to the mobile phone or tablet. Although people can - in best cases - integrate manually written expert rules to simulate complex management of the actuators, each time a new sensor is added to the setup, new management is wanted or a new person moves in, these rules need to be updated manually. This case gets more complicated if the fact is considered, that people may have different preferences according to their condition.

To ease the implementation and maintenance of such a Smart Home, and push research of Ambient Intelligence into well defined problems, an approach of such a sensor-adaptive and biomedical-sensitive Smart Home is proposed. The approach combines Association Rule Mining with Classification in order to learn the actuator management in a Smart Home, from the observed behavior of the inhabitants. Over time, the system learns rules from the data that can be executed by the aforementioned Smart Home software to control the installed actuators automatically.

Beside the theoretical concept, a prototype was implemented and evaluated against real data from an experimental Smart Home setup.

Contents

List of Figures

List of Tables

Abbreviations

AR	Activity Recognition	5
AGALZ	Autonomous aGent for monitoring ALZheimer patients	9
A.L.I.C.E	Artificial Linguistic Internet Computer Entity	9
ALZ-MAS	Alzheimer Multi-Agent System	9
AmI	Ambient Intelligence	1
ARM	Association Rule Mining	3
BP	Behavior Prediction	5
CCU	Central Control Unit	22
CL	Classification	3
FPM	Frequent Pattern Mining	32
GUI	Graphical User Interface	53
HRV	Heart Rate Variability	4
HUI	Human User Interface	3
IBI	Inter-beat Interval	40
ICF	International Classification of Functioning, Disability and Health	12
I.L.S.A	Independence LifeStyle Assistant	9
ISTAG	IST Advisory Group	7
ML	Machine Learning	3
POJO	Plain Old Java Object	31
RI	Rule Induction	3
SH	Smart Home	1
SPMF	Sequential Pattern Mining Framework	58

1 Introduction

1.1 Motivation

Since time immemorial people desire support in their daily life. When a train or flight is canceled they would like to be informed instantly in order to find another route to their destination. When they forget their wallet, they want to pay regardless of not having it with them. When they leave their old relatives alone at home, they want someone to look after them. People are needy in various situations during daily life. But the society is getting more and more busy. People do not have much time to care for each other while at the same time would like to have more support in their daily life. This gap is widening with the evolution of globalization and industrialization and will eventually cause a huge problem in society.

One way to solve this problem is to let computers support people's daily life. Starting with the gradual integration of computers into every object, a vision came into being of an omnipresent artificial intelligence, that cares and proactive supports people in their everyday life, being transparent to humans and adaptive to new situations. This vision is named Ambient Intelligence (AmI). Although manifested just a few decades ago, the development of AmI is already visible in society. People are supported by digital assistants who remind them of appointments, help them search for information or propose a restaurant they might like. They work in buildings with centralized and often computerized environmental control, travel in cars or trains driven by computers and can pay without cash or credit card. AmI provides many advantages in the environment it is applied to, because it is designed to support people.

One environment that is continuously growing in popularity is the domestic environment. A Smart Home (SH) enables remote control of any object that it is connected with, automatic regulations of domestic parameters like temperature, light or locks and can even optimize energy consumption of the whole household. Since people spend the most time of the day at home, the interest in SHs has boomed in the past years. As a reaction to the growing interest many manufacturers have developed solutions for SHs. This has resulted in an continuously growing variety of - more or less compatible - devices and platforms for domestic environmental management.

1.2 Problem Statement

Although advantageous for many scenarios, the installation of such a SH requires knowledge of the technology, the target areas and the way that AmI should support the daily life of the inhabitants. An expert usually configures sensors (observers) and actuators (regulators) in the SH after installation and rarely modifies it again. Nevertheless, technology and human evolve rapidly. If support in new areas of the domestic environment is wanted, new sensors or actuators need to extend the current functionalities of the SH, leading to - in best case - a configuration of the new devices or - in worst case - a reconfiguration of the whole SH system. Even if the behavior of inhabitants change, new people move in or something else changed in the domestic environment, an expert needs to reconfigure the behavior of the SH.

The problem increases in complexity if the fact is considered that people prefer different support according to their physical or mental condition. A sleepy person would not want to listen to loud music or a very stressed one would not want to be informed about everything at home. The (re-)configuration of the SH would not only require knowledge about the technology, behavior of the inhabitants and target reaction of the SH but also preferences under specific biomedical conditions from every inhabitant. An expert may be able to (re-)configure the sensors and actuators but the only ones knowing the target supportive behavior of the SH with respect to their own preferences are the inhabitants, leading to the situation that they need to (re-)configure the behavior of the SH by themselves.

However, there is not much benefit left if people trade time required to adapt the behavior of the SH every time something changes in the domestic environment, for support in their everyday life. To apply AmI in the domestic environment a SH is required that can adapt its behavior to new sensors or actuators, that learns it from the inhabitants, how to react on it and what they prefer under specific conditions.

But how could such a sensor-adaptive and biomedical-sensitive SH look like?

1.3 Solution Approach

Instead of developing a new SH, current open source solutions will be extended with the objected functionalities. In many open source SHs - like openHAB - the management of sensors and actuators - and thus the behavior of the SH system - is realized by a rule-based system. Rules are popular for complex behavior emulation because they can be read by humans and machines and - usually - do not require great knowledge of a programming language. Therefore a mechanism to adapt the behavior of the SH already exist by adapting the rules of its rule-based system.

In order to learn the behavior of an inhabitant, the first challenge is to learn rules that resemble this behavior. Rule Induction (RI), a scientific field of Machine Learning (ML) that infer knowledge from data in form of rules, can be used to infer the behavior of inhabitants by analyzing past sensor events in a SH. If an inhabitant uses an actuator, sensors related to this process would also produce events. This characteristic can be used by Association Rule Mining (ARM), a paradigm of RI, that aims to discover associations between elements in data. If a new sensor is added to the SH and it is somehow related to an actuator, ARM will discover - sooner or later - a relation between these devices.

But the behavior of the inhabitant is not purely based on events. For example, if an inhabitant enters a room, the light is not always turned on or the heating is not regulated every time. Therefore some context is required, that describes under what conditions the respective actuator is controlled by the inhabitant. This context is given by the current state of all sensors right before an actuator is activated. However, since a motion sensor in the kitchen may have nothing to do with the light switch in the bedroom, the context needs to be restricted to the related sensors that were discovered previously.

Since the context is a set of related sensor states, another RI paradigm - called Classification (CL) - can be applied in order to discriminate between the relevant and irrelevant context. The composition of both approaches generates a set of rules that resembles the actuator management of the inhabitants, supporting their daily life by taking the management of the actuators. The exported rules are stored in a performance table at the file system. The proposed learning system keeps track of the exported rules through the table in order to adapt the rule base and avoid producing the same result again, leading to the next challenge: adaptation of the knowledge.

Since a SH can be extended with different sensors and actuators over time, the learned knowledge - represented in form of rules - needs to be adapted to the new setup. However in order to adapt the rules, feedback is required that tells the SH which rules are obsolete. Direct feedback can be given by a SH specific Human User Interface (HUI) - like a website

- where the user can select wrong decisions done by the rules instantly. In addition the performance of the exported rules is evaluated periodically by the learning system which is triggered from the SH. Rules that fall below a specified performance threshold are tagged for adaptation. Instead of optimizing rules and risk to be trapped in a local optimum, the tagged rules will be removed and replaced by new learned ones. To avoid generating the exact same rule again, the removed one will be excluded from this learning process.

The last challenge is to make the SH sensitive for the physical and mental condition of the inhabitant. Since the condition of a person consists of various measurable and immeasurable features, only selected ones can be considered. Several studies proofed a relation between stress and the Heart Rate Variability (HRV), that is calculated from the inter-beats interval of the heartbeat. Many wearable devices like Smart Watches or Fitness Bands provide capabilities in measuring this data.

The main advantage of RI is that it can be applied on any data. By adding biomedical sensor devices as casual sensors to the SH, it is able to observe the behavior of the inhabitant under stressed and relaxed conditions, depending on the data available. The rule learning approach will differentiate the context and create rules for a stressed, and a relaxed inhabitant, enriching the SH with biomedical-sensitivity. Further types of biomedical data can be added the same way.

1.4 Major Results

Due to lack of time, the integration of biomedical sensors into the SH could not be realized. Therefore no evaluation concerning the biomedical-sensitivity could be performed.

Regardless of this, the proposed system learns many different highly specialized rules that could be generalized and hence reduced if clustering is applied to the input and output feature space of the CL approach. However this is not entirely necessary if a minimum dataset size threshold is applied and enough data is available. This leads to the next major result that ARM scales poorly on bigger datasets. Correlated with the dataset size, the number of discovered rules increased drastically resulting in the requirement of more computational resources.

On the other hand, the extension of association rules with descriptions learned from a classifier is able to generate very powerful decision rules and that they - as a set - can resemble the inhabitants behavior in controlling the SH actuators. In addition, by using a specialized approach for segmentation of the main event stream, the proposed system is surprisingly robust against varying sensor setups. In addition, due to its modular structure, it can be applied - with only a few changes in the configuration - on any SH.

Last but not least, the rules that were learned achieved very good results at the evaluation data, but most of them made semantically little sense. This happened because the data - collected during an experiment - represented just a peak of the real feature space. Therefore the exported rules should still be supervised by an expert or the inhabitant before they are integrated into the SH.

1.5 Structure of the Thesis

The document is structured into seven sections. After this introduction, the current state of the art in AmI is addressed in Section 2 opening with the definition of AmI and elaborating several approaches for Activity Recognition (AR) and Behavior Prediction (BP) in general and after their reusability for the problem statement. After that, some context-based background knowledge on SHs, rule-based systems, RI and the HRV is provided in Section 3. Following the flow, Section 4 revisits the learning problem in more detail, followed by a description of the proposed method for rule learning and adaptation. In Section 5 the implementation of the approach is presented in addition to the tools and libraries that were used. Subsequent to this, the proposed system is evaluated in Section 6 starting with a description of the experimental SH setup which is followed by an inter-algorithm evaluation in order to determine a reliable configuration under which the system learned the rules, before discussing the performance evaluation with respect the accuracy and convergence. Finally, this thesis concludes with Section 7 at which the problem is revisited first before the proposed system and respective results are summarized and future work is presented.

2 State of the Art

In this section, the current state of the art in AmI (2.1) and ML (2.2) is elaborated. However, since AmI is integrated in various fields of research, the focus is laying on the context of Home Automation. In addition, the concept of the elaborated approaches for AR and BP from Section 2.1.3 are evaluated from a ML perspective to not only differentiate the state of the art from the proposed concept, but to also highlight advantages of the respective approach and avoid similar drawbacks.

2.1 Ambient Intelligence - An Overview

To convey the fundamental notion of AmI, its definition and vision are elaborated in Section 2.1.1. Followed by this, several projects of various application areas for AmI are introduced in Section 2.1.2. Finally in Section 2.1.3, multiple approaches of AR and BP for the relevant area of application - the Smart Home - are presented and evaluated after their reusability.

2.1.1 The Vision of Ambient Intelligence

The term *Ambient Intelligence* refers to many slightly different definitions, depending on the context it is elaborated for and the author's scientific background. Simon Elias Bibri [41, pages 33-34] points out several definitions which were published after the reports concerning AmI from the IST Advisory Group (ISTAG) in 2001 [49] and 2003 [56].

These definitions share common characteristics like sensitivity, responsiveness, adaptivity, transparency, ubiquitousness and intelligence that an AmI system should have, which was also confirmed by Cook et al. in [45].

Rather than creating a new definition of AmI, the description of the ISTAG from 2001 will be used that defined AmI as

> *a vision of the Information Society where the emphasis is on greater user-friendliness, more efficient services support, user-empowerment, and support for human interactions. People are surrounded by intelligent intuitive interfaces that are embedded*

> *in all kinds of objects and an environment that is capable of recognizing and responding to the presence of different individuals in a seamless, unobtrusive and often invisible way* [49][P. 1].

This vision describes a supportive, responsive, adaptive, user-centered and transparent environment in which the society interacts with computers that are woven 'into the fabric of everyday life' like Mark Weiser described in [35]. However, Weiser did not describe AmI in 1991 but Ubiquitous Computing, which is a vision of the omnipresent computer, embedded invisibly into every object, connected to each other and intuitively usable by people. Therefore it is not surprising that AmI is often used synonymously for Ubiquitous Computing. Rather than equation, the notion of AmI builds upon it with a focus on human-centering, responding, adapting and intelligent supporting approaches in the daily routine [45] of people.

As a consequence of the aforementioned characteristics, AmI also interacts with technologies and artificial intelligence. In order to observe both, the environment and the people in it, sensors are used that are able to capture features of the environment. Artificial intelligence is used to analyze the observed situation and actuators are controlled to response like an intelligent entity [73]. To analyze such situations, not only abilities to identify and recognize entities in the environment - like a person or a bed - but also the determination of their current or future state - like an activity the person is currently doing or going to do - are required. The analysis of context, well elaborated by Brooks in [6], is an ongoing state of the art challenge in AmI and elaborated further in the following sections.

2.1.2 Areas of Application for Ambient Intelligence

Due to the technical evolution, AmI encompasses not only many fields of research but also many areas of application. In this section several representative projects in different areas of application are introduced in order to obtain the current status of AmI in the society.

Corchado et al. developed in 2007 an *autonomous deliberative case based planner agent designed to plan the nurses' working time dynamically, to maintain the standard working reports about the nurses' activities, and to guarantee that the patients assigned to the* *nurses are given the right care* [12, p. 1], called Autonomous aGent for monitoring ALZheimer patients (AGALZ). AGALZ is part of the Alzheimer Multi-Agent System (ALZ-MAS) [11, p. 382] by providing agents for planning and tracking of the patients. To guarantee their safety and optimize the working plan of the nurses, they are tracked by passive RFID microchips mounted on bracelets and worn on the wrist or ankle, and active RFID sensors that are installed at protected zones with an adjustable capture range of two meters.

Another agent based system - called the Independence LifeStyle Assistant (I.L.S.A) - was developed by Haigh et al. as a field study from 2001 to 2004 [20, 21]. Similar to AGALZ, I.L.S.A was a monitoring and supporting system to help elderly people maintain their independent daily routine and reducing the burden of caregivers. However compared to AGALZ, I.L.S.A was installed in the homes of four system engineers and eleven elderly people. Each home had four to seven sensors, including one medication caddy and several motion detectors.

Beside applications for health care, Augello et al. developed in 2006 a pervasive user-friendly information guide for the Museo Archeologico Regionale of Agrigento [3]. Using the Artificial Linguistic Internet Computer Entity (A.L.I.C.E) - an agent-based chatbot - as backbone, the system provides natural language communication between visitors and the information guide by transforming input speech to text, passing the text to the chatbot and transforming the textual answer to speech. Beside natural language, the conversion of the museum tour is also triggered by active RFID tags placed on PDAs and passive RFID tags near the museums attractions.

Another AmI system in the tourism application area is DALICA developed by Costantini et al. in 2007 for the Villa Adriana near Tivoli[13]. DALICA is a multi agent system where the agents - developed in the programming language DALI - proactive learn or enhance user profiles. They assist the visitors during their tour, educe their habits and preferences and propose cultural assets to them with respect to the learned profiles. Unlike the previous system, DALICA does not rely on RFID devices but traces visitors through the Galileo satellite and a receiver that is embedded in their PDAs.

Aside from health care and tourism, AmI also encompasses the area of work. In this area, Marreiros et al. developed a multi-agent based decision support system for spatial and temporal distributed discussion meetings [27]. The agents are used to analyze the emotions of the participants and imitate them to simulate the progress of the discussion. Each participant has access to an agent based simulation tool for group decision that was developed under the ArgEmotionsAgent project from 2005 to 2008 [70], which simulates the predicted reactions of other participants.

Another example of AmI in work is Sparse, an intelligent alarm processor that helps operators at power system control centers to rapidly interpret an enormous amount of alarm messages during incidents. Sparse was developed by Vale et al. in 1997 [78]. Unlike all projects described previously, Sparse follows a rule based approach with manually written expert rules. The system is divided into two parts: a time-tag logging software that sends all messages from up to two minutes ago if the beginning of an incident is detected, and the rule engine that interprets these messages and supports the operators with decision advice.

Finally in the relevant area of application for this work - the home automation area - many AmI systems were realized in multiple projects. Abowd et al. elaborated awareness of human activity in the *Aware Home* in 2002 [1]. The Aware Home is a homelike furnished laboratory at the Georgia Institute of Technology. Awareness of human activity is provided by detection of location and orientation, AR and AR. The location is determined through active RFID tags on floor mats that are distributed throughout the house and passive RFID tags that are below the knee of the inhabitants, e.g. in shoes or trousers. In addition, an unobtrusive camera grid is installed on the first floor of the laboratory, which is calibrated to the floor plan.

For research purposes in home automation, the *PlaceLab* was developed by the MIT House_n Consortium in collaboration with TIAX LLC in 2004. The PlaceLab was also an apartment-like furnished laboratory equipped with too many different sensors to describe for this work. A brief introduction of the sensor setup can be found at [60]. The PlaceLab was conceptualized to be easy expandable with further sensors or different sensor setups, depending on the current project requirements. The research agenda for the PlaceLab encompassed approaches for a proactive support of healthy behavior, AR, biometric monitoring and many more.

Although the projects described above were implemented successfully, each development was specialized, objected mostly just evaluation purposes and documented poorly. In order to gather more information about AmI at home, further state of the art publications regarding the main challenges - AR and AR - are elaborated in the following section.

2.1.3 Ambient Intelligence in Home Area: Smart Homes

To apply the aforementioned notion of AmI in the domestic area, sensors that are able to capture the area and actuators through that it can interact with it, are required. The application of AmI to this area provides several benefits like: an increased safety, more comfort and an efficient economical management [40]. However, the grade of support in this area - provided by AmI - depends strongly on the installed technology. A motion sensor for example can only detect inhabitants if they are moving. A person sitting on the couch and reading a book would not be noticed by it. Due to this it could confuse the absence of motion with an absent inhabitant and turn the light off. To compensate such constraints, the situation needs to be analyzed by the artificial intelligence thoroughly, before the decision is applied. Therefore the success of AmI is not only related to the available technologies, but also to the intelligence that provides the capabilities to support the inhabitants during their daily life.

Considering Friedewald et al., the housing application field of AmI can be divided into four functional categories: Home Automation, Communication / Socialization, Refreshing / Entertainment and Working / Learning [50]. While the last three are definitely important aspects of AmI in the domestic area, this work focuses on Home Automation also called Smart Home. According to them, a SH encompasses the support of basic housework, security and safety of the inhabitants to increase their autonomy and support their independent living. For these purposes, the SH needs to evaluate the current situation of the home. This includes not only the states of every domestic feature that may be influenced by unknown parameters, but mainly the objection to recognize the current and subsequent activities of the inhabitants.

So far, researchers approached the last challenge by assigning several observed features from an inhabitant during his casual lifestyle, to a predefined activity like walking, sleeping or cooking. This process is referred to as Activity Recognition. Taking this analysis one step further, scientists try to predict future activities of inhabitants based on probability theory that is applied to past sequences of activities, called Behavior Prediction. In the following, several approaches for both tasks are elaborated.

Activity Recognition

Following Bourobou and Yoo in [43], the recognition of activities encompasses the challenge to identify activity patterns from heterogeneous data and classify them to their respective activities. In this context, an activity pattern is defined as a common occurrence of data points from a sensor setup, that can be observed during the execution of the respective activity. Over the last decades many different approaches in AR were published. In this work, just a few distinct will be introduced to indicate the possibilities of AR.

Chen et al. proposed in [44] an ontological approach for AR based on the International Classification of Functioning, Disability and Health (ICF) standard of the World Health Organization [32]. An activity model is constructed from data in four steps: Ontological Activity Modeling, Activity Recognition, Unknown Activity Learning and Activity Model Adaption. At first a hierarchical initial ontology model for activities is created. Based on this model, a decision tree is generated using an ID3 similar algorithm to classify activities. After that, the most specific concept describing the current activity is retrieved. If the activity model does not contain a concept with the same specific properties yet, the next superior concept is checked until one is found that matches the target properties. In order to extend their ontology model, Chen et al. clustered unknown sensor property sets (or unknown activities) into clusters with similar sensor activations. To decide which of these unknown activities shall be learned next, they used a frequency threshold. Followed by this, the ontology model is updated manually or semi-automatically. In semi-automatic cases, the user is supported by a recommendation algorithm, which could for example recommend a location for an activity. Last but not least, the modified ontology model needs to be validated by an expert.

Bouchard et al. proposed in [42] a spatial-based AR approach. To determine an activity using only the spatial features of objects, their spatial relations are discovered first by determining the location of RFID tags - attached to each object - using the trilateration measurement method. For each location, one of three possible regions (small , medium, large) is calculated as circles around the center of the object. After that, a spatial reasoning approach that was proposed by Egenhofer and Franzosa in 1991 [15], is applied over the regions to determine one of eight different spatial relations. Based on the discovered relations, they inferred activities by matchmaking the plausibility - derived from a proportionality function - of spatiotemporal features with normalized plausibility values.

Another approach - proposed by Yang and Cho in [79] - utilizes a fuzzy Bayesian Network, that takes heterogeneous data from accelerometers and physiological sensors to recognize activities. The network is trained using preprocessed data from every sensor paired with a simultaneous annotated activity label from the user. Since continuous data is generated by the physiological sensor, it needs to be segmented by a fuzzy membership function that is based on the mean and standard deviation of the distribution for every sensory measurement. Using the preprocessed data and the labels, the fuzzy Bayesian Network trains a conditional probability table that can be used for AR.

Behavior Prediction ...

Beside AR, another interesting area of research in SHs is BP. Related to activities, the behavior of a person can be seen as a common sequence of actions that is executed by the person in a specific context like a morning routine or a working plan [52]. BP aims to predict the human behavior by generating a probability model of the observed action schedule from the target person.

Several different prediction algorithms - that were proposed by Cook et al. in [47] - are briefly summarized in the following paragraphs, starting with the Smart Home Inhabitant Prediction (SHIP) algorithm. In order to predict the behavior of inhabitants, sequential matching of events is applied on a collection of past sequences that the authors defined as history. The whole algorithm operates in two steps: At first, a function $l_t(s, a)$ is computed that returns the length of the longest sequence at time t for state s that ends with action a. In addition to this, a frequency measure $f(s, a)$ is calculated, that represents how many times action a has been chosen as the most probable action in state s. In the second step for every action a, a correlation between the longest sequence and frequency of action a is evaluated. SHIP returns the action a with the greatest correlated value as most probable next action. While the meaning of t and a can be derived by their respective names, a definition of state s was not given by the authors. Due to the description and usage of the term state, it is assumed that the algorithm uses a Markov-Model-like approach in order to match the sequences.

The second algorithm - called Active LeZi - is based on the LZ78 text compression algorithm [18]. In fact, the LZ78 algorithm is enhanced by a sliding window, that is sized by the longest sequence (k) seen so far, in order to generate an order-(k-1) Markov model. For prediction, the generated order-(k-1) Markov model is utilized by a predictor of the Predict by Partial Matching (PPM) family. In PPM, the probabilities of all ordered Markov models from order-1 to order-(k-1) are accumulated and the action with the highest probability is returned.

The last prediction algorithm that they proposed in [47], utilizes a task-based Markov model to predict high level tasks in action sequences of inhabitants. In their approach, the event stream is partitioned into individual tasks, based on fixed thresholds for sensor, location and time. Followed by this, a k-means clustering algorithm is applied onto the set of tasks to cluster similar tasks using features like number of actions, the starting time and duration, the locations visited and devices acted upon. For prediction they trained a hidden Markov model, which transition probability is additionally seeded from the clustered tasks.

Alam et al. proposed in 2012 a **S**equence **P**rediction algorithm via **E**nhanced **E**pisode **D**iscovery (SPEED) [38], to predict the actions of inhabitants in SHs. Based on the assumption that inhabitants generate repeating patterns of activities over the day, the continuous event stream is first split into distinct sequences, called episodes. Each episode is embraced by two complementary events, exemplary referred to as ON and OFF. After that, by using the maximum episode length of all discovered episodes as window size, a decision tree is generated that is based on all subsequences in the respective window. In addition, the frequency of occurrence for the respective subsequence is added to each node of the decision tree. Similar to Active LeZi, a PPM predictor is applied on the decision tree, to predict the next action.

... and Behavior Learning

While AR and BP are very important tasks in AmI, Guralnik and Haigh challenged the task in between by proposing an approach to learn a behavior model for the aforementioned agent-based system I.L.S.A, using sequential pattern mining [52]. This task does not only include AR, it can also be used for BP or anomaly detection. Since this approach differs from the previous elaborated tasks in the objective, that neither activities should be recognized nor behavior predicted, it is structured separately as last subsection in the state of the art of AmI.

In order to generate a model of the human behavior, high context information - when an activity starts and ends - is required that can not be retrieved directly from the sensory data. Therefore, a time region for every sensor is calculated, using the local minima of a probability density function - that is applied to each sensor - as separator, in order to segment the main event stream into several subsequences, suitable for sequential pattern mining. Afterwards, the sequences - segmented from the event stream - are stored in a database. Guralnik and Haigh defined a sequential pattern as frequent if its occurrence in the database is greater than a user specified threshold, called minimum support. For frequent sequential pattern mining they referred to several interchangeable algorithms. However, since the algorithms do not validate the discovered sequences further, Guralnik and Haigh applied three different filters during postprocessing - filtering for time regions, pattern redundancy and sensor firing repetitions - to reduce the number of interesting patterns. The final set of frequent sequential pattern is used as target human behavior model.

Several approaches for AR and BP were elaborated, followed by a similar but still different approach for behavior model learning. The approaches in AR highlight two general disadvantages: The incomparability due to different data or data-specific algorithms, and the strong sensor attachment in each approach making it impossible to reevaluate the experiment or even implement it in a common SH. On the other hand, the elaborated algorithms for BP and behavior learning mainly differ in the approach that segments the event stream. While most of the authors tried to discover regularities in the data using fixed thresholds or statistical methods, Alam et al. shined with a logical and easy reproducible approach that can be applied in common SHs.

2.2 Machine Learning - Focused on Ambient Intelligence

Machine Learning - as a field of research - encompasses the challenge to program computers to infer new knowledge based on previous experience [2]. The field is continuously growing in applications and approaches of learning. Since the background of AmI is Artificial Intelligence, which uses ML to adapt knowledge over time, a brief state of the art in ML is given. Because the objective of this thesis is to generate rules from sensory data of SHs, the state of the art is focused on approaches that aim to learn rules from data. Such approaches, that learn solely from data without supervision, belong to the unsupervised ML paradigm.

Section 2.2.1 discusses several state of the art unsupervised rule learning approaches. Afterwards Section 2.2.2 elaborates the in Section 2.1.3 introduced approaches for AR and BP from a ML perspective.

2.2.1 Approaches to learn rules

Rules can be derived from various different learning paradigms. In general, each paradigm that discovers knowledge can be used to derive rules from it. However, since a state of the art encompassing all learning paradigms in ML is out of scope for this thesis, only a few selected are elaborated that are - in general - applicable for the objected task to learn rules.

In [8, 24, 34] rules are extracted from results of clustering and applied for CL. Chiu [8] extracted rules from subtractive clustering results to classify pattern. After the data is clustered, rules are generated that compare new input data with the clustered one. If similarity - given by a membership function - of new input data exists, it is mapped to the corresponding cluster. Another approach is followed by Hruschka and Ebecken [24] who proposed a rule learning approach to describe an artificial neural network and improve the human understanding of it. They first trained an artificial neural network. Then a clustering genetic algorithm is applied to find clusters of hidden units in the trained network. Afterwards, rules are generated by describing the properties of each cluster. Similar to Hruschka's and Ebecken's approach, Setiono [34] extracted rules by clustering the activation values of the hidden units - from a trained neural network - into a small set of clusters. If the number of inputs - connected to a hidden unit - is small enough - defined by a user specified threshold - a rule will be created from the hidden unit, else it will be split into several sub units according to the clusters discovered previously. The previous two steps are repeated until rules for all hidden units are extracted. Finally Setiono mapped the input data to the correct output data by merging the rules.

Another common approach to learn rules is to extract them from a decision tree by simply traversing the path from the root to the leaf. Decision trees are popular approaches for CL due to their simplicity and efficiency. Although CL is generally applied on supervised learning problems, Karakos et al. challenged in [58] an unsupervised approach for decision tree generation by using clustering as decision criteria. Instead of classifying a feature at each node of the decision tree, they successfully applied an optimization algorithm on clustering to split the features according to the clusters they are in.

It is also common to induct rules directly. Several approaches for RI were proposed over the past decades. Clark and Niblett proposed in 1989 one of the first rule induction algorithms - called CN2 [9] - that learns class descriptions from given examples. The algorithm combines the efficiency and noise robustness from the ID3 [33], an algorithm that generates decision trees from data, together with the flexible search strategy of the AQ algorithm family [28]. In 2007 Reynolds and de la Iglesia proposed how Evolutionary Algorithms are applied to the field of RI [71]. By representing a rule or a set of rules as a tree - as described above - they successfully evolved the populations of different trees using Multi-Objective Genetic Programming to discover simple rules, that describe classes of interests.

Last but not least, Association Rule Mining - a branch of RI - aims to discover relations between elements in data. ARM is very well elaborated and popular compared to other rule learning approaches, because it does not require labeled data, follows the unsupervised learning paradigm and most important it is simple and comprehensible. According to Hipp et al. who surveyed several ARM algorithms in [54], the most known algorithm is Apriori, proposed by Agrawal et al. in [36]. The algorithm was first introduced to discover relations in context of shopping behavior. For example if a person buys product A and product B it is very likely that he buys product C too. Although the algorithm discovers association rules reliably, Apriori is very inefficient since it needs to scan the database multiple times.

From the elaborated approaches any one could be applied in this thesis to learn behavior rules for the SH management. However, most suited is ARM because it does not need any special data structure, it can be used most reasonable to discover relations between sensors and most important it discovers rules directly and does not need to extract them from another data structure. Therefore Apriori and its entitled successor - FP-Growth - are elaborated in more detail in Section 3.3.1.

2.2.2 Ambient Intelligence from a Machine Learning perspective

ML is in general parted into three different learning paradigms. Beside the unsupervised approach that learn knowledge from data, supervised and reinforcement learning are common paradigms used in various applications. Supervised learning is usually applied for classification tasks and learns by supervision. Supervision is usually given by previous classified data that is compared to the own classification result. A classifier learns to classify by minimizing the error between the own and the supervised classification. After this training phase, the classifier is applied to unknown data. Reinforcement Learning follows a completely different paradigm. Whilst supervised approaches learn from a giving function and unsupervised approaches revert engineers the function from the data, reinforcement learning actively explores the best function according to its quality of solution. Although this learning paradigm does not have a supervisor to learn from errors, a function is given that values the solution found with respect to the solution space and tries to optimize the respective function by searching for better solutions.

Chen et al. proposed in [44] an approach that follows the idea of Evolutionary Algorithms. By alternating ontology model learning and activity classification, they proposed an algorithm that evolves itself to optimize AR. This approach follows the reinforcement learning paradigm with an initial seed - gained by experts - to solve the problem of a “cold start”. Activity recognition is achieved by a pattern recognition algorithm, that uses the ontology model as knowledge to classify a sensor data stream into known and unknown activities. Unknown activities are then valued to the current ontology model and if exceeding a threshold adapted to the ontology model and thus reinforce the current solution. Missing an exact description of the sensors, by reading their publication, it is assumed that they used RFID tagged objects to generate events. Although adaptive in their sensor space, their approach - on the same time - lacks the possibility to expand it. In addition the initial seed requires prior knowledge of the target activity model, which is - at least in fresh installed SHs - not always given or known.

An unsupervised approach for AR was proposed by Bouchard et al. in [42]. By attaching labeled RFID tags to casual objects in an apartment, they were able to induce semantic to activities by tracking the location of every tagged object. To discover relations between these objects and infer activities from the relations, they applied - for comparative purposes - two ARM approaches, the Apriori and Generalized Sequential Pattern. The relations are mapped to activities according to statistical functions for plausibility and normalization, that are applied at the objects in the relation. Every sensory data is assigned to an activity using a calculated spatiotemporal relation. Bouchard et al. already stated that their trilateration algorithm for determining spatial relations has poor accuracy regarding the real position of the objects. Beside the missing application in common SHs, another disadvantage of their approach is the necessity to maintain a dictionary containing the semantic of every RFID tagged object.

In the final elaboration for AR, Yang and Cho approached the task in [79] from a supervised learning paradigm. They used manually annotated sensor stream data in order to train a Bayesian network. The input data consisted of various different sensors (accelerometers, biomedical sensors, etc.) that were used to classify the signal stream into the annotated activities. Although capable of handling a heterogeneous sensor space - especially biomedical data - the algorithm lacks adaptation since the Bayesian network requires a defined input space. A change in input space requires new training of the Bayesian network, resulting again in the requirement of annotated data in the new sensor space.

As before mentioned, BP is usually applied on the results of AR and does not require the approach itself. Hence it will be elaborated separately from AR in the following.

The approach SPEED proposed by Alam et al. in [38] follows the unsupervised learning paradigm, since its whole prediction model is based on stochastic calculations applied on a segmented sequential event stream. Every event in a subsequence - called episode - is used to calculate the most probable subsequent events. As before mentioned, a sequence is defined by two complementary events, such as ON and OFF. The resulted probability matrix can be used to predict the next event in real time. The approach - although unsupervised - requires pure binary event data that is not always given, especially in a SH with climate management.

In a similar fashion the approaches proposed by Cook et al. in [47] follow the unsupervised learning paradigm by using statistical and probability theory as background. They already stated that their SHIP algorithm stores the whole event history in memory in order to calculate the most probable next event and that their results on real data were terribly bad compared to synthetic data. Their second algorithm - Active LeZi - uses a order-k-1 Markov model and PPM to predict the next most probable event. The drawback of this approach is that current events are not taken into the prediction. To compensate all previous drawbacks, they proposed the task-based Markov model. This approach utilizes - as labeled - a hidden Markov model and in addition a k-means clustering to improve its training. Although this approach may be able to predict the next most probable task reliably, it requires fixed thresholds to divide the event or action stream into subsequences of different tasks, requiring prior knowledge.

While the previous approaches predicted only human behavior with no further application, Guralnik and Haigh learned a model in [52] that resembles the behavior of an inhabitant. Although differing from BP, the proposed approach follows the paradigm of unsupervised learning by using a sequential pattern mining algorithm to determine the frequent activities for the behavior model. Sequences for the pattern mining algorithm are gathered by segmenting the event stream based on the distribution of sensor firings over the day. On each local minima of a probability density function - that is applied on every sensor - the event stream is split. In addition they biased the system by relating higher semantics of activities to the time they were executed. In this case a meal between 8 AM and 10 AM would be interpreted as breakfast. Although the proposed approach of Guralnik and Haigh is abstract enough to be applied at any SH, the learned behavior model is based on the assumption that humans does everything, everyday in the same manner. Since this theory - although imaginable - does not reflect the reality, this approach is not suited for the objected system.

It can be summarized that the researched approaches for AR lacking a fundamental requirement for AmI, the capability of self-adapting the environmental input space. In addition the approaches for BP require a mechanism to reliably segment the event stream after objected activities. The proposed system in this thesis is differentiated from the previous elaborated approaches in such way, that it does not only adapt its feature space by the sensors installed at the SH, but also segments the event stream in a logical and unique manner.

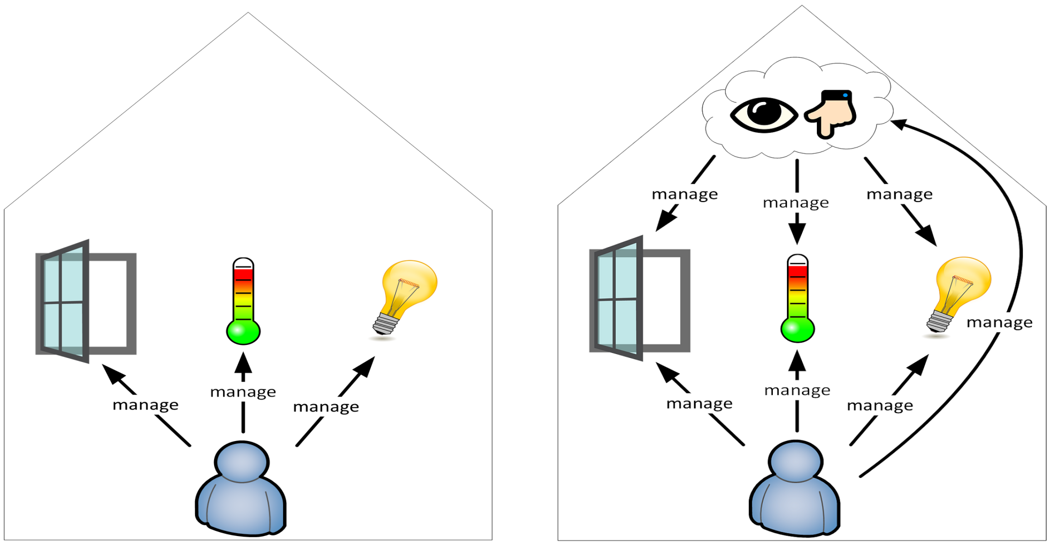
3 Project Context and Background

To comprehend the decisions made for the proposed rule learning concept, this section is concerned about topics in the context of the project. Starting with the area of application, Section 3.1 highlights the difference between a common home and a SH, elaborates openHAB - the utilized SH software - and introduces other SH solutions at that the proposed system could be applied to. In addition, since the main objective is to learn rules for the rule-based system of a SH, its general concept is elaborated in Section 3.2. After the area of application is understood, background knowledge about Rule Induction is provided in Section 3.3. Last but not least, in order to comprehend how biomedical data can be integrated into the learning system, different biomedical data types are introduced in Section 3.4 including an example using the HRV.

3.1 Initial Situation - The common Smart Home

To embrace all variations of SHs, the concept of controlling environmental features in an apartment is introduced in a setup and software independent way. Figure 1 outlines the main differences between a common home and its SH realization.

In a common home (see Fig. 1a), domestic environmental features like temperature, light or the states of windows are observed and regulated by the inhabitants. A SH (see



(A) A casual home interaction example. (B) The same example as a smart home.

FIGURE 1: The comparison of a common home and its Smart Home implementation.

Figure 1b) expands the casual home in such a way, that those environmental features are observed and regulated by a program - using sensors and actuators - while at the same time, providing remote access to regulate the domestic environment manually by the user. It does this by establishing a communication network of various sensors and actuators, that communicate among themselves or with a central station called Central Control Unit (CCU). These types of communication are referred to as centralized and decentralized.

In decentralized communication the devices communicate directly between each other. Due to this, a setup without a CCU can be realized. This increases the failure-resistance of the overall SH system, because the failure of one communication channel does not affect sensors or actuators of different channels. However, direct communication complicates the installation of complex environmental regulation, because the only usable data an actuator gets, is from directly connected sensors. Therefore, decentralized communication is usually applied in simple scenarios, like a brightness sensor that controls a roller shutter or a light switch triggered by a motion sensor.

The other approach, centralized communication uses a CCU through which every sensor and actuator communicates with. Usually the SH base analyzes its central information bus and manages all actuators according to a control logic. Popular SH solutions - like openHAB or Home Assistant - realized this control logic in form of rules. By using rules, the inhabitant is provided with a tool to describe scenarios in which an actuator should behave in a specific way. These rules are then checked by the CCU, and executed if such a scenario occurred in its information bus. However the drawback of this communication type is striking, if the CCU fails, all logic and every centralized functionality breaks down too.

Beside the communication type, the amount of supported protocols that are used by the sensors and actuators for communication, is another important aspect of a SH. Over the years various partly incompatible protocols were developed that provide different functionalities for communication between sensor or actuator devices in SHs. For a SH this is restrictive, because a device usually supports only one protocol, making the SH subjected to one device manufacturer. A common way to avoid this restriction, is the usage of a communication device plugged into the CCU through that it - if it is supported by the software - can communicate with devices using different protocols.

Since a qualitative state of the art overview of all communication protocols used in the SH context is missing, an elaboration of these protocols is not given.

The last aspect that should be noted is, whether the data is processed through the Internet into a cloud or locally through a CCU and stored in a database. In general, a SH exists local at the apartment and manages only the domestic environment. However, more and more SH cloud solutions are proposed where the devices may be installed at the apartment, but the logic and intelligence of the SH is applied on servers of the manufacturer. Since all data collected by the sensors are forwarded directly to the servers of the manufacturer, the inhabitant needs Internet access not only to control the domestic environment remotely, but also to even install the SH. The main advantage of such cloud solutions - at least for the inhabitant - is convenience, because the maintenance of the CCU is delegated to the manufacturer.

To finalize this section, many SH solutions were developed that differ in communication type, supported protocols, control logic and also in range of the data processing. From all solutions, openHAB stands out because it is an open source SH software that is completely written in Java, easy extensible, supports a broad variety of devices and runs autonomously on a local computer. Therefore in Section 3.1.1 openHAB is elaborated in more detail. For the sake of completeness, a few other SH solutions are introduced briefly in Section 3.1.2.

3.1.1 openHAB

openHAB (open Home Automation Bus) [63], continuously developed since 2009 by Kreuzer [59], is a platform independent home automation software built on Eclipse Equinox and completely written in Java (Fig. 2). Eclipse Equinox is an implementation of the OSGi (Open Services Gateway initiative) core framework specification [69], that allows management - like starting, stopping or managing communication - of modules and services (called bundles) during runtime. openHAB seizes this concept of dynamic bundle integration to change its functionalities variably during runtime.

The openHAB software architecture is composed of three different types of modules: The OSGi framework implementation Equinox, the core components of openHAB and the openHAB add-ons. Since openHAB is still under development, a fix point in the

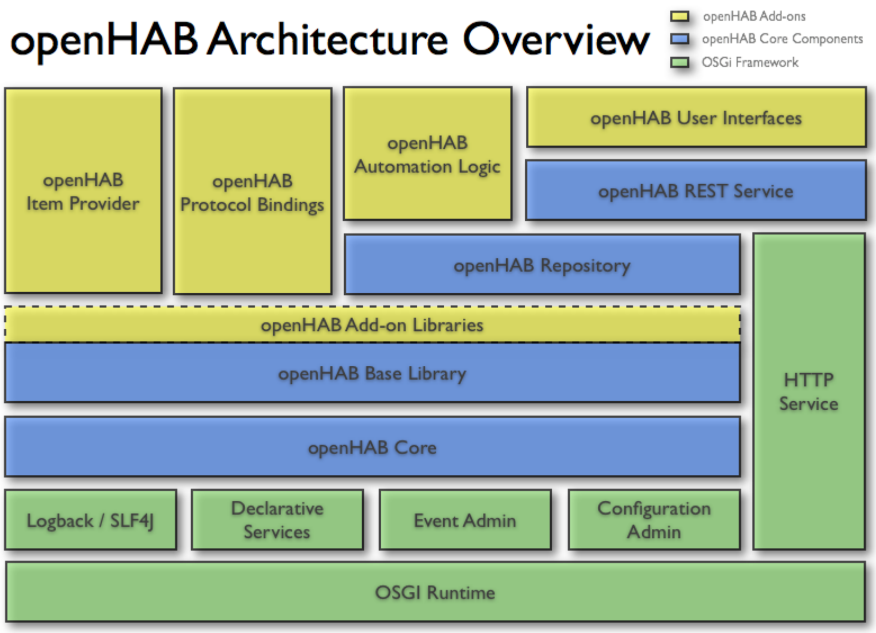


FIGURE 2: The software architecture of openHAB [66].

development process is needed to describe the software architecture. Due to this, the current stable release of openHAB in version 1.8.2 is used as reference.

OSGi Framework: From all services listed in the OSGi Compendium [39] that is implemented by the OSGi Framework Equinox, the openHAB software uses the following five services:

The Logback / SLF4j (Simple Logging Facade for Java) service provides a message logger for the OSGi Service Platform. The Declarative Services provide a publish/find/bind model for the usage of services. Applications use this model to manage the process and communication of bundles. The Event Admin service provides an inter-bundle communication mechanism based on a publish and subscribe model. The Configuration Admin service enables an operator to configure deployed bundles. The HTTP Service provides network communication access to bundles for HTTP, HTML, XML and other servlets.

openHAB Core Components: Since an official specification of openHAB's core components is missing, a high-quality project documentation [62] from Peter Manheller served as basis for the following description.

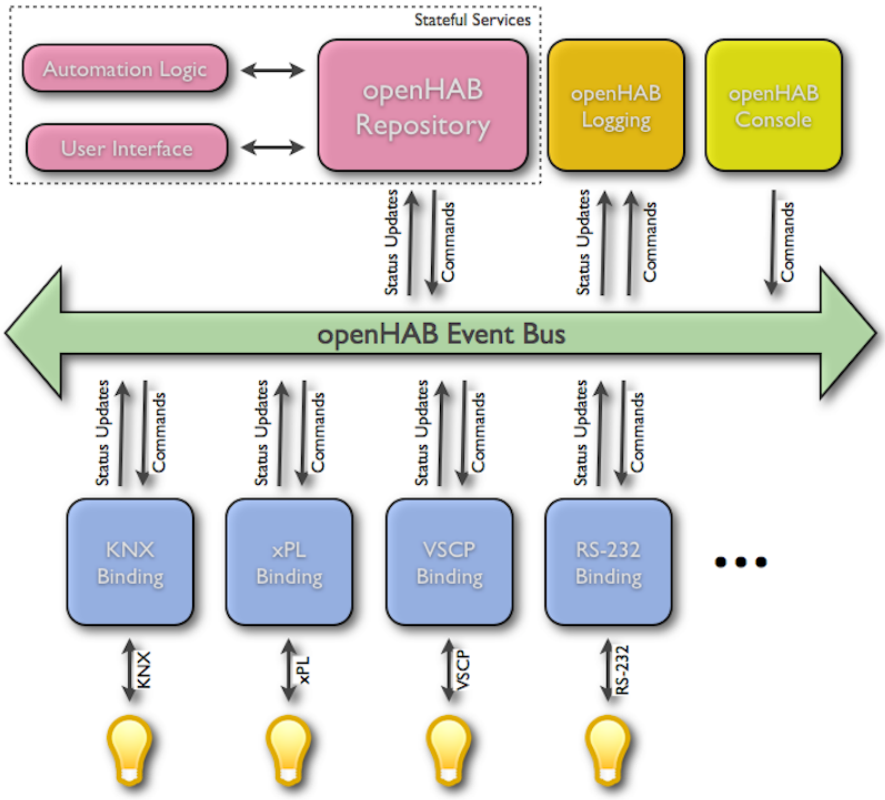


FIGURE 3: An overview of the communication structure in openHAB [66].

The main functionality of the openHAB core implements the Event Admin service as an event bus, providing a communication channel between bundles (see Fig. 3). The openHAB Base Library encompasses additional functionalities like a persistence service to store event data on databases [67], a script engine service to execute openHAB-scripts and several transformation services to process documents of different languages [68].

The openHAB Repository keeps track of the current state of all items in openHAB. In addition it manages the communication between bundles and items. An openHAB item is an abstract object that stores data which can be read from and written to in order to interact with it (as shown in Fig. 3). The openHAB REST Service component implements the HTTP service of the OSGi framework to provide network access for the user.

openHAB Add-ons: The openHAB software divides OSGi bundles into five categories: item provider, protocol bindings, automation logic, user interfaces and add-on libraries to provide a flexible configurable SH setup. The update interval of the bundles and general settings of add-on libraries is defined at the core configuration of openHAB.

The openHAB item provider initiates the items - that are managed by the aforementioned openHAB Repository - based on a configuration file, enabling the creation, management and deletion of items during runtime. The openHAB protocol bindings embraces many communication protocols to easily extend the functionalities of openHAB with respect to the SH setup [64]. Each binding is implemented as an OSGi bundle.

openHAB's automation logic encompasses a rule and a script engine. Scripts can be used to tidy rule code by transferring redundant code into a script and execute it in the consequent of an action rule. In addition, the rule engine is capable of parsing Java code in the consequent of a rule, providing limitless possibilities of actions. The rule engine grabs from the openHAB Repository all relevant items (see Fig. 3) to check their states. Relevant items are those, that occur in the antecedent of a rule.

Beside a webpage based interface, openHAB provides user interfaces for various desktop and mobile environments [65]. For reasons of simplicity, the user interfaces are configured by declarative UI definitions of the sitemaps. An openHAB sitemap is a file written in Xtext that configures the positioning of the declared items.

3.1.2 Other open source Smart Home solutions

Beside openHAB various other SH solutions were developed. Another popular open source solution is FHEM [55] (pronounces [FEMM][72]), instantiated by Richard König and continuously developed by a community around it. The project started in 2005 with a simple Perl script and was extended over the years by various modules from the community. The server component in the system architecture (see Fig. 4) still consists of a single Perl script, that is extended by various components that are connected via interfaces to the server.

As visualized, the main functionality of the FHEM server is to provide accessibility of the SH functionalities - included in the components - remotely over intra- or internet connections. For this purposes the server provides several frontends for web applications and smart phone operating systems, a floorplan module to create an outline of the apartment including positions of sensors and actuators, and an information panel to visualize sensor data in real time. Missing an internal rule-based system for complex

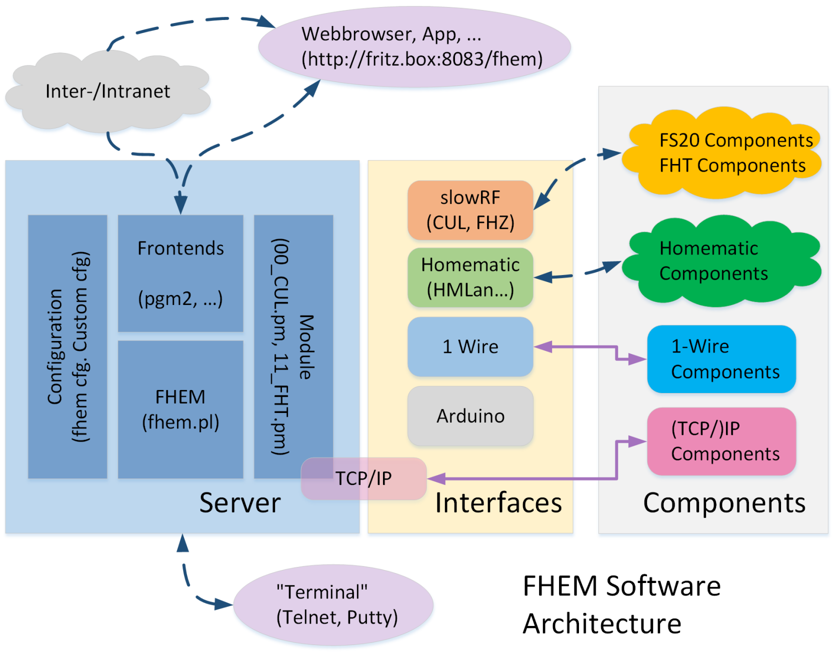


Figure 4: The system architecture of FHEM based on [76]

device management, FHEM users are constrained to programmatically implement a rule in a component or in the configuration file of the server.

Aside from FHEM that focuses on visualization and accessibility of the SH data, the more recent SH software Home Assistant [75] is built around a state machine enabling fast and simple home automation. The software architecture of Home Assistant - that is completely written in Python - is visualized in Figure 5. As in the previous SH solutions, the core functionality is the event bus that is used for communication and facilitating of firings and listening of events. The state machine keeps track of the state for all objects that are registered in the system. Components can provide services to the user or other components using the service registry. The service registry is also used to access services from other components. The timer - necessary to update the state machine without sensor events - generates a simple event every second. A major drawback in Home Assistant is its limited range of supported devices compared to FHEM or openHAB.

Regardless of the comparison, the choice for a SH software should suit the personal objectives of installing a SH. Every solution has its strengths and weaknesses. A solution suited for every case needs to be developed yet.

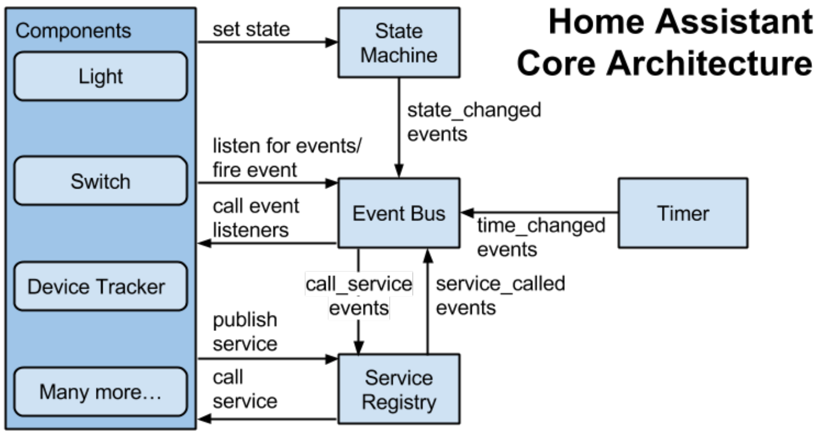


FIGURE 5: The software architecture of the Home Assistant software by [74].

3.2 Basics about Rule-based Systems

Rules are used by rule-based systems to infer knowledge in data and are therefore part of knowledge-based systems. Knowledge-based systems generally consists of two components, the knowledge base that contains informations and an inference mechanism that uses it to infer new knowledge. In rule-based systems the knowledge base consists of a rule base and a fact base whereas the inference mechanism concatenates the rules in a rule base.

3.2.1 Rules and rule bases

Considering the definition of Beierle in [4, p. 72] rules are formalized conditional clauses in the form of

If X then Y.

X represents a condition, also called premise or antecedent, which needs to be fulfilled in order to fulfill the conclusion Y, also called action or consequent. The state, if a rule's antecedent is fulfilled and the consequent is put into effect is also known as *firing* the rule.

Rules can be differentiated by the formulation of the consequent. A rule of the form - if X then *follows* Y - is a previous mentioned conditional clause while a rule of the form -

if X then *do* Y - is called a production rule. Production rules are used in this work to execute actuators.

The previous presented rules are of abstract nature and do not contain any domain knowledge. Domain knowledge in rules is represented as relations between facts (see Section 3.2.2). A common approach to formalize domain knowledge as rules is to consult a domain expert. Another approach is to infer domain knowledge from data which is followed by RI (see Section 3.3).

Irrelevant of the formalization approach, it needs to be considered that rules with contradicted conditional subclauses of the form - if X and $\neg$X, then Y - will be never fulfilled. The detection of such contradicted rules is a NP-complete problem since it is related to the boolean satisfiability problem (SAT) described in [26] by Richard M. Karp.

A rule base is a collection of rules that are applied on a fact base. If multiple rules manipulate the same facts, they can influence each other, infer unwanted knowledge or even stuck in infinite rule firings. Therefore, to produce qualitative results and avoid such issues, a rule base usually encompasses only a fragment of all possible rules that are necessary to achieve the objective.

3.2.2 Facts and fact bases

As before mentioned, facts are used by rules to infer knowledge. But before inference is applied it needs to be defined what facts are. Considering Beierle [4, p. 76] facts are objects including a description of their state, usually consisting of a discrete value. However, like an object can be a composition of multiple objects, a fact can also encompass multiple objects with their respective descriptions.

Rules represent relations between one or more objects and their respective states. A rule's consequent is able to modify the fact base during runtime which could lead to the fulfillment of further rules. The linkage of consecutive fulfilled rules is discussed in the following section.

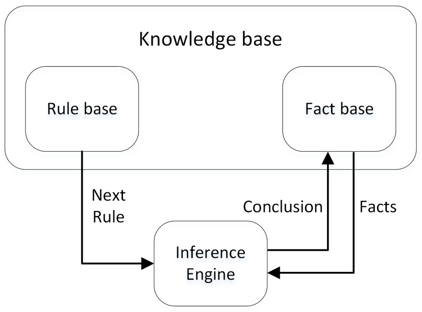


FIGURE 6: The abstract concept of rule-based systems by [80].

3.2.3 Inference mechanism

While rules and facts build the knowledge in a rule-based system the inference mechanism is a strategic problem solving algorithm that links this knowledge together to infer new knowledge which is again stored in the knowledge base (Fig. 6).

The general linkage approach of rule-based systems is to apply rules consecutively on the fact base. The most common concatenation strategies are forward and backward chaining.

The forward chaining inference strategy concatenates rules transitively by checking its antecedent and if fulfilled, executing the consequent. This is continued over all rules until no more conclusions are added to the knowledge base.

While the forward chaining inference strategy is suited to create an overview of a system, information of single facts are lost during the process, since the algorithm does not stop until no new knowledge can be inferred. Complementing this strategy, backward chaining searches for rules in the rule base that contain a given goal in their consequent, adding its antecedent if unknown to its goals and continues the search until all goals are reached.

3.2.4 Rule Engines

In order to execute rules, a rule engine is required that manages the fact base - usually realized as working memory - and implements the inference mechanism. In this section several open source rule engines are introduced, that are completely written in Java.

One of the best known rule engines is JBoss Drools [14]. Drools is licensed under the Apache License 2.0 and developed continuously by the JBoss community since 2001. In its current version 6.4.0, Drools uses the PHREAK algorithm as inference mechanism to match rules with facts in the working memory. The rule engine uses its own syntax for rule description, called the Drools Rule Language (.drl). While maintaining the classic If-Then schema, the syntax is mainly inspired by Java providing an additional Java parser for the consequent of the rules.

Another popular rule engine is OpenRules [31]. Since 2003 OpenRules was continuously developed by the identically named company. It is dual licensed under the GNU General Public License for non-commercial usage and under the OpenRules commercial license for commercial services. From the beginning, the rule engine utilized decision tables implemented in Excel spreadsheets as rule base. Single rules are represented as rows in the decision table and executed - by default - in top-to-bottom order as defined in the excel spreadsheet.

Two different implementations of a rule-based system were already presented. However it is not necessary to develop complex software in order to create a rule engine. In 2013 Mahmoud Ben Hassine developed a stupid rule engine called Easy Rules [53]. Easy Rules is licensed under the MIT license and in contrast to the previous rule engines, Easy Rules can be seen as a collection iterator that checks a collection of rules after their respective conditions and executes their consequents if matched. In Easy Rules, a rule can be represented by two ways: As a class that implements the Rule interface or extends the BasicRule class, and a Plain Old Java Object (POJO) that utilizes the @Rule annotation inspired by the JUnit syntax.

After understanding the concept of rule-based systems it is important to understand how rules can be learned from data, which is elaborated in the following section.

3.3 Rule Induction

Rule Induction (RI) as an area of ML aims to induce new knowledge in form of rules from given experience. Various paradigms for RI were developed that differ in objective, approach to discover the knowledge and applied data structure. A well elaborated approach - called Association Rule Mining (ARM) - is used in this work to discover the backbone of the target rule. Since many different algorithms for ARM exist, two well known representatives - elaborated in Section 3.3.1 - were chosen for this thesis to proof the concept. To further induce knowledge onto the backbone of the rule, another RI paradigm is applied, called Classification (CL). As for ARM, many algorithms were developed that induce knowledge from training examples by classifying them first and extract decision rules from the learned classifier afterwards. Again, to proof the concept, three different CL approaches are introduced in Section 3.3.2.

3.3.1 Association Rule Mining

ARM, as specialization of RI, aims to discover rules for associations in databases. An association rule is defined as an expression of the form $X \Rightarrow Y$ where X and Y are sets of items, in the following referred to as itemset. The meaning of such association rules is intuitive: If an item X occurs in the itemset, it is likely that an item Y will occur too. More precisely, given a database D containing itemsets $T \in D$, an expression $X \Rightarrow Y$ means that whenever T contains X it is - with a given confidence - likely that T also contains Y. The confidence is usually represented as percentage describing the ratio between itemsets containing X and Y with all itemsets containing X.

ARM is usually applied on Frequent Pattern Mining (FPM), because the values that are required to determine the confidence of the rule, are computed incidentally by FPM algorithms. Therefore the real challenge in ARM is to discover frequent pattern in itemsets. As a subfield of Data Mining, FPM aims to discover interesting pattern in data, where the interest is derived from the frequency of occurrence in the data. Several approaches for FPM were developed and applied to various types of:

- data (e.g. transaction or sequence databases, streams, spatial data, graphs, etc.),
- pattern (e.g. subgraphss, association rules, lattice, sequential pattern, etc.) or
- problems (e.g. discovery of frequent and compact item sets or sequential rules).

In this section the well elaborated ARM algorithm - Apriori - and its designated successor - FP-Growth - will be introduced. Since the extraction of association rules from the results of FPM is a common approach and shared among several ARM algorithms, only the FPM part of the respective ARM algorithm is considered. The extraction of association rules from frequent pattern is presented afterwards.

ID	Itemset
1	Bread,Butter,Milk
2	Bread,Honey
3	Milk,Eggs,Flour
4	Bread,Milk,Eggs,Cheese

TABLE 1: An example of a transaction database.

FPM algorithms are usually applied on a transaction database D. A transaction database stores data numbered in sets of unique items. Such a set is called transaction T, written $T \in D$ (see Tab. 1). An itemset $L_i \subseteq T$ - where i denotes the length of the set - is called frequent if its occurrence in all transactions exceeds a manual specified threshold called *Minimum Support* s.

Apriori

Apriori was proposed by Agrawal et al. in 1994 in order to discover association rules in large databases [37]. The algorithm operates iteratively bottom-up, increasing the length of the frequent itemsets by joining L_{k-1} with itself - where k represents the current itemset length - until no new frequent itemset are discovered. To apply pruning on the generated itemsets, a requirement needs to be satisfied that demands that any subset of a frequent itemset needs to be frequent too (in the following referred to as Apriori Property). For a better understanding, the steps of the algorithm will be described in the following enumeration:

1. Scan the database to get the frequency of every item in all transactions, also referred to as support S, generate for every item a 1-itemset and apply the threshold s on it to get L_1.
2. Use all variations of L_{k-1} itemsets in combination with the Apriori Property to generate frequent candidate k-itemsets.
3. Scan the database to get the S for each frequent candidate k-itemset and filter those out that are below s to get L_k. Repeat Step 2 & 3 until no candidate set passes s.

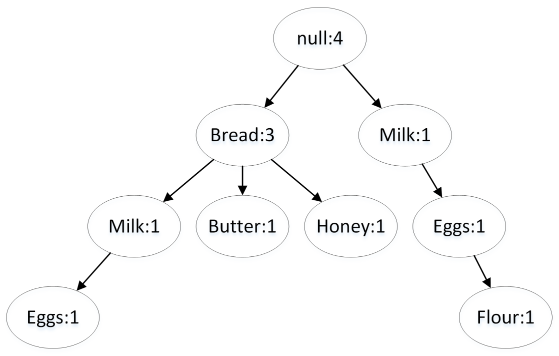


FIGURE 7: A complete FP-Tree example generated from the database in Table 1.

The advantage of Apriori is the pruning paradigm that filters out infrequent itemsets on the fly. Its major disadvantage is the multiple database access that is executed each candidate generation.

FP-Growth

While Apriori defined a powerful paradigm for pruning, its operations are applied on arrays of items ignoring any - for frequent pattern mining - superior data structure. An approach that differs in this way is the FP-Growth (Frequent Pattern). FP-Growth, proposed by Han et al. in 2000 [23], uses the properties of a tree data structure to discover frequent pattern without candidate generation. In general the algorithm is divided into two phases, building a pattern tree and extracting frequent pattern from it.

To build the FP-tree, the algorithm first calculates the support of every item in a separate table to determine its priority. Then every sequence is reordered in priority of support to every item it includes. Finally the FP-tree is build traversing through every sequence iteratively, creating a new node if an unknown item transition appears and increasing the counters - applied on each node - every time the traversal passes an already created node. Pointers are maintained between nodes of the same class creating a single linked list for every class. They will be used in the second phase for faster discovery of frequent items. As example, the final FP-tree generated from the transaction database in Table 1 is visualized in Figure 7.

After the FP-tree is built completely, the patterns are extracted by a bottom-up divide and conquer traversal. Starting with the least prior item, the support of all points on the single linked list is summarized and the item is extracted as frequent pattern if its support exceeds the Minimum Support s. Then the FP-tree is traversed recursively in

ID	Itemset
1	~~Bread,Butter,Milk~~
2	~~Bread,Honey~~
3	Milk,~~Eggs~~,Flour
4	Bread,Milk,~~Eggs~~,Cheese

TABLE 2: The database to build the conditional FP-Tree for the frequent item Eggs, based on the database of Table 1

inverse priority order, generating longer patterns by exploring paths of the tree that end with the previous discovered frequent item. For this, conditional FP-trees are generated in order to determine the support of all prefix path, that end with the frequent item. A conditional FP-tree is a tree, that would be built by the algorithm if it used the prefix tree-paths of an item in all branches as database. This means that only those itemsets are used as database, in that the frequent item is included while it is excluded for the FP-tree generation itself. An example of the resulted database to build a conditional FP-Tree for the prefix tree-path of the item Eggs is given in Tab. 2. The algorithm terminates if the root node of the original FP-tree is reached.

Derivation

In order to derive association rules from the discovered frequent itemsets, Agrawal et al. proposed in [37] an approach that requires simple counting and a threshold, called rule confidence. Considering the initial situation that a list of frequent itemsets F is given. Based on this, the extraction of association rules can be separated into the following steps:

1. For each frequent itemset in F extract all non-empty subsets $\{s_i\}$.
2. For each subset $\{s_i\}$ export the rule $\{s_i\} \Rightarrow \{F - s_i\}$.
3. Calculate the confidence as $Conf(\{s_i\} \Rightarrow \{F - s_i\}) = \frac{Support(F)}{Support(s_i)}$.
4. Drop all association rules with lower confidence than the given threshold.

This simple derivation of association rules from frequent itemsets highlights the fact, that the most challenging part in ARM is FPM.

3.3.2 Classification

Although ARM can be used to discover strong relations between sensors, they are not suited to describe a class or even classify classes as implied by [19], because the resulted set of discovered association rules would be enormous and highly specialized on every value. Therefore, in order to differentiate relevant from irrelevant situations before an actuator's execution, CL is applied.

In general, each CL approach can provide - more or less simple - a set of decision rules that describes the learned CL model. However, logic based approaches like decision trees or rule set induction are focused in this work, due to their easy interpretation and transparency. In addition, since there exist various different approaches for CL, only a few selected are introduced.

Decision Trees: PART

The algorithm PART - developed by Frank and Witten in 1998 [17] - combines learning paradigms from C4.5 and RIPPER by using the separate-and-conquer principle (RIPPER) to build partial decision trees (C4.5) and extract in each iteration the "best" leaf into a rule. Frank and Witten experimented with different definitions of best, like the lowest error rate according to the Bernoulli heuristics of C4.5 but used the most general as best since other approaches did not have any effect on the overall accuracy of the generated rules.

A rule is generated similar to C4.5 by building a decision tree. Starting at the root the algorithm splits a given set of training examples recursively in a breadth-first-search approach. At each split the entropy is calculated for every attribute in the examples, following the branch of the smallest entropy until all children of a node are leafs. Then pruning begins. The pruning of PART is similar to the subtree replacement of C4.5. If the estimated error for the subtree is greater or equal to the estimated error of a node, the subtree will be discarded and the node will turn to a leaf. After that the algorithm backtracks its path exploring the nodes sibling. However if during backtracking a node is encountered whose children are all further nodes, then the remaining subsets are not explored and the corresponding branches are left undefined. Due to the recursive structure of the algorithm, this event will terminate the decision tree building.

Finally - as mentioned before - the most general leaf is extracted as a rule. This is accomplished by transforming the path from root to leaf as a rule. The rest of the decision tree will be discarded. The aforementioned separate-and-conquer strategy is applied in the rule set generation process. For every rule that is built, its covered instances are removed from the training set and the procedure starts again until no instances are left.

Ripple Down Rules: Ridor

Ridor is an implementation of the RDR (Ripple Down Rules) technique - proposed by Compton and Jansen in 1990 [10] - for knowledge acquisition and maintenance that follows the rule generation procedure of an expert. By starting with a default rule - a rule with no conditions and the most frequent class as target - exceptions are generated for the default rule. While training data is processed, exceptions are recursively added to other exceptions until each exception contains either only exceptions or a pure set of classes. These exceptions are then turned into rules. The resulted rule set contains rules for every case except the default rule. While RDR started as a binary classification technique, Kang et al. upgraded the approach in 1995 to support multiple classes [25].

It should be noted that RDR resembles a lot the generation of a decision tree. However, the main difference to a decision tree is, that the decisions are made upon on exceptions of conditions, and that they do not need to cover all cases.

Inducing Modular Rules: PRISM

PRISM - introduced by Cendrowska in 1987 [7] - originates from the ID3 algorithm - the predecessor of the aforementioned C4.5 - that generates decision trees from data. Cendrowska argued that the origin of one of ID3's major weakness - the decision tree output - lies in the inductive learning of the algorithm and proposed an alternative approach.

PRISM can be described in five steps. If rules for multiple classes are required, this algorithm can be applied on every class δ_n where $n \leq N$ and N is the number of all target classes:

1. Calculate the probability of the occurrence for every attribute-value pair α_x in δ_n.

2. Select the most probable α_x and create a subset of the training set comprising all instances that contain the selected α_x.
3. Repeat steps 1 & 2 until the training set contains only instances of class δ_n. The resulted rule is a conjunction of all attribute-value pairs used to classify this homogeneous subset.
4. Now remove the homogeneous subset from the original training set.
5. Repeat steps 1 - 4 until all instances describing class δ_n have been removed from the original training set.

This approach can be repeated after the original data is restored for every other class. It has been noticed that the induction approach is very similar to ID3's approach. However, the major difference lies in the attribute-value choice of both algorithms. While ID3 aims to find the most relevant overall attributes, PRISM focuses on finding only relevant values of attributes.

3.4 Biomedical Data

Since the term *biomedical data* embraces a broad variety of different perceptions of data, this work defines biomedical data as data that is: (1) measurable, (2) is emitted continuously by a human body and (3) is somehow related to the physical condition of the person. In addition, because it is assumed that a person can behave differently or prefer a domestic environment only a under specific physical condition, this data can be used to sensitize a system after the current condition of an inhabitant.

Nevertheless, one should not think that the raw data of his heart rate or blood pressure will have any affect on the learning system. The data needs to be prepared and interpreted in order to gain meaningful information from it. Therefore, this section aims to introduce various different types of interpretable biomedical data and exemplary elaborates on the HRV, which information can be used to sensitize the system.

3.4.1 Types of Biomedical Data

Various different measurable biomedical features exist that describe the current physical conditions of a person. The measuring method of biomedical data depends on the feature and is usually divided into invasive and non-invasive techniques. As named, invasive measurements invade the human body by using sharp tools like needles. The duration of the measurement for invasive techniques is limited, because the body is stressed due to the intrusion and requires time to recover from it. Although limited in time frame, a long distance observation can be realized by repeating the measurement periodically. A well known example for an invasive measured biomedical feature is blood glucose, which is regularly checked by people who suffer from diabetes. Non-invasive techniques on the other hand measure biomedical features that a person emits out of his body. Many of them can be measured continuously because they do not harm the body. Popular examples for non-invasive biomedical features are the temperature, heart rate or blood pressure.

In the context of SHs, non-invasive biomedical features are preferred because they can extend the continuous observed domestic feature space, by features from the supported subjects. Using discontinuous data always requires additional computation for regression and prediction, while never reaching the accuracy and reliability of continuous data. Regardless of the time space, the data needs to be interpreted correct in order to sensitize the system. An example for an interpretation of biomedical data is given in the following section.

3.4.2 The Importance of Heart Rate Variability

Every living person has a heart beat. It is a series of cell reactions that pumps blood through the body, refreshes its oxygen saturation and is set in motion by the electrical signal of a unique composition of cells, called the sinus node. Its frequency is described by the heart rate, that states the number of beats per minute. The HRV describes the differences of the time intervals between each heart beat.

Usually these time intervals vary each time, because muscle tension, breathing rate, sweating and other energy consuming functions in the body change continuously requiring always different amount of energy. The heart tries to maintain homeostasis and varies the production of energy according to the current need. A healthy body will therefore always have a varying HRV.

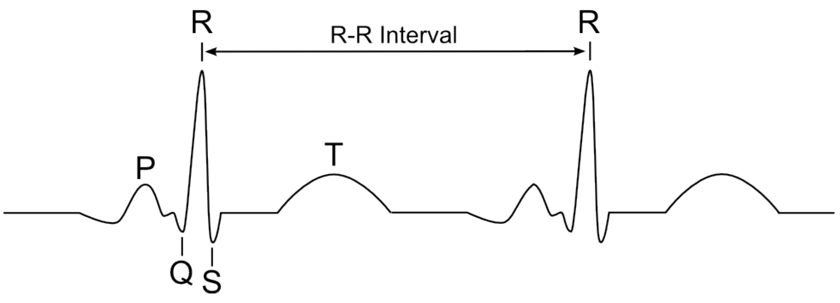


FIGURE 8: An electrocardiogram that highlights the R-R interval by [77]

However, if a person is stressed or something else is disturbing the cardiovascular system, the HRV becomes rigid. It was observed that the HRV can be accurately used to indicate stress on a person. Olsson confirmed this relation during several studies in [30]. A low variability in the interval of time between two heart beats indicates that the person is stressed. This indication can be used to determine the condition of an inhabitant, adapt the domestic environment according to the past observations of his behavior in this condition and so, sensitize the system to the inhabitants.

The HRV can be measured in time domain or frequency domain. In time domain it is determined based on the Inter-beat Interval (IBI), that measures the time in milliseconds between two heart beats, also called the R-R interval in an electrocardiogram (see Fig. 8). Important measures in time domain are the standard deviation of the IBI or the NN50, that counts every consecutive IBI that differs from the previous by more than 50ms. By using Fourier or Fast Fourier Transformation those measures can be transformed into frequency domain, enabling the exploration of IBIs as a function of frequency.

By preprocessing the HRV over the heart rate data and clustering it into several categories, the rule learning system will be enriched by a feature that indicates the current stress level of the inhabitants and therefore probably relate its behavior to it. It should also be noticed that - beside sensitization - the interpreted biomedical data could be used to optimize the behavior of the system against some calibrated optimum. However, since the calibration procedure would require domain knowledge from an expert, it was not included into the concept of the system.

4 Solution Approach

In Section 2 the current state of the art in AmI for SHs was elaborated, which main conclusion was that most of the current approaches are specialized on sensors, lacking the capability of self-adapting their input space and vary in the definition of behavior related context.

In this section, a solution for behavior rule learning is proposed, that tackles these disadvantages and addresses AmI for SHs from a more practical perspective. For this, Section 4.1 starts by revisiting the objected learning problem in more detail. Based on this the general behavior rule learning concept is described in Section 4.2. Finally, the adaptation mechanism for the learned rules is introduced in Section 4.3.

4.1 The Learning Problem

Considering the definition of a Learning Problem by Tom Mitchell [29, P. 2]:

> *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

With respect to this definition, the tasks in T are to learn the executions of actuators - in a SH - similar to the usage behavior of the inhabitants. The experience E consists of the SH event stream, where the history of the sensory states and past activations of actuators are recorded in temporal order. The performance measure P is the F-Measure, a value that describes the correlation between the accuracy of rules and the coverage of relevant situations.

These entities are elaborated in the following subsections. In addition, the underlying assumptions are described and similar problems are compared to this learning problem.

4.1.1 Assumptions

Since the setup of a SH varies from apartment to apartment, some assumptions need to be defined in order to build a solid base line for the learning problem. The first paragraph guarantees that the data, on which the algorithms are applied on, consists of sequential events from a SH and that some of these events are associated to the target actuator. The second paragraph guarantees that, based on the inter-device association, another contextual relation exists, which is observed directly or indirectly by the installed sensors.

It is assumed, that the activation procedure of an actuator is observed by at least one sensor that differs from the actuator itself. It is also assumed, that every actuator is activated manually by the inhabitant in a more or less regular order of actions. Due to the previous two assumptions, it can also be assumed that the observed event sequence - before an actuator is activated - contains at least one regular event pattern.

Beside sequential observation it is assumed, that at least one domestic environmental feature of the SH, that is influenced by an actuator or its respective activation procedure - in the following referred to as the environmental state of an actuator - is observed by sensors of the SH. Due to the assumption of regular event pattern before an actuators execution, it is assumed that similar regularities for the observed environmental state in a SH exist. Therefore it can also be assumed that the environmental state before the execution of an actuator, contains at least one feature that is similar over previous activations.

4.1.2 Similar Problems

Similar problems of AmI in SHs encompass the challenges of AR and BP as described in section 2.1.3. In fact, this learning problem can be categorized between these problems.

As for AR this learning problem relies on sequences of events to detect known pattern of activities. However, the objective is not to recognize different pattern of events in order to describe an activity that the inhabitant is executing, but to describe when a actuator should be executed.

The main difference to the BP is not only the objective but also the way the behavior model is generated. While BP first generates a model to predict the behavior of the inhabitants, this problem first generates rules that represent one aspect of the behavior of the inhabitant. By putting every aspect together, the behavior model can be generated.

However, compared to BP there is no further usage, since learning the model is the main focus of this problem.

4.1.3 Task

The aforementioned tasks in T can be seen as a behavior learning problem. Human behavior can be seen as pattern of activities, that are executed under specific environmental states and therefore follow some rules. Hence main task T can be divided into ARM to discover the pattern that leads to the execution of an actuator, and CL to discriminate relevant from irrelevant environmental states. By solving both subtasks, a rule should be discovered that resembles the behavior of the inhabitant for an actuator.

Since a physical actuator device has usually several executable settings through that the inhabitant influence its environmental state, every setting or usage mode needs to be learned separately. The combination of a physical actuator device and one specific configuration it is executed with, is in the following referred to as virtual actuator.

By applying the aforementioned concept on all virtual actuators of a SH, a rule set should be generated that describes the whole behavior model of the inhabitant for the SH usage.

4.1.4 Experience

The experience E in this learning task consists of the data observed by the sensors in the SH, in the following referred to as event stream. Since ARM - as the first subtask of T - requires a transaction database that it can be applied to, several itemsets are retrieved by segmentation from the event stream. In addition, since CL - as the second subtask of T - requires positive and negative training examples, several relevant and irrelevant environmental states are extracted from the event stream.

4.1.5 Performance

Performance is measured by computing the F_1-Score, a statistical measure that describes the value of a classifier. Using the generated rule as a classifier by checking if it was activated or not for a given situation, the F_1-Score can be calculated to determine the

	Positive Situation	Negative Situation
Classified Positive	**t**rue **p**ositive	**f**alse **p**ositive
Classified Negative	**f**alse **n**egative	**t**rue **n**egative

TABLE 3: A confusion matrix for a binary classifier.

performance of the rule. In order to ensure an independent evaluation, the performance of the rule is determined on unknown data.

The F_1 correlates the Recall and Precision of a classifier by calculating

$$\mathbf{F}_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{1}$$

The Precision, also called the positive prediction value, describes the number of correct classified situations in relation to all positive classifications formulated as

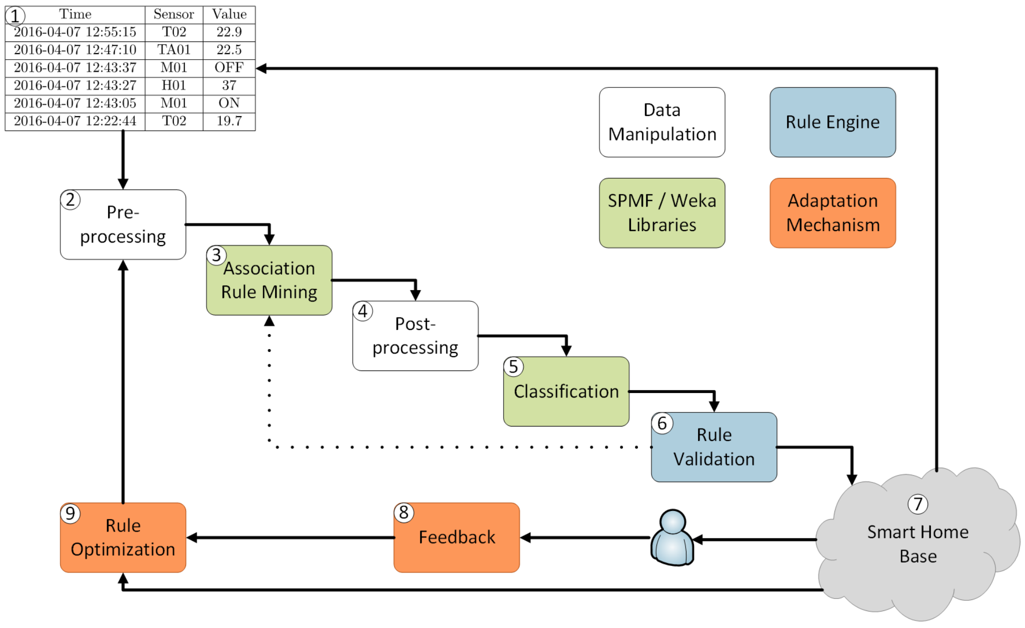
$$Precision = \frac{tp}{tp + fp} \tag{2}$$

The Recall, also called true positive rate, describes the number of correct classified situations in relation to all positive situations formulated as

$$Recall = \frac{tp}{tp + fn} \tag{3}$$

To understand the formulas 1 - 3 easier, the responsible values are visualized in a confusion matrix in Table 3. Finally, using the F_1-Measure it should be considered - like Reynold and de la Iglesia stated in [71] - that false positives and false negatives should not always be weighted equally, because - depending on the area of application - a false positive could lead to fraud data whilst a false negative could cost human lives when something important is not detected.

However, since the importance of specific sensors in a SH can not be predetermined, these assumptions can not applied to this project. Therefore - although biased - the F_1 measure is preferred because it can be easy calculated without the requirement of true negatives. The determination of true negative context requires knowledge about the inhabitant that may not be observable.



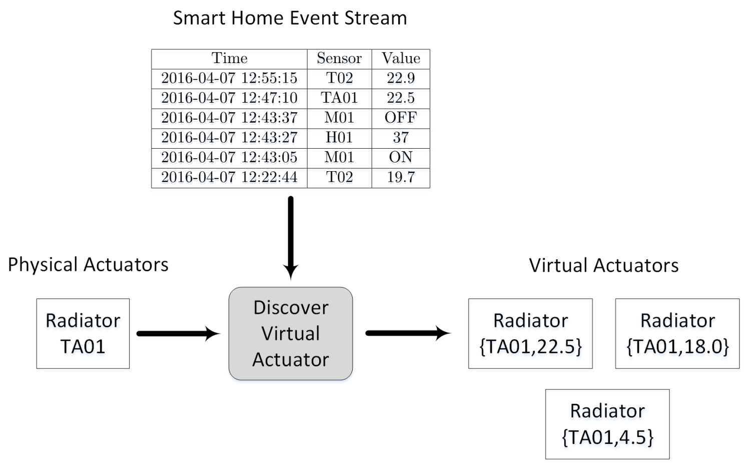
Time	Sensor	Value
2016-04-07 12:55:15	T02	22.9
2016-04-07 12:47:10	TA01	22.5
2016-04-07 12:43:37	M01	OFF
2016-04-07 12:43:27	H01	37
2016-04-07 12:43:05	M01	ON
2016-04-07 12:22:44	T02	19.7

FIGURE 9: The conceptual design of the learning approach.

4.2 Smart Home Actuator Rule Learning

After the learning problem was defined in the previous section, the conceptual challenges of realizing the aforementioned learning task is addressed in this section. An overview of the concept is given in Figure 9. The whole system can be divided into a rule learning and an adaptation module. This section addresses only the concept for rule learning. The adaptation mechanism - colored orange in Figure 9 - is described in Section 4.3. For convenience reasons, the aforementioned virtual actuators - that were introduced in the previous section - are in the following referred to as actuators.

The learning process - where the order of every component is identified by the top-left number in Figure 9 - embraces a serial composition of components for data preparation, ARM, data postprocessing, CL and validation of the learned decision rules. Since the possibility exists, that no rules are discovered due to a bad allocation of training and evaluation data, the concept is capable of trying to learn rules for the same actuator several times on a different data basis. If the performance of a validated rule passes a user defined threshold, it is exported to the file system of the SH. In addition, to keep an overview of the learned rules and for management and adaptation reasons, several basic informations about every exported rule are stored in a table and saved to the file system of the proposed learning system.



Time	Sensor	Value
2016-04-07 12:55:15	T02	22.9
2016-04-07 12:47:10	TA01	22.5
2016-04-07 12:43:37	M01	OFF
2016-04-07 12:43:27	H01	37
2016-04-07 12:43:05	M01	ON
2016-04-07 12:22:44	T02	19.7

FIGURE 10: A exemplary split of one physical actuator into several virtual actuators.

As input, the learning system requires the data format of a common SH bus, where each event on the bus has a timestamp, a label of the responsible sensor or actuator and the data that it transmitted. The events are ordered chronically reverse, so that the newest event is always first as shown in the table of Figure 9. Since a sensor does not differ from an actuator in this data, the labels of the target actuators need to be given beforehand to the learning system. In addition, if not a specific actuator but all possible actuators shall be learned, they need to be discovered by searching for unique combinations of the target actuator label and its respective value as visualized by Figure 10. Having the list of target actuators and the event stream prepared, the system can start the learning process for every actuator separately.

Starting with Section 4.2.1, the tasks in preprocessing are presented. Then in Section 4.2.2, the integration of ARM into the learning task is explained. After that, the tasks for postprocessing the data are presented in Section 4.2.3. Followed by this, the integration of CL into the learning task is described in Section 4.2.4. Finally, the concept for validation of the learned decision rules is presented in Section 4.2.5.

4.2.1 Preprocessing

As described by Han et al. in [22], preprocessing encompasses tasks like cleaning, integrating, reducing and transforming the initial data to increase the data quality and hence the output quality of algorithms. In this case, preprocessing encompasses the tasks of segmenting the event stream and extracting an itemset from every sequence.

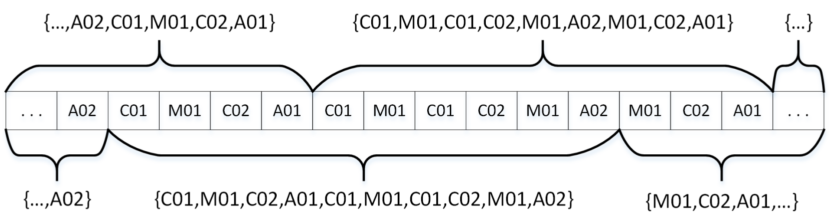

{...,A02,C01,M01,C02,A01}
{C01,M01,C01,C02,M01,A02,M01,C02,A01}
{...}
. . .
A02
C01
M01
C02
A01
C01
M01
C01
C02
M01
A02
M01
C02
A01
. . .
{...,A02}
{C01,M01,C02,A01,C01,M01,C01,C02,M01,A02}
{M01,C02,A01,...}

Figure 11: The sequential segmentation for two actuators.

There are multiple ways to segment an event stream based on density [81], spatial distance [42], time [43], a sliding window [46] or any other reasonable criteria. However, since the objective is to discover strong association rules between sensors that are related directly to the execution of an actuator, the pattern before such an event is the most interesting part.

In addition, since the optimal target pattern length depends on unknown criteria like the number of installed sensor and the relations between them, it is assumed that an actuator could be executed at any time after its last execution. Therefor it is obvious to segment the event stream on exactly these characteristics, so that every sequence ends in an actuator event and starts after one. Figure 11 visualizes an exemplary segmentation of the event stream for two different actuators, named A01 and A02. For a better understanding, the timestamps and values are excluded from this visualization.

After a list of sequences is segmented from the event stream, an itemset is extracted from each of these and stored in a separate list. Each item in such an itemset consists only of the sensor or actuator label of its respective event. The reason for this is explained in the following section.

4.2.2 Association Rule Mining

After preprocessing, from the list of itemsets - containing the sensor and actuator labels similar to the sequences of Fig. 11 - a transaction database is created and given to an ARM algorithm. Although the whole learning task could be realized using only ARM - for example by merging device label and its value as one item - this approach would move the meaning of relations from devices to device-value pairs, generating a vast amount of highly specialized association rules instead of only several robust. To avoid

M01 M02 -> TA01
C01 T01 -> TA01
C01 M01 M02 -> TA01
C01 M01 T01 -> TA01
C01 M01 M02 T01 -> TA01

TABLE 4: A sample output of the Association Rule Mining task.

such overfitting, it was decided to first discover relations based on the device labels only, and then induce knowledge about the environmental state to them.

The concept of ARM was elaborated previously and will not be repeated in this section. However it should be noted, that from the two algorithms that were elaborated in Section 3.3.1, FP-Growth outperforms Apriori in execution time [51] and should therefore be preferred. In addition, ARM can output several association rules of different lengths (see Fig.4). This is explicit wanted - instead of discovering only the most likeliest - to explore a broader area of solutions.

Last but not least, as described in Section 3.3.1, association rules are generated for every subset of a frequent pattern and thus leading to association rules for other actuators or even sensors. Since these rules are not required in the current concept they are ignored and handled during postprocessing.

4.2.3 Postprocessing

The output of ARM consists of a list containing various association rules. Postprocessing iterates over this list applying the following operations: (1) Consequent actuator filtering and the (2) extraction of positive and (3) negative examples.

As described previously, the list of association rules may contain rules for the wrong target. By comparing the consequent of an association rule with the label of the target actuator, irrelevant association rules are filtered out. A rule that passes this filter is a rule candidate for the SH.

As mentioned before, positive and negative examples of the environmental state - when an actuator was executed - are required for CL. Since the target decision rule will only consider the related sensors and not the whole environmental state of the SH, just a subset of it - covering every sensor or actuator in the antecedent of an association rule - is required as training example. Sequences that do not contain all sensors of the respective association rule, will not be used for either positive or negative examples, because the environmental state can not be differentiated completely.

While positive examples can be extracted from sequences that were segmented during preprocessing, the extraction of negative examples is not unambiguous. This is because a negative example can originate from two reasons: If a physical actuator was used differently or if it was not used in the wrong context. Since it is - without further analysis - not possible to reason the environmental state about missing executions, negative examples are defined as positive examples for different settings of the same physical actuator. However, this definition causes an uneven distribution among the positive and negative examples if an actuator has more than two executable settings. Therefore all negative examples are stored in a pool, from that - according to the positive set size - a number of random examples is used as negative training examples.

As shown in Figure 9, the initial data consists of a table with three columns: Time, Sensor and Value. In order to increase the feature space for the discovered decision rules, it is considered to extract as much information as possible from the data. Although there exist algorithms that discover important features automatically, it is preferred - for evaluation purposes - to bias the system manually and thus guide the CL algorithm to consider only specific conditions in the decision rule. Therefore the following features are extracted manually from the timestamp: Month, Week, Weekday, Day, Hour, Minute and Second.

The feature space for the CL approach is generated by merging the temporal with the sensory features - that are defined by the antecedent of the respective association rule - as visualized by Table 5. The last column classifies the example as positive or negative. The columns vary, depending on the bias and current association rule. It should be noted, that all numerical values are treated as nominal, because it can not be guaranteed that every numerical value in a SH has a continuous range. For the same reason no features are clustered, although the temporal extracted would be considerable candidates for clustering.

ID	Weekday	Hour	Minute	Second	C01	M01	TA01
1	Monday	11	10	34	OPEN	ON	+
2	Monday	13	28	45	CLOSED	ON	-
3	Tuesday	10	58	22	OPEN	OFF	+
4	Tuesday	15	33	10	CLOSED	ON	-

Table 5: Four sample training examples on that Classification can be applied.

4.2.4 Classification

As for ARM, a specific approach for CL is not required. Using the generated training examples as input, a classifier is trained that differentiates relevant and irrelevant environmental states with respect to the initial association rule. If the CL approach itself does not generate decision rules, they need to be extracted manually from the classifier. However, all approaches that were introduced in Section 3.3.2, output decision rules from the trained classifier.

This leads to the fact, that not only one decision rule is extracted from the classifier, but rules for every possible classification. Since only rules for the target actuator are wanted, a simple filter is applied at the output of the classifier to remove all rules from negative classifications. In addition it should be noted that, due to missing publications concerning the comparison of the presented approaches, a preference on one algorithm will not be given.

After the classifier was trained, the output of the CL algorithm is a set of decision rules of the form

$$IF \ \texttt{Sensor A} = \texttt{Value} \ THEN \ \texttt{Actuator} = \texttt{Value}$$

where the antecedent can embrace several conditions. It should be noted that - due to the transformation of numerical to nominal values - only rules are generated that check for equality.

The validation of the rules is described in the following section.

4.2.5 Rule Validation

Since the proposed algorithm generates for every discovered association rule several decision rules, a vast amount of different rules is learned. In order to support the inhabitant, it is important to export only the most reliable rules into the SH system. As described in the learning problem, the F_1-score is used as performance measure for the validated rules. To export only the best rules for an actuator, the following operations are applied on all decision rules: (1) Determination of the performances, (2) application of a given performance threshold and (3) final exportation of all passed rules. Based on a parameter, the system will repeat the whole learning process several times in order to increase the rule discovery rate.

As mentioned in Section 4.1, to determine the performance of a rule, its execution can be seen as a binary classification problem. The data - that consists of unknown positive and negative environmental states - needs to be classified into right and wrong. The states can be retrieved in a similar way as the training examples by using the positive and negative sequences from the target actuator and its different settings. By adding the sequential events in order and applying the rule on the final environmental state, a real SH scenario is emulated.

As for ARM and CL before, several different approaches of rule engines were introduced in Section 3.2.4. From these systems, Easy Rules is preferred over Drools and OpenRules, due to its lightweight properties and complete integration in Java, including the rules. However, the concept is not bound to one rule-based system and the application can be changed if transformation is adapted to support the rule-syntax of the target rule engine. In order to execute the decision rule, it needs to be transformed into an engine compatible format. Since the proposed algorithms require different formats of rules, this task is postponed to the respective implementation of the rule engine.

Each time the rule fires, the current inserted state can be seen as *positive* whilst anything else is classified as *negative*. The standard of truth, that is required to validate such a binary classification task, is retrieved incidentally by generating the positive and negative states. The discovered rules are not guaranteed to perform well, especially if the environmental state or behavior of the inhabitant vary each execution. Therefore a user defined performance threshold needs to be applied, that filters out bad performing rules. Every rule that passes the filter will be exported to the SH.

Attributes	**Rule 1**	**Rule 2**
RuleID	-713358601	894089539
Antecedent	{C01=OPEN, M01=OFF}	{C01=CLOSED, T01<=10.0}
Consequent	{TA01=4.5}	{TA01=20.0}
F_1	0.77	0.73
Precision	0.63	0.58
Recall	0.99	0.99
Timestamp	2016-05-04T01:16:40.555	2016-05-05T16:30:22.966
Filepath	\SmartHome\rules\-713358601.rule	\SmartHome\rules\894089539.rule

Table 6: An exemplary performance table, filled with artificial rules.

As before mentioned, in order to retrieve the exported rules for later use, basic informations are captured in a table and stored onto the file system of the learning system. These informations include a path to the location of the exported rule, a timestamp of its generation, basic performance results, its antecedent and consequent naturally and an identifier generated from their composed hash value. A structured overview of the captured informations - including two artificial examples - is given in Table 6.

Before the system continues with the export of the rules, the learning process can be repeated several times by a given parameter to increase the best performance of the number of discovered rules. Although comprehensible, the algorithm will not stop in between if a rule has been discovered that has passed the performance threshold, because it is aimed to discover even better rules and thus will always retry the learning process according to the given amount of trials. However, in order to discover new rules, the datasets for training and evaluation needs to differ each time. Therefore, before the repetition is started, both sets are cleared and mixed anew from the original dataset containing all sequences of the segmented event stream.

After that, the internal rule needs to be transformed into a SH compatible format provided by an interface, in order to keep the core system distinct from the SH software. During transformation it should be considered that the rule could check for environmental features like current time in hours or the month but the rule-based system of the SH is not capable to evaluate such conditions. In order to propose a SH independent system, a solution to this problem is not included in this concept. If such a situation occurs, specialized workarounds need to be provided by the class that implements the rule interface, that may remove the condition, adapt it or delete the whole rule.

Finally the rule itself is serialized onto the file system by using the rule identifier as filename, and accessible for both, the SH software and the proposed rule learning system.

4.3 Adaptation Mechanism

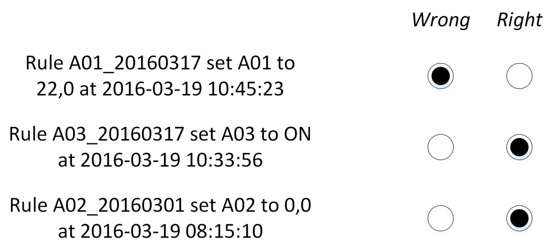
Until now, the system is capable of learning actuator management in a SH by observing the habits of the inhabitants. However, the process is executed a single time and needs to be triggered manually, and thus does not adapt on a changing environment what contradicts to the definition of AmI. To apply adaptation on the rule base, the system needs a mechanism to recognize that the learned knowledge is deprecated and thus new rules should be discovered. This applies also to rules that may perform well for observed data, but poorly under real conditions. In order to determine the performance of a rule aside from the validation process, feedback is required that describes if a rule behaves as wanted in the SH. In addition a mechanism is required to adapt the learned rules to the changed environment.

In the following section, the concept of feedback retrieval is proposed. Afterwards, the mechanism for rule adaptation is addressed in Section 4.3.2. Last but not least, it should be noted that - due to the conceptual design of the system as an external learning module - most of the following concepts need to be implemented in the SH directly. This is applied particularly to functionalities that require an external system, like the configuration of the learning frequency and its execution or the logging of past rule decisions.

4.3.1 Feedback

Due to the unsteady environment of the system - originated from changes in the peripheral setup, persons or behavior - feedback is necessary to continuously evaluate the learned knowledge under real conditions. Feedback can be provided mainly from two sources: directly from the inhabitant or indirectly hidden in the data.

Direct feedback can be captured by a Graphical User Interface (GUI) that presents past executions of actuators based on the generated rules (see Fig. 12), so that the inhabitant can tag wrong decisions retrospectively. For this purpose, the GUI requires access to a file where such executions are logged. As mentioned before, this task needs to be



	Wrong	*Right*
Rule A01_20160317 set A01 to 22,0 at 2016-03-19 10:45:23	●	○
Rule A03_20160317 set A03 to ON at 2016-03-19 10:33:56	○	●
Rule A02_20160301 set A02 to 0,0 at 2016-03-19 08:15:10	○	●

FIGURE 12: A conceptual Graphical User Interface for feedback.

realized by the logging facility of the SH. By adding a logger to the consequent of an exported rule and log its identifier, every execution is captured in a logging file, then through the GUI linked to the responsible rule and - if wanted - passed to the learning system for replacement.

Regardless of direct feedback, adaptation can be realized by continuous frequent (e.g. daily, weekly, monthly) learning. If the performance of a discovered rule deteriorates on new data and falls below the predefined threshold, it can be replaced by a new one or simply removed. Since it is not predictable at which interval the rule base should be updated, the configuration of the frequency is left to the inhabitants.

Another more complex way to gather indirect feedback, is to keep track of the state change from an actuator and relate it with wrong rules generated by the system. Although an automatic collection of qualitative feedback is desirable at first, the association of an actuator state change and its origin is no trivial question because it can occur due to many reasons like the preferences of another person, simple passing of time, the change of any other unobservable parameter or a real wrong decision. Since it cannot be guaranteed that the change in the state of an actuator originated from the wrongness of the rule, its usage will not be considered in this work.

The conception and implementation of the GUI depends strongly on the SH software and its functionalities. From the solutions presented in Section 3.1, openHAB is preferred over FHEM and Home Assistant due to its broad support of devices, simple extensibility and main programming language, Java. The feedback that is given by the inhabitant directly, triggers the rule adaptation mechanism immediately.

4.3.2 Rule Adaptation

The previous section concerned about the feedback for adaptation and challenges originating by the analysis of hidden information in the event stream. While feedback is used to evaluate the performance of the system, adaptation is necessary to improve it.

In machine learning two approaches for adaptation are common: Optimize the current solution using optimization algorithm or simply forget what the system has learned and train it on new data. While in some cases it is important to never forget the learned knowledge - for example in a spam filter of an e-mail software - in this case it is preferred because it allows re-usage of implemented functionalities, and avoids being trapped in local minima by an optimization algorithm.

To guarantee a new solution for the next rule, the previous generated antecedent will be excluded from solution space. Exclusion although necessary to guarantee new solutions can be mischievous if the environment changes to a state where the previous solution would fit best. Therefore excluded solutions will not be stored physically on the system but will be used for the next generation of rules. This means that an excluded solution can be discovered every second generation.

Summarizing the previous decisions, when rule adaptation is triggered by the SH, at first all tagged rules are removed from the SH file system. Then for every deleted rule, an actuator is created. Afterwards the list of target actuators and their excluded antecedents are passed to the rule learning component to learn rules anew. The exclusions are not stored physically on the file system and will be forgotten after the process has terminated.

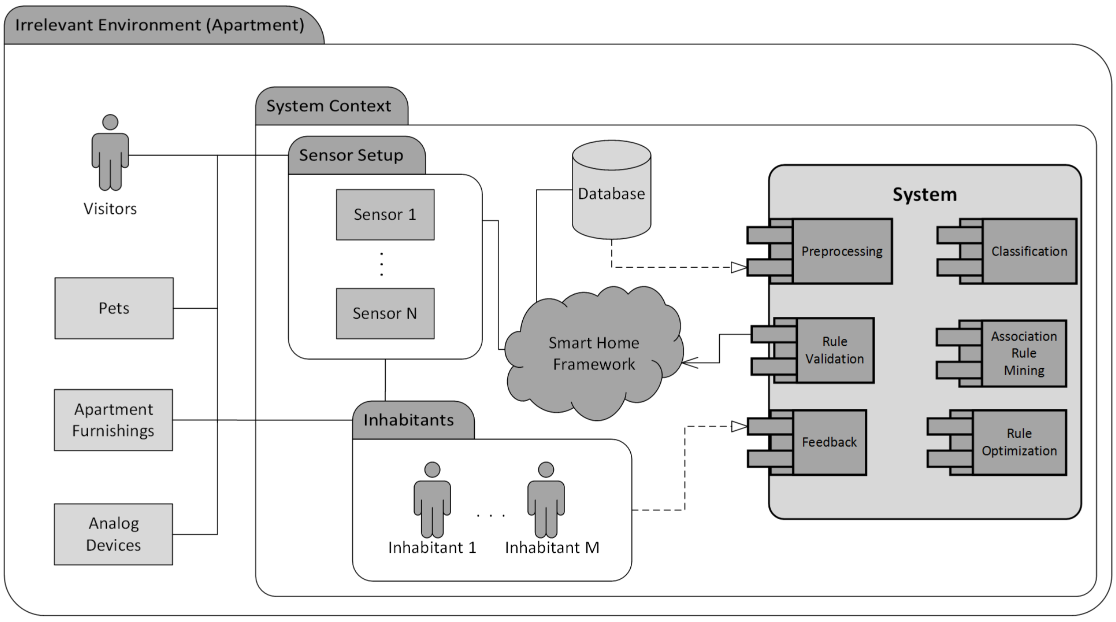


Figure 13: The overview of the system's context.

5 System Implementation

To proof the concept - that was proposed in the previous section - a prototype was designed and applied to a SH platform, in order to learn rules for actuator management. In this section its software architecture is presented, own implementations are differentiated from used frameworks and tools, and the interface to communicate with the system is described. An overview of the system's context is provided in Figure 13.

As mentioned in Section 4.3.1, the preferred target SH platform is openHAB (in its stable version 1.8.2). Three different interfaces for the learning system were developed (see Fig. 13), in order to communicate with the SH software. Usually the sensory data would be retrieved by a service from the SH. However, in the stated version of openHAB, no convenient interface is provided to retrieve all data as a single event stream, like it is stored by its persistence service. In addition, the HUI of openHAB provides no convenient interface to display and manage adaptation of past executed rules. Due to this, a GUI was developed as interface for the adaptation mechanism, that can be embedded into a HTML page. And finally the learned rules are exported into the SH by placing them in a specific folder location, missing the possibility to export them directly into the rule base of the SH.

All tools and libraries that were used or integrated into the developed system are introduced in Section 5.1. Followed by this, basic informations and the implemented design of the developed system, as well as settings are introduced in Section 5.2. After that, the implemented modules - represented by subsections of Section 4.2 - are described in Section 5.3. Although the implementation of the adaptation mechanism could not be realized completely due to several security related issues, a prototypical GUI for feedback is presented in Section 5.4.

5.1 Tools and Libraries

As highlighted in Sections 4.2.2 and 4.2.4, one major advantage of the proposed concept is - in contrast to the elaborated state of the art - its independence for specific algorithms. To proof the proposed concept and evaluate different compositions of ARM and CL approaches, an own implementation of several algorithms was unfeasible. Therefore two libraries - for ARM and CL - were integrated through interfaces to solve the respective subtasks.

SPMF

The first library is the Sequential Pattern Mining Framework (SPMF) [16], a data-mining library that contains algorithms for FPM, ARM and Sequential Pattern Mining. The library, used in its current version v0.99e, is open source distributed under the GPL v3 license and completely written in Java. Beside direct integration, the library can be used via command line or a GUI. However, since the algorithms are not implemented with a standardized interface, every algorithm has different starting parameters and may use a different data input format. This characteristic and its consequences are described in more detail in Section 5.3.

Weka

Since SPMF is only aimed for pattern mining, another library was required for the CL task in the rule learning process. Again, to enable a broad variety of different algorithms, Weka [5] was chosen for this task. The Waikato Environment for Knowledge Analysis (Weka) is a workbench for machine learning containing a collection of various algorithms for data mining tasks like CL, clustering, ARM or predictive modeling. It is continuously developed at the University of Waikato and open-source distributed under the GPL v3 license.

Although providing a standardized interface for ARM algorithms, the use of Weka as library for both approaches was rejected due to two reasons: Compared to SPMF only a few algorithms are implemented and - the main reason - the resulted rules for ARM and CL are only accessible as text and not as objects. Due to this, for each algorithm an interpreter is required, that takes the output string of the Weka algorithm, and restores all rules that are described in it. The interface at which the Weka algorithms are integrated is described in more detail in Section 5.3.

Other integrated software

As mentioned before, openHAB is missing a proper interface to load the event stream directly via a service. Therefore, since openHAB is configured to persist the events in a MySQL database, the official JDBC driver library for MySQL - called Connector/J - is integrated into the rule learning system to load the persisted data.

As stated in Section 4.2.5 and introduced in Section 3.2.4, the rule engine Easy Rules is integrated for validation and performance evaluation of the generated rules. The rule interpreter and integration of Easy Rules is described in more detail in Section 5.3.

To integrate the developed system into openHAB, the SH platform needs to be configured respectively. This task is supported by the openHAB Designer, an Eclipse RPC (Rich Client Platform) application that provides a full IDE including syntax checking, highlighting and content assistance of openHAB's runtime configuration files [62].

5.2 General Implementation of the System

As stated previously, the system should be designed to extend the functionalities of a SH not autonomously as separate service, but frequently as executable learning module. In addition, to develop platform independent the system was implemented in the programming language Java. Therefore the developed system resulted in a runnable Java Archive (JAR) file, that can be executed via command line and placed in the same folder as the SH.

To parametrize almost any functionality of the learning system, it was decided to utilize both: a configuration file and the command line of the operating system. The configuration file contains settings for the learning process, like the labels of target actuators, settings for ARM or CL, the database containing the event stream, the integrated rule engine and many more. A structured overview of all parameters in the configuration file, is given in Table 11 in the appendix. It should be noted, that the configuration file needs to be placed in the same folder as the JAR component.

Settings considering the function of the system, its targets and the dates that limit the retrieved data, are given directly to the system via command line parameters. The following functionalities are provided by the software in order to: learn, update, annotate, replace, remove or validate rules. These functionalities can be divided into learning (learn, update, replace) and managing (annotate, validate, remove) tasks. The difference between the proposed learning functions is not the learning process but the operations applied before or after it. While the function *learn* simply learns a new rule, *update* replaces a rule if a different one for the same virtual actuator is discovered that performed better than the current solution, and *replace* does almost the same with the slight difference that the discovered rule does not need to perform better than the current one but needs to surpass the performance threshold. Aside from learning, the managing functions should be self-explanatory. The *annotate* function tags a rule as wrong, which is either immediately or during the next update process replaced. The *validate* function validates a rule based on new data and updates the rule table according to the results and last but not least the *remove* function removes a rule from the SH and the rule table. With each execution only one functionality can be addressed by using the mandatory parameter *-mode* [value].

In order to apply the functionalities on a target, the mandatory parameter *-target* [value] requires either *all*, the label of an actuator or the identifier of an existing rule. In addition, an optional argument *-value* [value] can be passed to the system in order to focus the learning process to a specific virtual actuator. To limit the time frame of the data used for learning, two optional parameters - named *-from* and *-until* - can be passed to the system in order to limit the time frame from which data is retrieved. Last but not least, as described in Section 4.2.5, the system is capable to *retry* the learning approach several times on a different data basis in order to increase the rule discovery rate, and maybe the performance of the best rule using the optional parameter *-trials*.

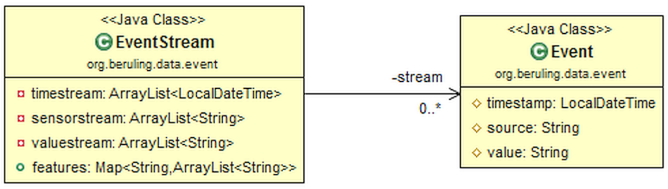


FIGURE 14: A class diagram of the structure from the internal event stream.

The following command line example would execute a learning process for the actuator with the label TA01 and its target value of 4.5:

```
java -jar LearningSystem.jar -mode learn -target TA01 -value 4.5 -trials 3
```

5.3 Implementation of the Learning Process

After some general informations about the developed system were introduced, the implementation of the rule learning functionality is presented in this section. Since the flow of the data was already defined in Section 4.2 the focus in this section is moved to the internal data structures.

Starting with preparations of the initial situation, the sensory data needs to be imported from a predefined database. In order to manage the resulted list of events, two classes were developed that represent a single event or a stream of events (see Fig. 14). As required from the concept, each Event class consists of a timestamp, a source and a value field. The class EventStream contains - beside the original persisted stream - one list for every column to ease repetitive processing in later steps.

Continuing with the preparations, the list of physical actuators is given to the system either manually or through predefined labels in the configuration file. As mentioned in the conceptual description, virtual actuators are determined by searching for unique values of actuator labels in the database.

To avoid preprocessing every virtual actuator separately, a Java Map object, consisting of virtual actuators as keys and the segmented sequences as values, is created and filled completely during one processing of the event stream. The list of segmented sequences is split into distinct sets for training and evaluation according to a configuration parameter. It should be noted that actuators, whose number of segmented sequences is below a given threshold, will be removed from the learning objective to avoid the discovery of useless

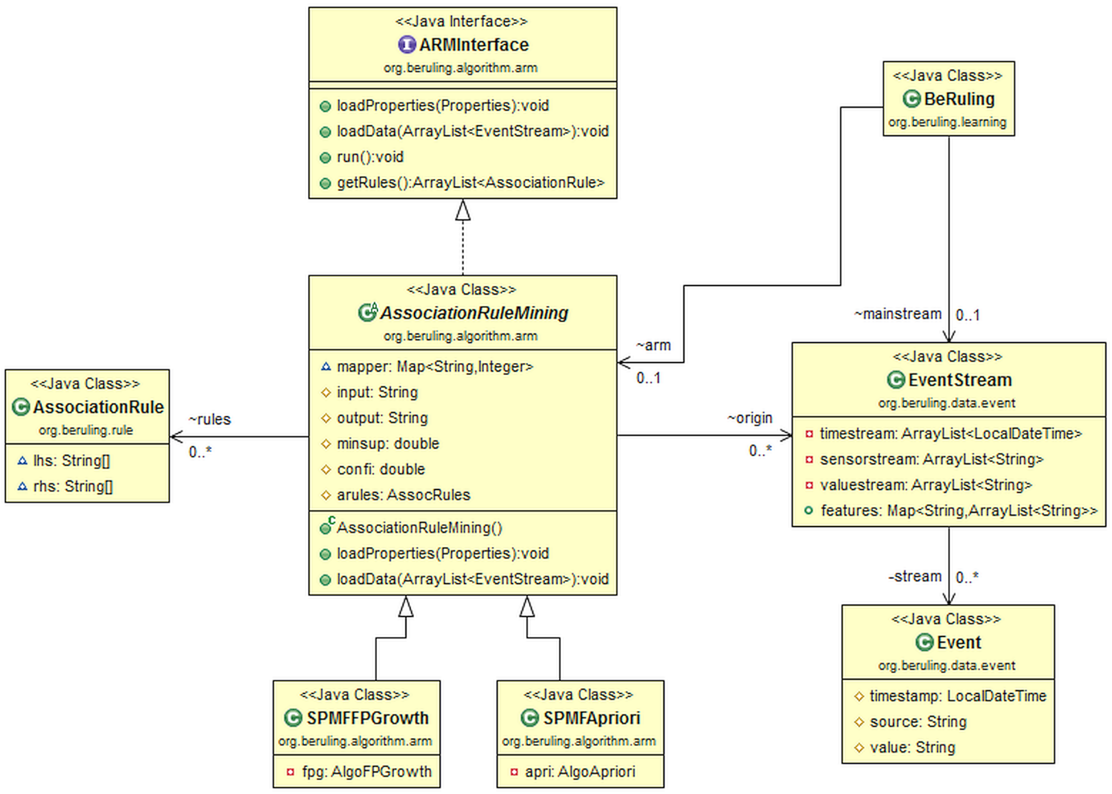


FIGURE 15: A class diagram showing the integration of Association Rule Mining algorithms.

rules. After the map is filled with all target actuators and their respective training and evaluation set, learning is performed for every virtual actuator separately.

According to the elaborated concept, behavior rule learning starts with ARM. The integration of SPMF into the learning system is presented in Figure 15. The specific ARM algorithm is instantiated during runtime and is defined in the aforementioned configuration file. Due to missing standardized interfaces in SPMF, each algorithm is integrated as a specialization of the AssociationRuleMining class. However, if a complete different library needs to be integrated, it is sufficient if the algorithms implements the ARM interface. Since the integrated Apriori and FP-Growth require both a transaction database, the list of EventStream objects is written in a CSV-format to the filesystem and then loaded via SPMF intern functions into a transaction database on that the algorithm is applied with preloaded configurations. Since the implemented algorithms of SPMF operate mostly on sequences of integers, a mapper is required that maps the specific sensor label to a unique integer value. This mapping is reverted during retrieval of the discovered association rules, and every resulted list of associated sensor labels are stored in an internal AssociationRule object.

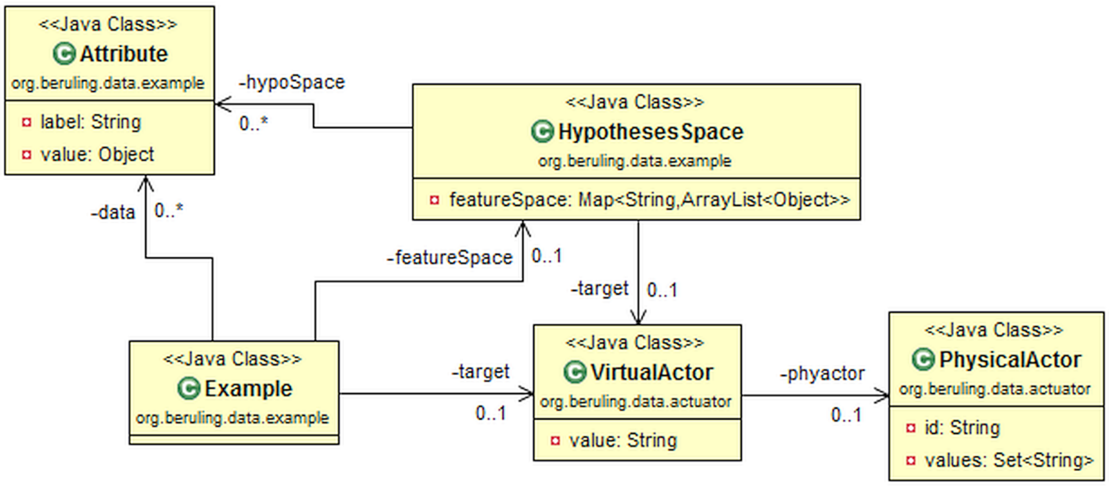


FIGURE 16: A class diagram of the data structure at which Classification is applied.

Following the concept, the output of ARM is a list of association rules. In order to apply CL on examples for every association rule, a HypothesesSpace object is generated first, that defines the feature space for the CL algorithm. Then training examples are generated using the Attributes of the HypothesesSpace (see Fig. 16), to extract the correct data from the sequences. A training example can be visualized as a row of Table 5 in Section 4.2.4. It is implemented as a list of Attribute objects (columns of the table) where the elements of the list are predefined by the HypothesesSpace object (the table header) and a target VirtualActor object (target class) at which the example is classified to. An Attribute object contains a label (the column header) and a value (the content of a table cell).

Like ARM, the algorithm for CL is instantiated during runtime and defined by the configuration file. However as mentioned before, the algorithms in the Weka library do not provide standardized accessibility to their respective classifier. Therefore each algorithm used in Weka requires a defined interpreter class, that transforms the resulted classifications into a set of Rule objects. Custom CL algorithms can be implemented using the classifier interface. The class structure of the integrated CL algorithms is visualized in Figure 17. The Rule object differs in such way from the AssociationRule object, that its antecedent and consequent consists of Condition objects. These objects represent an abstract triple, each consisting of the sensor label, an operator - like "$>$","$<$" and $=$ - and a value.

As described in Section 4.2.5, the performance of each rule is determined by an validation process, where it is applied on unknown SH emulated data by an interchangeable rule

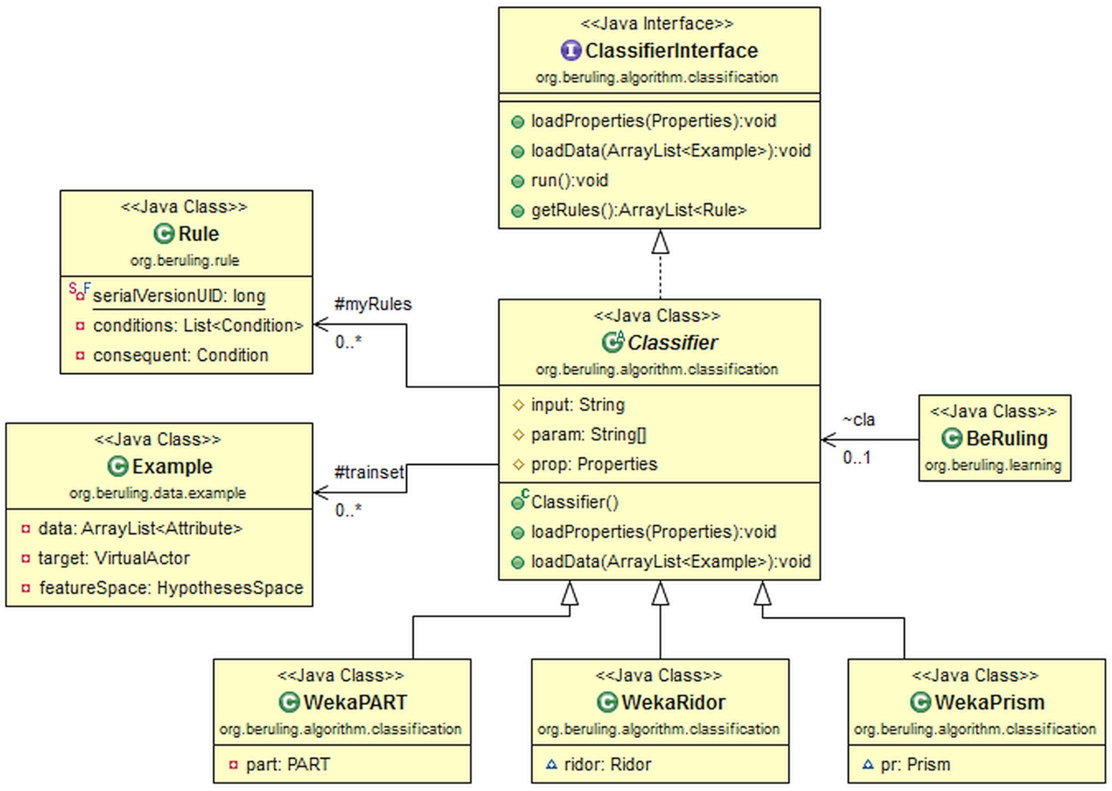

<<Java Interface>>
ClassifierInterface
org.beruling.algorithm.classification
loadProperties(Properties):void
loadData(ArrayList<Example>):void
run():void
getRules():ArrayList<Rule>
<<Java Class>>
Rule
org.beruling.rule
serialVersionUID: long
conditions: List<Condition>
consequent: Condition
#myRules
0..*
<<Java Class>>
Classifier
org.beruling.algorithm.classification
input: String
param: String[]
prop: Properties
Classifier()
loadProperties(Properties):void
loadData(ArrayList<Example>):void
~cla
0..1
<<Java Class>>
BeRuling
org.beruling.learning
<<Java Class>>
Example
org.beruling.data.example
data: ArrayList<Attribute>
target: VirtualActor
featureSpace: HypothesesSpace
#trainset
0..*
<<Java Class>>
WekaPART
org.beruling.algorithm.classification
part: PART
<<Java Class>>
WekaRidor
org.beruling.algorithm.classification
ridor: Ridor
<<Java Class>>
WekaPrism
org.beruling.algorithm.classification
pr: Prism

FIGURE 17: A class diagram showing the integration of Classification algorithms.

engine which is instantiated during runtime. The data is represented by a Context object that manages a set of Sensor objects and a Time object. While the sensory environment in a SH is emulated by the set of Sensor objects, the current time is represented by the Time object. Having the emulated data structure initialized, the events of every EventStream object are processed in ascending time order. If the sensor of an event is part of the rule's antecedent, its label and value are extracted and added to a set of unique Sensor objects. After the whole event stream is processed, this very set of sensors contains the state of every - rule relevant - sensor right before an actuator is executed. A visualization of this description is given through the class diagram in Figure 18.

While the input data for rule validation is defined, the internal rules are not suited for execution by a rule engine. To load the internal rules into the knowledge base of a rule engine, a parser needs to transform the conditions of the internal Rule object into a rule engine compatible format. The parser for Easy Rules is implemented as a POJO. It takes the conditions directly into its object and returns itself as rule.

Using the emulated data for validation, a rule engine executes the parsed rule on positive and negative environmental states. By knowing the respective situations in that a rule

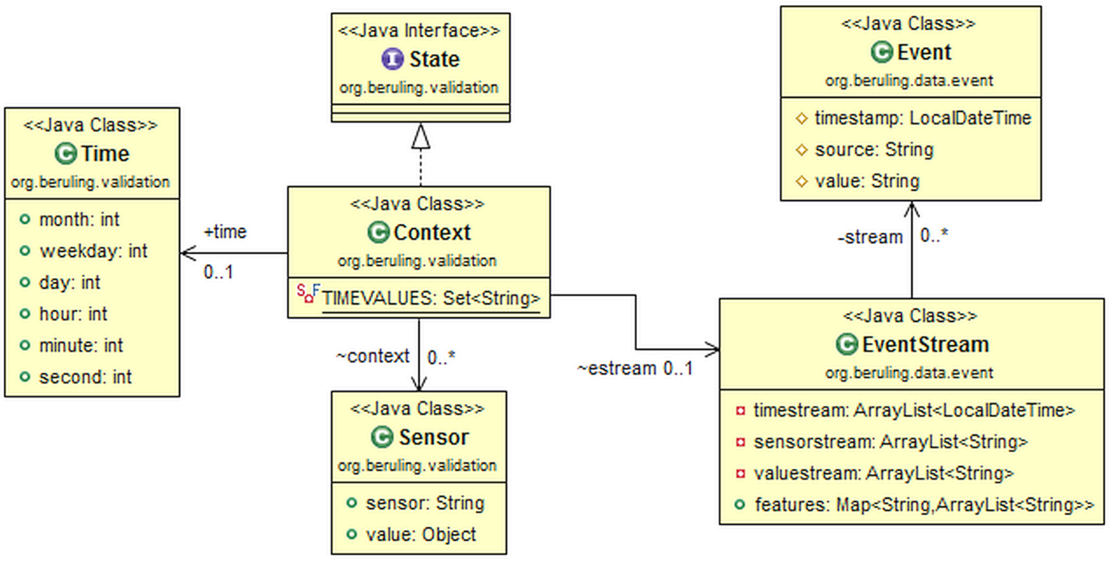


FIGURE 18: A class diagram of the emulated data structure used for validation.

should fire, a ConfusionMatrix object is filled with the binary classification results (see Section 4.1). Based on this, a Performance object is generated that contains the F_1-score of the rule. The score including several other informations like the timestamp, rule identifier and even the whole rule itself are stored into an Entry object. A class diagram embracing all these previous descriptions of the validation data structure, is given in Figure 19.

If the performance of a rule passed the predefined threshold, its internal Rule object will be parsed by another rule interpreter for the target SH platform and basic informations about it are stored according to the concept. The rule table is implemented as a RuleTable object. The class can be seen as a mask for a Java Property object, using predefined labels as keys and the data of the Entry objects as values. In addition, the RuleTable object can not only write Entry objects to a file, but also restore them from it and thus load exported rules back into the learning system. Before the program terminates, the RuleTable object and all passed SH rules are serialized to predefined locations.

5.4 Implementation of the Adaptation Mechanism

To adapt the discovered rules, two mechanisms - described in Section 4.3 - are implemented. The first mechanism - continuous update of the rule base - is implemented

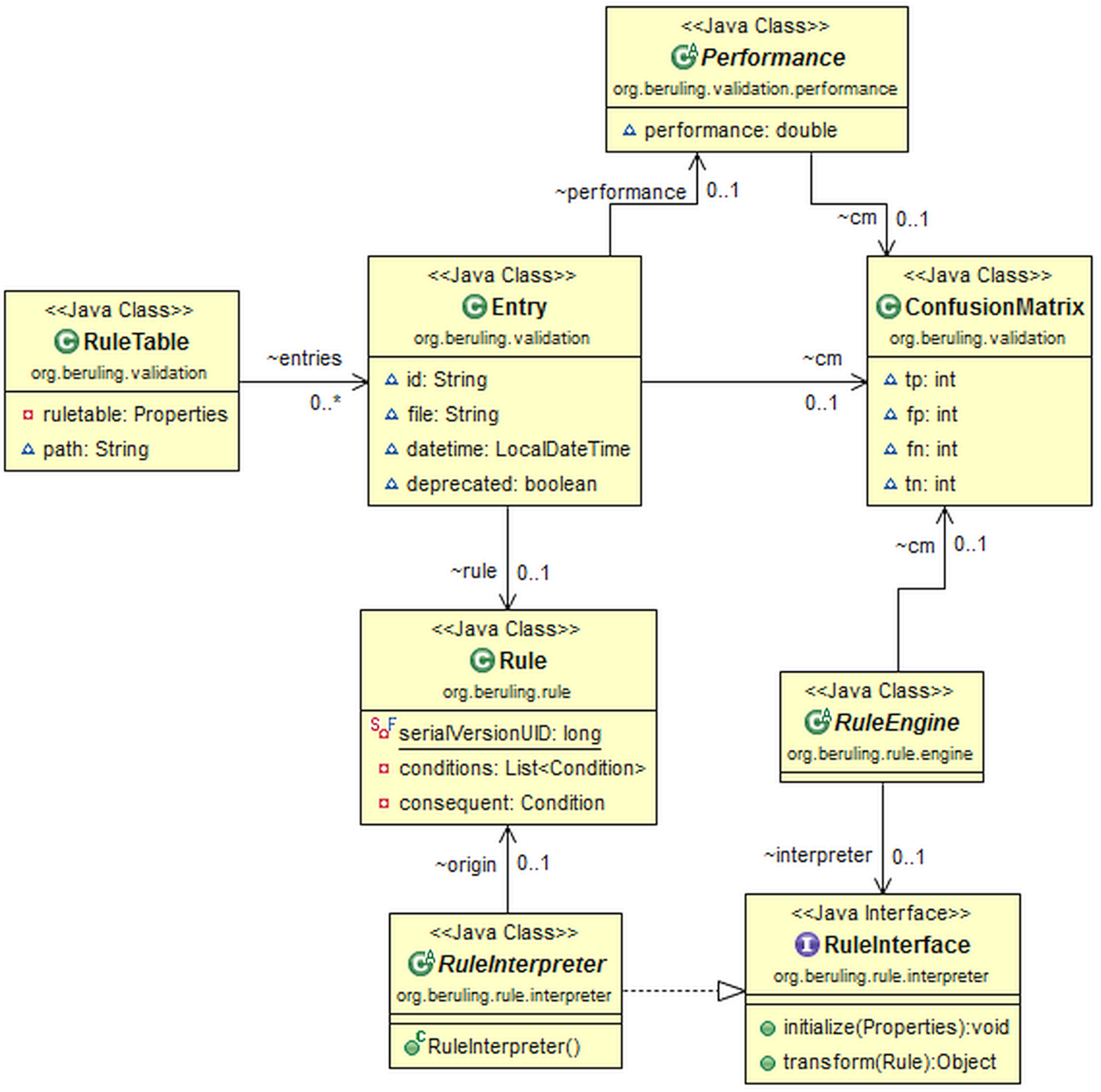
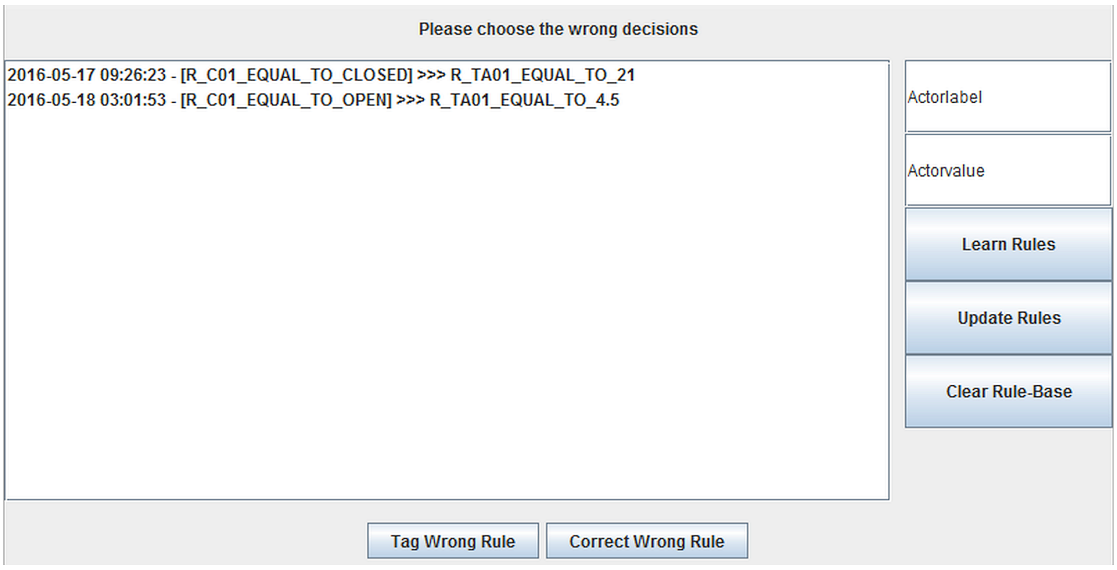


FIGURE 19: A class diagram of the internal validated data structure.

using openHAB's own configuration possibilities. The second mechanism - update on explicit feedback - is implemented as Java applet and embedded into the HUI of the SH.

Since the learning system does not run autonomously, a trigger is needed that commands it to update the current rule base. This trigger needs to be realized by an autonomous configurable system for which the SH software is predestined. By using the internal rule engine of openHAB, a timer was realized that executes the learning system and resets itself according to some point in the future. As mentioned previously, it is not predictable at which interval the system needs to update its rule base. Therefore its configuration is provided to the inhabitant using the HUI of the SH. In addition, the time frame of used data to update the rule base can be configured by this interface too.



```
2016-05-17 09:26:23 - [R_C01_EQUAL_TO_CLOSED] >>> R_TA01_EQUAL_TO_21
2016-05-18 03:01:53 - [R_C01_EQUAL_TO_OPEN] >>> R_TA01_EQUAL_TO_4.5
```

FIGURE 20: The implemented Graphical User Interface for feedback.

Until now the SH system uses the functionalities of the learning system to update the discovered rule base frequently. However, this does not include single rules or the feedback of the inhabitants. To gather direct feedback, a Java applet was developed that presents past actuator executions of the generated rules and the possibility to annotate, update, replace or remove the respective rule behind the execution. A screenshot of the prototypical GUI is given in Figure 20. Since the applet needs to know where the rule log file is located - which is defined in the configuration file - it should be placed in the same folder as the learning system.

Although running in the IDE, it should be noted that the applet could not be embedded successfully into the HUI of the SH. The main reason for this is that in the recent history, several critical security issues were discovered in Java and due to is, almost any webbrowser is blocking the Java applet by default [48]. Even a self certified applet was blocked, and the option to disable these security checks was removed from the Java console since Java 7.

6 Evaluation

In order to proof the concept, the developed prototype was evaluated under varying settings. In this section, the evaluation is presented starting in Section 6.1, where the experimental setup is introduced including a description of the apartment, the installed sensors and actuators, used computers and the general configuration of the SH. To determine a good configuration of learning approaches for the proposed system, the algorithms - introduced in Section 3.3 - were tested under varying parameters. The results of this competition are described in Section 6.2. Using the results of the previous competition, the evaluation of the learning system, its configuration, objectives and results are discussed in Section 6.3.

6.1 Experimental Setup

Evaluation is applied on data from a real world experiment. In this experiment, a common apartment was is enriched with devices and computers to model a common SH. The objective of the experiment was to learn behavior rules that resembles the inhabitants actuator management. After a failed installation of light actuators due to the old circuit in the building, a heating thermostat was chosen as target actuator for its simple installation procedure. Upon request of the inhabitants, it was revealed that the only frequently used radiator in the apartment was in the toilet. Hence this experiment is focused to learn behavior rules for the heating management in the restroom.

In this section basic informations about the apartment, installed sensors and actuators, configurations of the CCU and further hard- or software installations are described in order to reproduce the experiment. During the experiment the apartment was inhabited irregularly by one to three people. They were told to behave naturally and use the actuator in their own behavioral pattern.

Beside this work, a parallel thesis from Peter Manheller [61] aimed to detect anomalies in human behavior using the data generated by the SH. For differentiation it should be noted that the toilet setup including the heating actuator is detached from the rest of the apartment, because it was installed few months after Manheller's start. While this work relied on the toilet setup as basis for development, Manheller's project used data

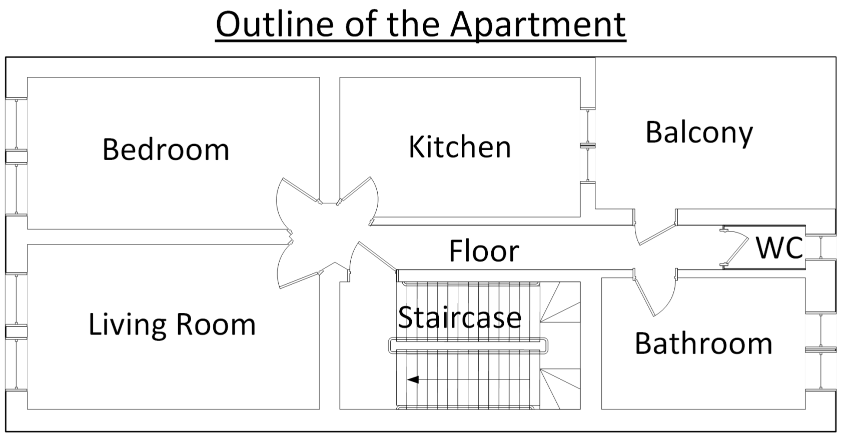


FIGURE 21: A rough outline of the apartment used for the experiment.

from the rest of the apartment. Nevertheless, because data of the whole apartment were used for evaluation, the sensory setup of both projects will be considered in this section.

6.1.1 Apartment Description

The SH experiment was conducted in a two-room-apartment of an old building with no previous installed SH devices. The distribution of rooms in the apartment resembles a star topology - with the floor as center of the apartment - as shown by a rough outline of the apartment in Figure 21. A more detailed outline of the apartment including the approximate positions of every sensor is presented in Figure 33 in the appendix. Due to readability, the toilet is outlined separately in Figure 32 in the appendix. By entering the floor through the front door, the inhabitants can reach any room, while at the same time, needs to pass the floor when they move between the rooms.

The Balcony is the only part of the apartment that has no sensors installed because it is shared with a neighboring apartment that is excluded from the experiment.

6.1.2 Sensor and Actuator Setup

Due to the age of the building, privacy considerations and the available hardware, only a few selected devices could be installed in the apartment. To avoid (de-)installation difficulties of the devices, one requirement was to rely only on external attached sensors and actuators. To bridge the gap between installation simplicity and usability for behavior observation, most attention was given to motion, contact and temperature sensors.

Type	Device	Amount	LV*	BD*	FL*	KI*	BT*	WC*
Motion	HM-Sec-MDIR-2	6	1	1	1	1	1	1
Contact	HM-Sec-SC-2	5	-	-	1	2	1	1
Temperature	HM-WDS40-TH-I-2	1	-	-	-	-	-	1
Heating Control	HM-CC-RT-DN	1	-	-	-	-	-	1
CCU	HomeMatic CCU2	1	1	-	-	-	-	-

Table 7: An overview of all installed devices in the Smart Home.

* LV = Living room, BD = Bedroom, FL = Floor, KI = Kitchen, BT = Bathroom and WC = Toilette.

In addition, to simplify the installation procedure, all devices are from the same manufacturer and applied to a CCU. Due to availability, HomeMatic devices were used and connected to HomeMatic's CCU2. The CCU2 although not a sensor or actuator, is used as central access point for the openHAB software. An overview of the installed devices is given in Table 7.

Motion Sensors

In order to detect presence in the whole apartment, each room was enriched with exactly one HM-Sec-MDIR-2 motion sensor, that observes most of the room space that connects to the floor. Exception is made for the bedroom. Considering the privacy of the inhabitants, the view angle of the motion sensor in the bedroom was modified to see only the area directly connected to the door. In general the motion sensors were placed in such way, that only the own room can be observed in order to produce independent sensor event. However exception in this case in made for the floor. Due to the apartment structure, regardless how the motion sensor is placed, it can always look into a least one other room. In this experiment the sensor was placed between the living room and the bedroom which enables the motion sensor to detect presence in the kitchen. This needs to be considered when the independence of events in data is required.

Beside motion, the sensor is capable to capture the brightness of its environment. However the sensor in the toilet is the only one that used this functionality.

Contact Sensors

The usage of contact sensors was limited to the available devices. Therefore they were applied on objects that state a clear semantic usage. The sensor in the floor was attached to the front door, which indicates that a person left or entered the apartment. The two sensors in the kitchen were installed at the refrigerator door and the cutlery drawer to indicate when an inhabitant is preparing or eating a meal. The sensor in bathroom was attached to its door, because of the inhabitants' habit to close it when they take a shower or enjoy a bath. Finally the sensor in the toilet was installed at the window in order to observe the behavior concerning ventilation of the toilet.

Temperature Sensors

In order to observe the managed domestic feature as stated in Section 4.1, the sensor HM-WDS40-TH-I-2 was placed at the toilet near the window to update frequently the current temperature and humidity of the room. Beside the external device, an additional temperature sensor - integrated into the heating actuator - is used to observe the current temperature. To ensure an accurate observation both sensors are placed in the opposite corner of the room.

Heating Actuator

The HM-CC-RT-DN radiator thermostat is installed at the heating in the toilet to observe the behavior of the inhabitant concerning the heating management. Although the device has many functionalities, it was only used as a manual heating controller in order to observe casual behavior of the inhabitants. Beside the aforementioned temperature, the device observes the state heating valve and the current heating mode too.

6.1.3 CCU Configuration

As mentioned before, the SH software and therefore the *real* CCU of the SH is openHAB. However, the common approach for openHAB to communicate with eQ3 devices is over the HomeMatic CCU. Due to this, the CCU of HomeMatic is used in this experiment only as an access point that operates transparently by communicating only data between openHAB and the installed sensors and actuators.

In order to the learn casual behavior of an inhabitant in a common home, no automatisms - like rules or scripts - were used for the actuator during the experiment. In addition, the remote access for the heating controller was prevented, leaving the only possibility to execute the actuator manually. Last but not least, openHAB was configured to store an Item on a local MySQL database each time its value changed.

6.1.4 Further Hard- and Software installations

The aforementioned openHAB software was installed on a Raspberry Pi2, connected over Ethernet to the HomeMatic CCU2. However, in order to not disturb the SH experiment, a different computer - with 4Gb RAM and an Intel Core i3-2120 CPU - was used for evaluation. In addition, to compare performances on both setups, two MySQL databases were set up locally, containing data from either one of the two parallel experiments.

6.1.5 The Smart Home Data

Summarizing the previous setup as datasets, Manheller's data consists of five motion and four contact sensors, that were installed in every room except the toilet. Therefore only binary values were captured. The dataset contains continuous sensor data from several months. As mentioned before, the dataset does not contain completely independent sensor events which should be considered when its used. This dataset is in the following referred to as apartment dataset.

Since this experiment started later, another dataset was created consisting of data from the four devices, that were installed at the restroom. From these devices, nine different sensors captured data about the environmental state of the toilet continuously for two months. The dataset consists of observations from a motion, a brightness, a contact, a humidity, a valve state and two temperature sensors. In addition, it contains the actuator mode and the target temperature from the heating controller. This dataset is in the following referred to as restroom dataset.

Both datasets are published open source and available at the MCLab [57] of the Bonn-Rhein-Sieg University of Applied Sciences.

6.2 Screening of Algorithms

Since the composition of ARM and CL covers many different algorithms, a good working combination of the integrated approaches needs to be screened in order to evaluate a functional learning system. In this section, the evaluation of the two introduced ARM algorithms Apriori and FP-Growth of the SPMF library, and the three CL algorithms PART, PRISM and Ridor of the Weka library, are discussed. For each RI paradigm, the *best* is chosen in order to evaluate the learning behavior of the implemented system. The respective best approach is determined by the evaluation criteria in the following section.

6.2.1 The Evaluation Criteria

The most important criteria is the resulted performance, looked from three angels: the performance of the best rule, the average performance of all learned rules and the performance of the worst rule that was discovered during one execution. This criteria is evaluated for both: ARM and CL approaches.

Another important aspect for evaluation is the number of discovered rules. Regardless of the best, average or worst rule, a good algorithm should consider not too few or too many algorithms, because it could miss a good solution or waste resources in validating meaningless rules. As the previous criteria, this evaluation is applied on ARM and CL algorithms.

Last but not least, since the ARM algorithms from the SPMF library require the configuration of support and confidence, several settings were evaluated experimentally in order to determine the most promising combination of settings. This evaluation was not applied to the CL algorithms of the Weka library due to two reasons: First the library provides default values for every algorithm avoiding the necessity to configure them. Second and more important, the list of parameters for a classifier can have much more than two features, which increases the complexity of finding the optimal setting exponentially. Since the target of this screening is to discover a good composition of algorithms and not the best, solving this optimization problem would be out of scope for this thesis.

6.2.2 Evaluation of Apriori and FP-Growth

In Figure 22, the results of FP-Growth are confronted with Apriori's using a boxplot. Throughout all runs, the performance of the rules has varied through the whole spectrum from zero to one, which causes can not be derived directly. However, the important value which should be considered is the median, represented as a thicker horizontal line. The worst results aside, FP-Growth has outperformed Apriori in best and average results,

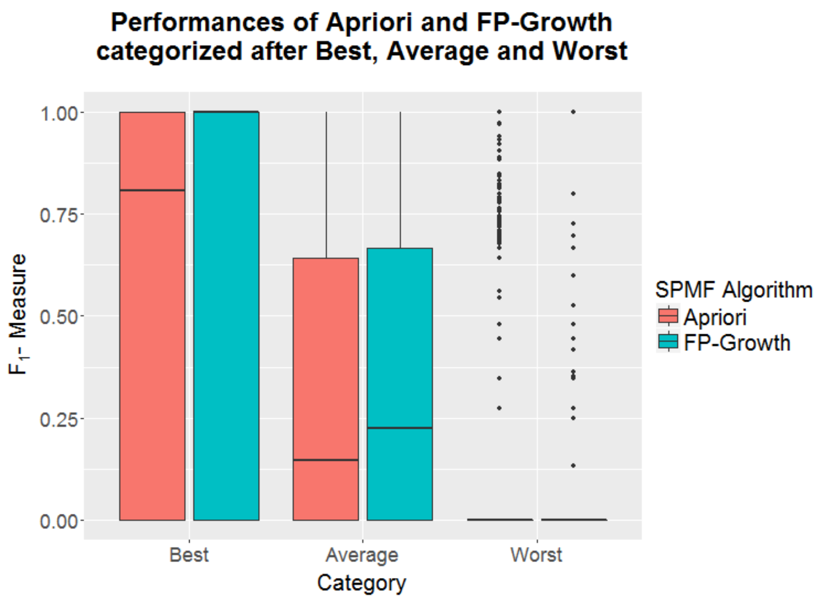
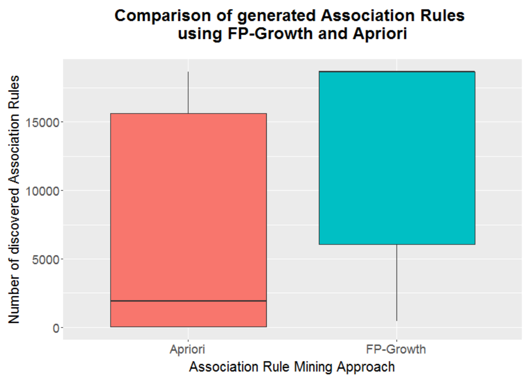


Figure 22: A comparison of performances with Apriori and FP-Growth.

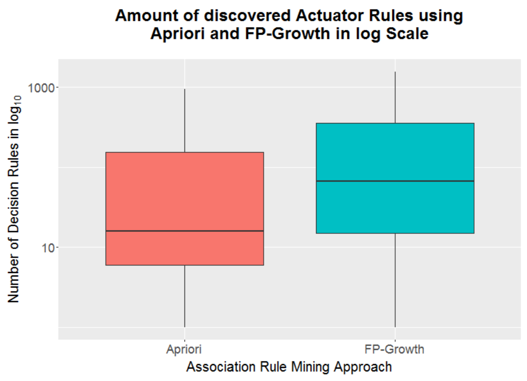
which means that the F_1-Measure was in general higher when using FP-Growth as ARM approach than Apriori. These results are surprising, because it was assumed that both algorithms would discover nearly the same association rules which should then again lead to similar results. The huge difference between the results of these two algorithms is visualized in Figure 23.

Considering the amount of discovered association rules as shown in Figure 23a, FP-Growth has generated almost ten times more association rules than Apriori. Although many of them were filtered out during postprocessing, the superior amount of association rules, resulted in general to more decision rules which is confirmed by Figure 23b. The more rules are generated from the data, the higher the possibility is that one good rule is discovered. In addition, the higher number of discoveries influences the median which clears the changed difference between best and average results. Nevertheless, this does not explain the still striking difference of the average performances between Apriori and FP-Growth fully.

Elaborating the results further led the indication, that FP-Growth may be more robust against changes in parameter than Apriori. This assumption is affirmed by Figure 24a, in which both ARM approaches are confronted under different Support configurations. While FP-Growth shows similar results under different settings, the performance with Apriori drops with higher Support configuration drastically. This behavior can be explained through the Apriori Property introduced in Section 3.3.1 which states that all



(A) A comparison of discovered Association Rules using Apriori and FP-Growth.



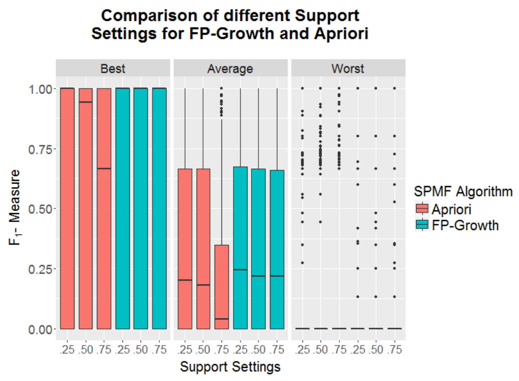
(B) A comparison of discovered Decision Rules using Apriori and FP-Growth.

FIGURE 23: The confrontation of Apriori and FP-Growth in terms of discovered rules.

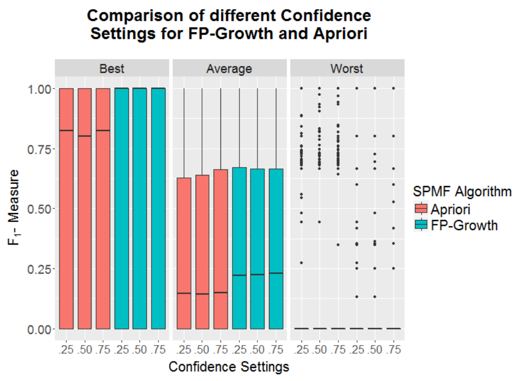
subsets of a frequent itemsets, need to be frequent too. The result of this constraint is that high support parameter will filter out potentially good frequent pattern very early resulting in lesser association rules and thus - in general - to an inferior performance.

Another surprising behavior was observed at the FP-Growth. Considering the definition and functionality of the Confidence parameter, it should have increased the performance of the discovered rules for both ARM approaches. However Figure 24b reveals that compared to Apriori, the Confidence parameter had no significant influence on the performance of the generated rules. Since both approaches generate association rules from frequent itemsets at the same way, a difference in implementation is not responsible for this behavior. One possible reason would be that so many association rules were generated by FP-Growth with every configuration, that good and bad rules were balanced and no real improvement could be reached. However the true reason behind this needs to be evaluated in future work, using a more complex learning problem.

Summarizing the comparison of Apriori and FP-Growth, the first one is outperformed by the second in terms of performance and robustness. In addition the most promising configuration of the Support and Confidence parameter is for both 0.25. Therefore the evaluation of the learning system will be configured using FP-Growth with the aforementioned settings. More information regarding the performance of every configuration is given in Table 10 in the appendix.



(A) The influence of the Support parameter in Apriori and FP-Growth.



(B) The influence of the Confidence parameter in Apriori and FP-Growth.

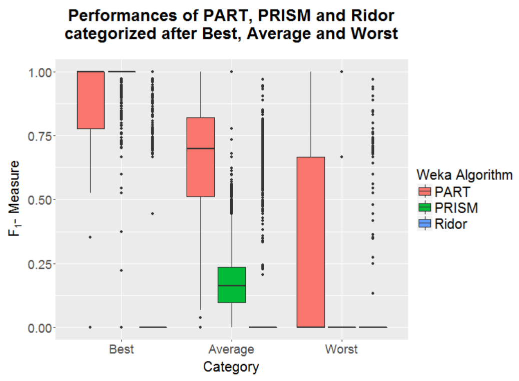
FIGURE 24: The influence of Support and Confidence at the performance of discovered rules.

6.2.3 Evaluation of PART, PRISM and Ridor

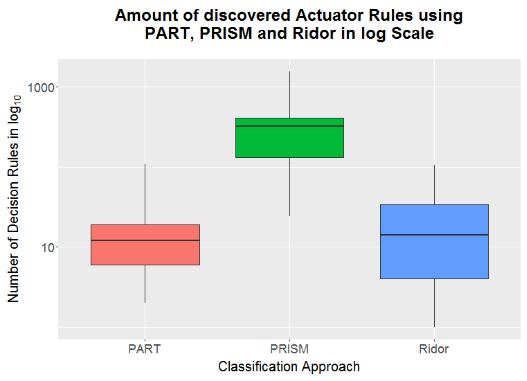
The evaluation of the Classifier is not straight forward compared to ARM. As visualized in Figure 25a the least performance is clearly reached by Ridor as Classifier, where even the best rules reached only a median of zero. From there on bargaining starts. While PART clearly outperforms the PRISM classifier in terms of average performance, only outliers of PRISM reach worser results that the perfect score of 1.0 where PART reaches in more than 3 of 4 runs not a perfect score. The causes of these results can not be traced completely, because both algorithms implement a complete different approach.

As for ARM, one possible cause for the change in these results can be derived when the number of discovered rules is considered as visualized in Figure 25b. As mentioned in Section 4.2.4, for every association rule, several decision rules are extracted from the classifier according to the resulted classifications. However, PRISM generates so many decision rules, that the set of rules contain almost every time an antecedent that suits the evaluation data perfectly. While the best rule strikes the statistic, the average takes all failures into account which are also many and thus resulting in a much lower median than PART.

Considering the amount of rules to validate and the average performance, PART is superior to PRISM and Ridor because it discovers - in general - the least rules without loosing much performance. As mentioned before the parameter for the CL algorithms are not evaluated due to the exponentially increase in complexity and thus evaluation duration. In addition the results using default parameter by Weka may not be optimized,



(A) A comparison of the performance results using PART, PRISM and Ridor.



(B) A comparison of generated amount of decision rules using PART, PRISM and Ridor.

FIGURE 25: The Comparison of the PART, PRISM and Ridor algorithms with respect to performances and discovered rules.

but sufficient to discover good decision rules. Therefore the settings for the evaluation of the learning system will be configured to use the CL algorithm PART, with its default parameter provided by Weka.

6.2.4 Final Annotation

The implemented algorithms for both ARM and CL - except for Ridor - can solve the learning problem magnificent. The results are comparable and a *best* setting of algorithms and parameters is emerged. However for both, ARM and CL the sheer amount of discovered association rules and decision rules influences the performance of the best rules which makes especially this category very biased. Therefore, this comparisons can not be seen as a general performance evaluation of FP-Growth and Apriori - or PART, PRISM and Ridor - but a context specific experimental screening of selected ARM and CL algorithms.

The results of this evaluation should not be used for reference, and their only purpose was to emerge a good combination and configuration of algorithms at which the behavior of the learning system will be evaluated and discussed in the following section.

6.3 Behavior Learning System - Evaluation

In this section, the evaluation of the developed learning system is presented. In order to determine the limits of the system based on statistical confirmed results, it was executed

General	Value	Algorithmic	Value
actors	1	**ARM**	FP-Growth
actors.0	R_TA01	**arm.support**	0.25
performance	F-Measure	**arm.confidence**	0.25
performance.min	0	**cla**	PART
minsetnum	2	**cla.param**	default
engine	Easy Rules		

Table 8: Some static settings of learning related parameters for evaluation.

under various different configurations, ten times in each case. In Section 6.3.1, the basic settings and varying parameters are described, under that the system was executed. After knowing the configuration, the objectives of the evaluation are introduced in Section 6.3.2. Finally, the results are discussed with respect to every objective, in Section 6.3.3.

6.3.1 Configuration and Scenarios

As mentioned in Section 5.2, almost every function of the developed system is parametrized, resulting in a huge variation of settings. To simplify the evaluation, only the few selected parameters of Table 8 were configured static. A short description of every parameter is given in Table 11 in the appendix. In addition to these static settings, several parameters where changed during execution:

- **-until**: Since behavior could change over time due to changing outdoor temperatures, this parameter seemed necessary in order to evaluate the system at different seasons. It was configured at the middle and the end of April and May resulting in the four configurations: 2016-04-15, 2016-04-30, 2016-05-15 and 2016-05-31.
- **-days**: This parameter defines the time frame between the earliest and latest database entry in days, that will be used for learning. It is used to derive the *-from* parameter for the learning system by using the given *-until* parameter. This parameter was configured as 14, 28 and 56 days respectively during evaluation.
- **-cvsplit**: As mentioned in Section 5.2, the *-cvsplit* parameter was labeled after its functionality to split a given dataset into the respective sets for training and evaluation and thus simulating cross-validation. During evaluation this parameter was configured as 0.5 and 0.75 in order to evaluate the influence of cross-validation on the results.

- **-trials**: As described in Table 11, this parameter defines the attempts of the system to learn rules. Redundant rules will not be added to the results, but all other new rules that passes the threshold will be exported. For evaluation purpose this parameter was configured at 1 and 3 attempts.

In order to evaluate the learning system in a real situation, two different SH scenarios were created under which the system was evaluated using the aforementioned static and variable parameters.

The first scenario resembles a SH - optimized for learning - in which all sensors are somehow related to an actuator. For this, the restroom dataset is used. In this dataset the target temperature from the sensor - labeled as R_TA01 - is used as actuator.

In contrast, the second scenario extends the first by using all sensor data from the SH, regardless of localization or relation to the actuator or other sensors. This scenario resembles more a common SH in which many sensors can be completely unrelated to an actuator. For this the aforementioned apartment dataset is merged with the restroom dataset.

In order to easily switch the scenario during evaluation, two MySQL databases were set up which contained each one of those datasets.

6.3.2 Evaluation Objectives

Compared to the previous screening, this evaluation does not primary aim to evaluate the performance of the discovered rules. Since the learning system is designed to extend a SH framework, this evaluation aims to emerge circumstances under that the system can learn rules and under which it struggles to find any rules.

Therefore, in order to guide the evaluation, several questions were formalized that can be separated into two categories: learning and dataset focused. As named in the first category, the evaluation is focused on the learning behavior of the proposed system. Its main purpose is to highlight difficulties, unexpected behavior or advantages in learning that emerged from the underlying concept. Although evaluation of the learning behavior is also considered in the second category, those question are focused with subject to the used dataset and conclusions originating from this evaluation should be comprehended carefully.

The following five questions are focused on the learning behavior:

1. Is there a significant difference between an optimized and a common SH in rule discovery rate and performance? With this question, the noise robustness of the proposed system will be evaluated.
2. How does the number of available sequences influence the rule discovery rate and performance? Since it is not assumed that the system will learn with few examples very good rules, in this question a convergence point at which the system can learn useful rules will be determined.
3. Does the *trials* parameter of the system influence the rule discovery rate and performance significant? With this question, the benefit of the *trials* parameter will be evaluated.
4. Does the system learned all virtual actuators equally well? Two objectives led to this question: First the distribution of usage for the installed actuator is considered and second its influence on the learning system is evaluated.
5. How does an increase of training data while decrease in evaluation data influence the rule discovery rate or performance? Since the data collected during the experiment would not be considered as *Big Data*, this question aims to evaluate the learning behavior of the system, if more data would be available for training at the expense of evaluation.

In addition three questions will be evaluated that are focused on the dataset:

6. Are there significant differences in learning the actuator using data from April compared to data from May? Since the experiment was performed for a longer period of time, one of the most interesting results is to evaluate, if the learning behavior of the system - what it learned exactly - changed with the behavior of the inhabitants.

7. How does the amount of days - used from the dataset - influence the performance and discovery rate and based on this, how many days should be used for learning in general? Background for this question is, that an approximate lead time is determined that the system requires to learn something valuable. Compared to the previous learning time evaluation, in this question the behavior of the inhabitants is considered additionally in order to evaluate the learning system under a real use case.

8. How good are the rules really? Can they be used directly in a SH? Since until now the rules were evaluated only on data basis, in these final two question the real value of the learned rules is evaluated. In order to answer them, the top ten most discovered rules are evaluated manually with consideration of their application in a common SH.

6.3.3 Results and Discussion

Using all configurations and repetitions, the learning system was executed in total 1920 times and exported during these runs 20160 rules. Within each run, several statistical values were captured like performance of the best rule or number of discovered rules for a virtual actuator, and stored into a single dataset. This dataset is unpublished but open source available on request.

Using all data available, and without any expectation of the results, the median of the best performance for all rules was surprisingly zero. After further analysis, one of the first results from this evaluation was, that a specific amount of sequences is required for the learning system in order to not only to learn rules, but also to discover well performing ones. However, the problem did not originated only from a small dataset.

The cause of this result can be understood, if the distribution of the dataset sizes is considered as visualized in Figure 26. The amount of sequences in a dataset is the result of the segmentation approach, that generates a sequence for each occurrence of the virtual actuator in the event stream. However, if only few events of the target actuator are stored in the database, the resulting dataset for training would also contain only a few sequences. Imagine the management of a heating actuator that ranges from 4.5 to 30.0°C in steps of 0.5°C. It has a broad range of 31 possible states resulting in the same

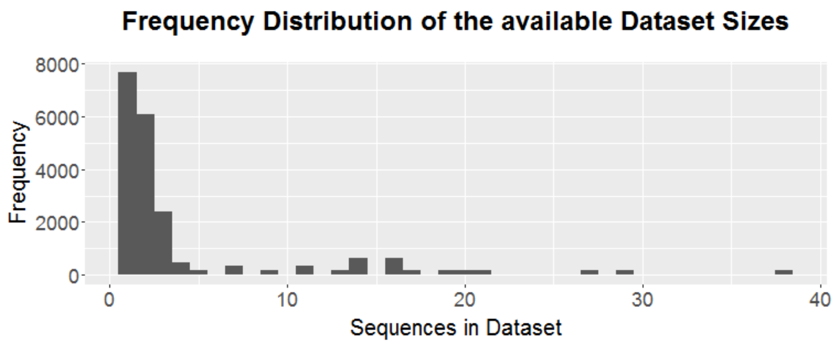
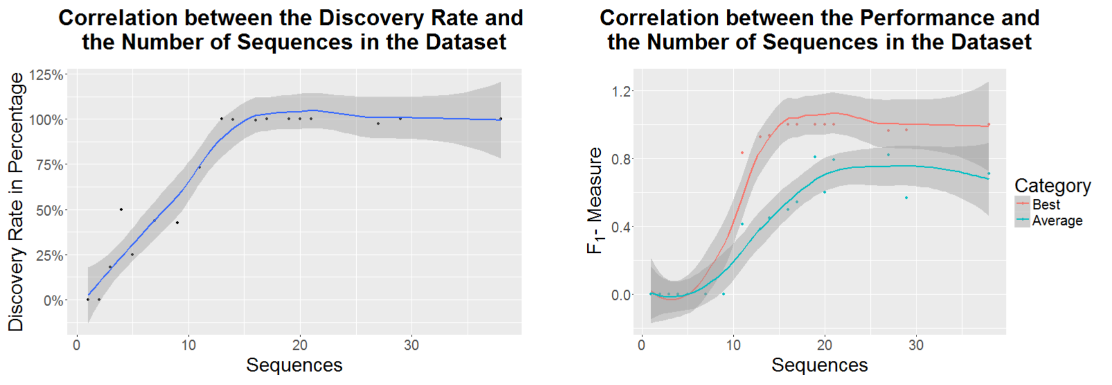


FIGURE 26: The frequency distribution of the dataset sizes used for learning.

amount of different virtual actuators for that rules could be learned by the system. The equal distributed usage of all states will most probably never happen, since the room temperature depends also on various other parameters, like opened windows or outdoor temperature or the seasons. In addition, these states can be set unintentional, due to a change of the aforementioned parameters or any other reason, resulting in many virtual actuators that were executed only a few times during the whole experiment. This reasoning was confirmed by further analysis of the data and consultation of the inhabitants. One solution that may reduce the problem, is to cluster the configured temperatures into fewer groups like {Low, Medium, High}. However, this is not always necessary as shown later in this section.

Evaluation focused on the Learning Behavior

Getting back to the topic, very small datasets were biasing the evaluation, because the percentage of datasets containing 1 to 3 sequences was already 80.15% of all datasets. In order to remove them from the results, a definition of *small* is required. Considering the second question, the solution to this problem is directly related to answering the question. Therefore, the median for the rule discovery rate (27a) and the best and average performances of discovered rules (27b) - with the respect to the dataset size - were calculated as visualized by Figure 27. The correlation between small datasets and the learning behavior is striking. Not only does it confirm a dataset of approximate 15 sequences is required for the system in order to discover good results, it also reveals that the minimum size of the dataset should not fall below 10 and that a convergence occurs after 15 sequences.



(A) The influence of the dataset size on the rule discovery rate.

(B) The influence of the dataset size on the performance of the rules.

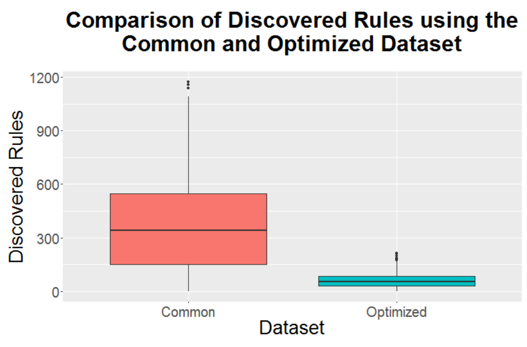
FIGURE 27: The correlations between the dataset size and the learning behavior.

As a result, the data for evaluation was adjusted by removing all results based on datasets with less than 10 sequences. From the previously mentioned 20160 rules, merely 2880 were left, that were used for the following evaluation.

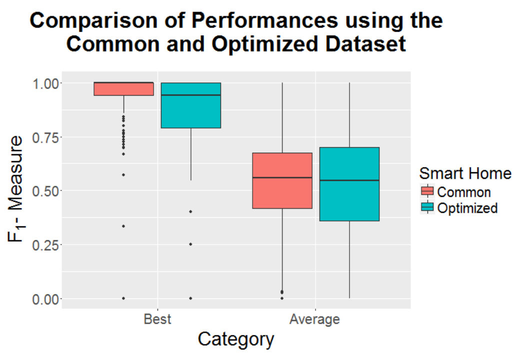
One of the most interesting question is the first, because it addresses the usability of the proposed system in a common SH. Due to the segmentation concept, the system is expected to be robust against noise and assumed to perform similar to the optimized scenario. Considering the performance of the discovered rules in Figure 28b, there exist - as expected - no significant difference between an optimized and a common SH.

However, Figure 28a reveals a significant difference in the number of discovered rules. At first this seems reasonable because using more sensors, increases the number of association rules and thus the discovered rules in total. Nevertheless, a difference of this scale was not expected, because only nine sensors were added to the dataset. This can turn into a problem if the SH encompasses many sensors but the computational resources are limited. One solution for this problem, would be to filter out association rules between sensors, who are not really related to each other, but in the data appearing so. However, this would require prior knowledge about the SH setup, which needs to be given manually or determined from the data first.

Continuing with the third question, it was expected that an increase in learning attempts will also influence the rule discovery rate and the general performance of the best rules positively, since the learning system has more chances to discover rules on different training and evaluation sets. However, first results stated that there were no differences between one or three attempts, leading to the reasons that the learning problem may be



(A) A Boxplot of the discovered rules for both scenarios.



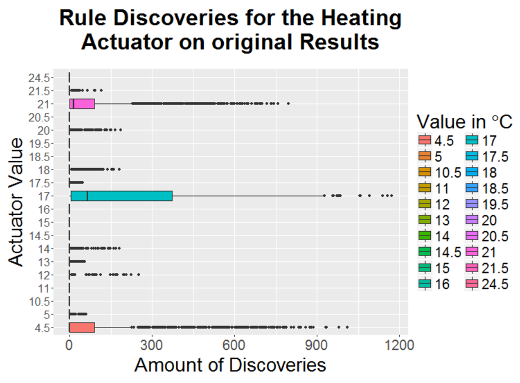
(B) A Boxplot of the performance results for both scenarios.

FIGURE 28: A comparison of results using optimized and common datasets.

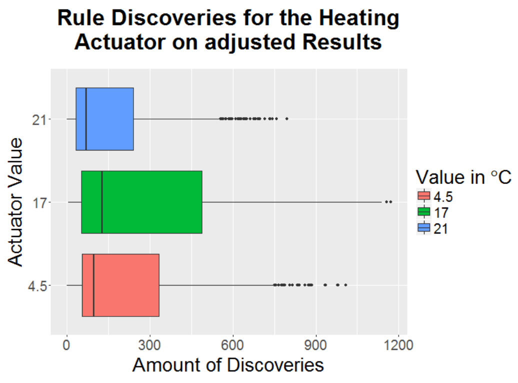
too simple, so that the system discovers all possible rules in only one run. Reconsidering Figure 23b of Section 6.2 and the amount of association rules generated by FP-Growth, this may be a reason. However, rethinking the circumstances of the evaluation - and that every configuration ran ten times each - the cause of this may be statistics itself. The fact that every configuration was executed multiple times, is - when evaluating the resulted data - the same as configuring the *trials* parameter to 10 or 30 attempts. Nevertheless a separate analysis of every run still led to similar results, leaving the first reason - although unexpected - as most probable cause. For confirmed results, this parameter needs to be evaluated using a more complex learning problem.

Considering the aforementioned small dataset problem and its origin, the answer to question four is already given. However, since it is related to this question, a reason why clustering of the target temperature setting is not directly necessary will be given too. Clustering is applied when every data point needs to be grouped into a specific category. A data point would be in this case, the target temperature value of the actuator setting. The executed value for a rule, could be determined by using the cluster center's value. An obvious advantage of clustering is that many slightly different temperature values can be represented as one group or value, which reduces the number of virtual actuators, and thus avoids small datasets.

However, Figure 29 highlights the reason why clustering is not always necessary. Although the actuator was executed in twenty different settings during the experiment, the amount of discoveries for three values - 4.5, 17 and 21 °C - strike the results. Using the original dataset, Figure 29a visualizes the amount of discovered rules from the



(A) The rule discoveries for virtual actuators using the original dataset.



(B) The same visualization as 29a using the adjusted dataset.

FIGURE 29: A comparison of virtual actuator learning w/o minimum dataset threshold.

evaluation with respect to the target actuator value. As evident, the median amount of discovered rules for rare value settings is always zero. However, when the same visualization is applied on the adjusted dataset, only three values remain as shown by Figure 29b. The reason behind this difference lies in the fact, that for not regular executed values only small datasets are generated due to the aforementioned and in Section 4.2.1 introduced segmentation concept. The remaining values are similar to the cluster centers if clustering would be applied.

It should be noticed that the proposed system is strong against unbalanced distributed values, using a minimum dataset threshold. However, this advantage strikes back if the setting of an actuator is used not frequently, because no valuable rules would be learned for it.

The last learning specific question concerns about the *-cvsplit* parameter. Although splitting the available dataset into equal sized sets for training and evaluation is a good starting point, the rules may improve or deteriorate if the distribution of the split changed. The reason behind this objective is, that with 38 sequences even the biggest dataset available for evaluation is still quite small. Since half of the data is per default used for training, this makes only 19 sequences usable for ARM and CL. Although it was already confirmed that more data lead to more discoveries and better performances, the effect of changing the distribution needs to be evaluated yet. While more data for training can lead to better results, fewer evaluation data will value the influence of every sample higher and thus variety in performances can increase.

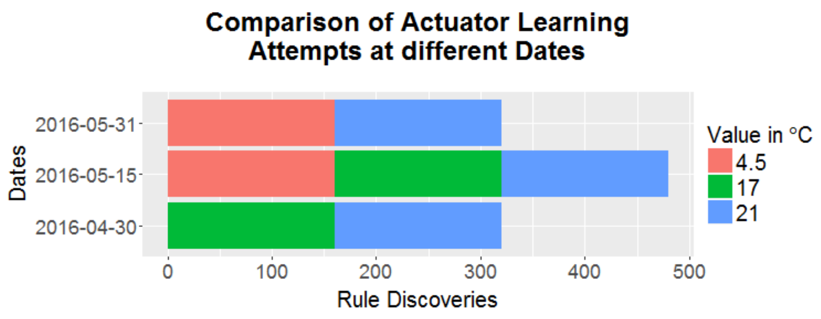


FIGURE 30: The attempts of virtual actuator learning at different dates.

Comparing results with -*cvsplit* set to 0.5 and 0.8 confirmed the expected behavior. Although the improvement is low, the median for both: discovered rules and their average performance, and the performance variety of the validated rules increased. However, using more data for training in exchange for evaluation precision, should be treated with care in this context, because the rules in a SH should be tested against as many situations as possible before they are exported into the SH system in order to provide benefit against manual configuration of behavior.

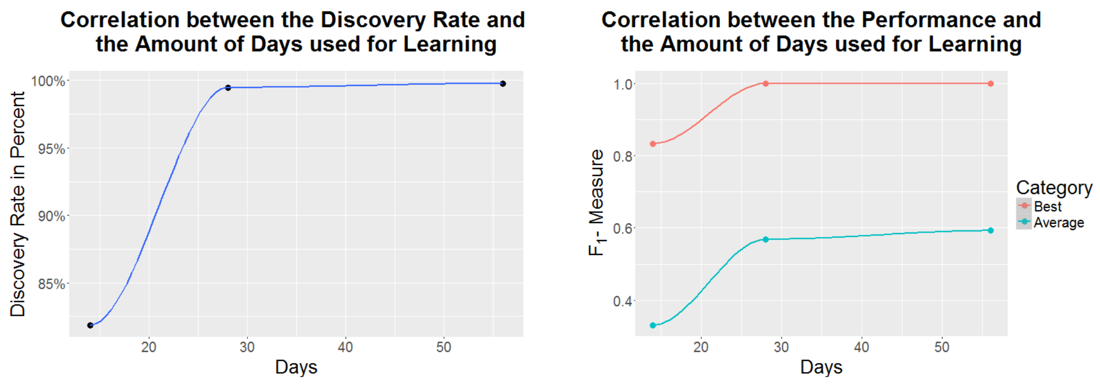
Evaluation focused on the Dataset

As mentioned before, the following evaluation is focused on the data that were collected during the experiment. Therefore the source of the behavior in this part of the evaluation, can not always be reasoned by the algorithm alone.

The 6th question concerns the challenge of learning rules in a continuous manner. Taking all previous results into account and the knowledge that a rule represents just a snapshot of the inhabitants' current behavior, it can be expected that the actuators, that the system tries to learn, will change over time according to the observed behavior.

Evaluation revealed that - as expected - the target actuators that were discovered mostly, changed over time as visualized by Figure 30. It should be mentioned, that the bar plot is based on the adjusted dataset. Nevertheless, while rules in April were focused on higher temperatures like 17 or 21°C - starting with May - also rules that set the heating controller to 4.5°C were discovered. There are two reasons for this and they are not explainable through the observed data alone.

The first reason is the change of seasons. The first month was rather cold, so the inhabitants used the heating controller several times to change the restroom temperature



(A) The correlation between discovery rate and the time frame, used for learning.

(B) The median performance with respect to the time frame, used for learning.

FIGURE 31: The correlation between the time frame (in days) used for learning and the learning behavior.

to a comfortable state. During the second month, the outdoor temperature rose to a comfortable degree so that the heating was turned off most of the time (in case of the installed device, this means set to 4.5°C). Another reason for this result were the inhabitants itself. Upon request it turned out, that one of the inhabitants was a convenient being. He wanted to maintain a comfortable room temperature even if the window was opened or he was absent. Due to this, the heating was rarely turned off, which had an effect on the dataset size and hence, was simply filtered out from the results.

Continuing with the 7th question, its main motivation for evaluation was - as mentioned before - to determine a lead time, during that a SH should observe the inhabitant before the developed system can start to discover rules. Therefore it was evaluated using the three aforementioned configurations with 14, 28 and 56 days.

As visualized by Figure 31, a correlation between days, the rule discovery rate and their respective performances for the best and average rules is striking. In some way, these results are similar to Figure 27 with the difference, that in these results the behavior of the inhabitants is included. The combination of both results reveal information about the frequency of the actuator usage. Since the system requires more than 10 samples to learn a virtual actuator, the heating temperature was rarely set more than one time per day to a specific. On the other hand this is not surprising, because the actuator was set to twenty different configurations during the experiment. This leads to the conclusion, that clustering of the actuator states would not only increase the discovery rate and performances of rules by avoiding small datasets for learning, but also decrease the lead time required by the system in order to learn well performing rules.

Rule	**Frequency**	**Percentage**
[] »> ___	16460	81.64
[Hour_EQUAL_TO_21] »> R_TA01_EQUAL_TO_17	393	1.94
[Hour_EQUAL_TO_22] »> R_TA01_EQUAL_TO_17	277	1.37
[Hour_EQUAL_TO_5] »> R_TA01_EQUAL_TO_21	258	1.27
[Hour_EQUAL_TO_6] »> R_TA01_EQUAL_TO_21	163	0.80
[Hour_EQUAL_TO_17] »> R_TA01_EQUAL_TO_21	146	0.72
[R_C01_EQUAL_TO_OPEN] »> R_TA01_EQUAL_TO_4.5	146	0.72
[Hour_EQUAL_TO_9] »> R_TA01_EQUAL_TO_17	89	0.44
[WeekDay_EQUAL_TO_4] »> R_TA01_EQUAL_TO_4.5	77	0.38
[R_M01_EQUAL_TO_ON] »> R_TA01_EQUAL_TO_4.5	63	0.31

Table 9: A structured overview of the top ten most discovered rules during evaluation.

Finally, the last question will be evaluated manually by considering the most frequent discovered rules. During evaluation 395 different rules were exported of the system, from that the most frequent rule covers 81.64% of all rules, as presented in Table 9. The empty rule is generated if the system could not discover any rules. Due to the aforementioned dataset size problem and since no minimum thresholds for the performance of the dataset size were given, it could be expected that this rule was discovered most frequent and dominated the statistic. However the results also confirm, one of the major disadvantages that the proposed system currently has, is the complete renunciation of clustering. Considering the second and third rule, the system exported two rules, that set the target temperature of the heating to 17°C at 9 and 10 o'clock in the evening. However, these are hours neighboring numbers and could have suited into a single - more general - rule. The same can be applied to the 4th and 5th rule. As a result the system generates many high specialized rules, that may be usable in a SH but could be expressed more general, to reduce the overload in managing the final set of rules for the rule-based system of the SH.

Another problem of the system is, that the learned rules - that were derived from the data - are not semantically validated. The last rule for example turns the heating off each time an inhabitant enters the toilet, regardless of the current temperature in it. Although performing well on the observed data, such rules are completely useless for real usage in a SH. However, the system is also capable of discovering useful rules, as shown by the 7th rule that turns the heating off when the window is opened.

Summarizing the evaluation section, the system has many advantages useful for real applications like: the robust segmentation concept, the adaptive rule discovery which is based only on data, and the independence from clustering of actuator states, when a minimum dataset threshold is applied. At the same time, the system has still many flaws like: the generation of many high specialized rules instead of fewer more general ones, the often not semantically useful rules that are however confirmed by the observed data, and the increase in computational resources on larger SH setups.

7 Conclusion and Future Work

7.1 Revisiting the Problem

During the last decades, the proliferation of AmI in the domestic area has risen steadily. Driven by this development, more and more manufacturer are flooding the marketplace with various solutions for the so called, *Smart Home*. However, current solutions share the characteristic, that no adaptive intelligence manages the domestic environment, which contradicts with the definition of AmI.

Instead this task is postponed to the inhabitants, leaving them the choice to simply manage light, heating or locks via remote access or by using manually written expert rules, which are executed by an integrated rule-based system. However, the skill to write and maintain these rules needs to be acquired and refined continuously, since every new installed sensor, new target scenario or new inhabitant behavior needs to be integrated into the rules. In addition, the difficulty increases if the fact is considered, that humans are very unsteady beings, and depending on their current condition, may prefer other configurations for the domestic environment, like a different temperature or light intensity.

Therefore the aim of this thesis was to propose a system, that not only learns rules to manage the domestic environment by observing the inhabitants behavior, but that also adapt the learned knowledge over time to new situations, and that is able to integrate biomedical data from the inhabitant into its knowledge base.

7.2 Summary and Results of the Thesis

The proposed system combines two different paradigms of ML: ARM and CL. By segmenting the captured event stream after executed actuators with their respective states into datasets of sequences, ARM is applied in order to discover associations between sensors. For every relation, the respective environmental state of the SH is learned, by applying CL on examples, generated from the extracted sequences. The resulted decision rules are validated and finally exported into a SH compatible format - in case of this project, openHAB. To adapt the rule base over time, the key properties of the exported rules are stored in a table at the file system. By executing the process periodically, the system will adapt the rule base to new sensors and even exchange rules, if better are discovered.

Due to the abstract nature of the concept, several algorithms for ARM and CL were implemented. In order to determine a reliable configuration of the learning system, a screening was performed at which the algorithms for ARM and for CL were executed under different configurations. As a result, the combination of the algorithm FP-Growths with the C4.5 descendant PART were used for evaluation of the learning system. Even though the integration of a biomedical sensor into the SH platform could not be realized due to lack of time, the conceptual usage of such sensors to sensitize the system, was elaborated.

Evaluation of the learning system revealed several potential improvements like: clustering of the classifier's input and output feature space, constraining the discovered association and decision rules even more to reduce the computational load, and increase the rule discovery rate on smaller datasets. At the same time, the evaluation confirmed positive results for several choices that were made like: the segmentation concept that allows the system to learn rules for any actuator in any SH setup, the composition of ARM and CL that provides a sensor setup independent approach for behavior rule learning, and the modular implementation allowing to evaluate various combinations of ARM and CL algorithms.

Last but not least the most important result from this thesis is the following: The generated rules can perform very well on the data but may still fail in real usage, because the data - regardless of how much of the home is observed - is just a very limited view on an infinite feature space, at which human behavior exists. Due to this, rules that were generated only from data captured by sensors in a SH, should never be used unchecked in this context.

7.3 Future Work and Final Thoughts

As mentioned before, the proposed system has still much room for improvement. Beside the aforementioned functionalities, the influence of biomedical sensors to the learning behavior of the proposed system, needs to be evaluated yet. In addition, due to the old building and difficulties in the SH installation, only one actuator could be installed.

However, a real SH installation contains many actuators and therefore the possibility to generate loops in rules (if two actuators activate each other), its influence in a working SH and possible approaches for avoidance, needs to be elaborated in future work.

The vision of AmI is very blurry, which is also reflected by the state of the art and the general research of it. Instead of following this blurry definition and try to solve a Herculean task, more focus should be given to current problems like the missing adaptivity or restrictive supportive functionalities in SHs. Maybe then, when more and more of the current problems are solved, this vision of AmI will sharpen too.

A Appendix

Support	Confidence	Approach	F_1* Best	F_1* Average	F_1* Worst	|Rules|	%-Used
0.25	0.25	Apriori	0.96	0.15	0	14	4
0.50	0.25	Apriori	0.94	0.18	0	11	9
0.75	0.25	Apriori	0.00	0.00	0.00	1	22
0.25	0.50	Apriori	0.97	0.18	0	14	5
0.50	0.50	Apriori	0.80	0.13	0	5	9
0.75	0.50	Apriori	0.66	0.07	0	2	19
0.25	0.75	Apriori	1.00	0.20	0	13	8
0.50	0.75	Apriori	0.91	0.11	0	7	17
0.75	0.75	Apriori	0.66	0.13	0	1	24
0.25	**0.25**	**FP-Growth**	**1.00**	**0.26**	**0**	**19**	**1** **
0.50	0.25	FP-Growth	1.00	0.19	0	20	1
0.75	0.25	FP-Growth	1.00	0.17	0	19	2
0.25	0.50	FP-Growth	1.00	0.22	0	28	1
0.50	0.50	FP-Growth	0.94	0.20	0	19	1
0.75	0.50	FP-Growth	1.00	0.17	0	17	2
0.25	0.75	FP-Growth	1.00	0.23	0	23	2
0.50	0.75	FP-Growth	1.00	0.19	0	34	2
0.75	0.75	FP-Growth	1.00	0.22	0	21	2

TABLE 10: A comparison table of Apriori and FP-Growth containing the results of all evaluated support and confidence combinations.

* F_1 refers to the median F_1-measure of twenty runs.
** The bold row is the chosen ARM configuration for the evaluation of the learning system.

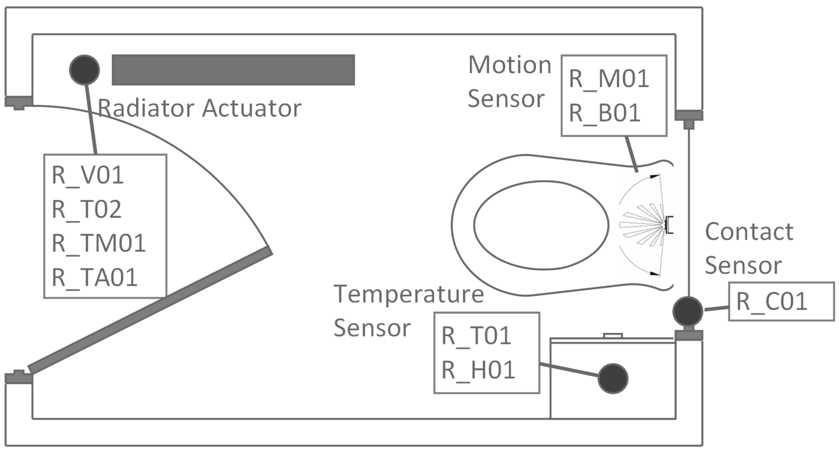


FIGURE 32: The outline of the restroom setup.

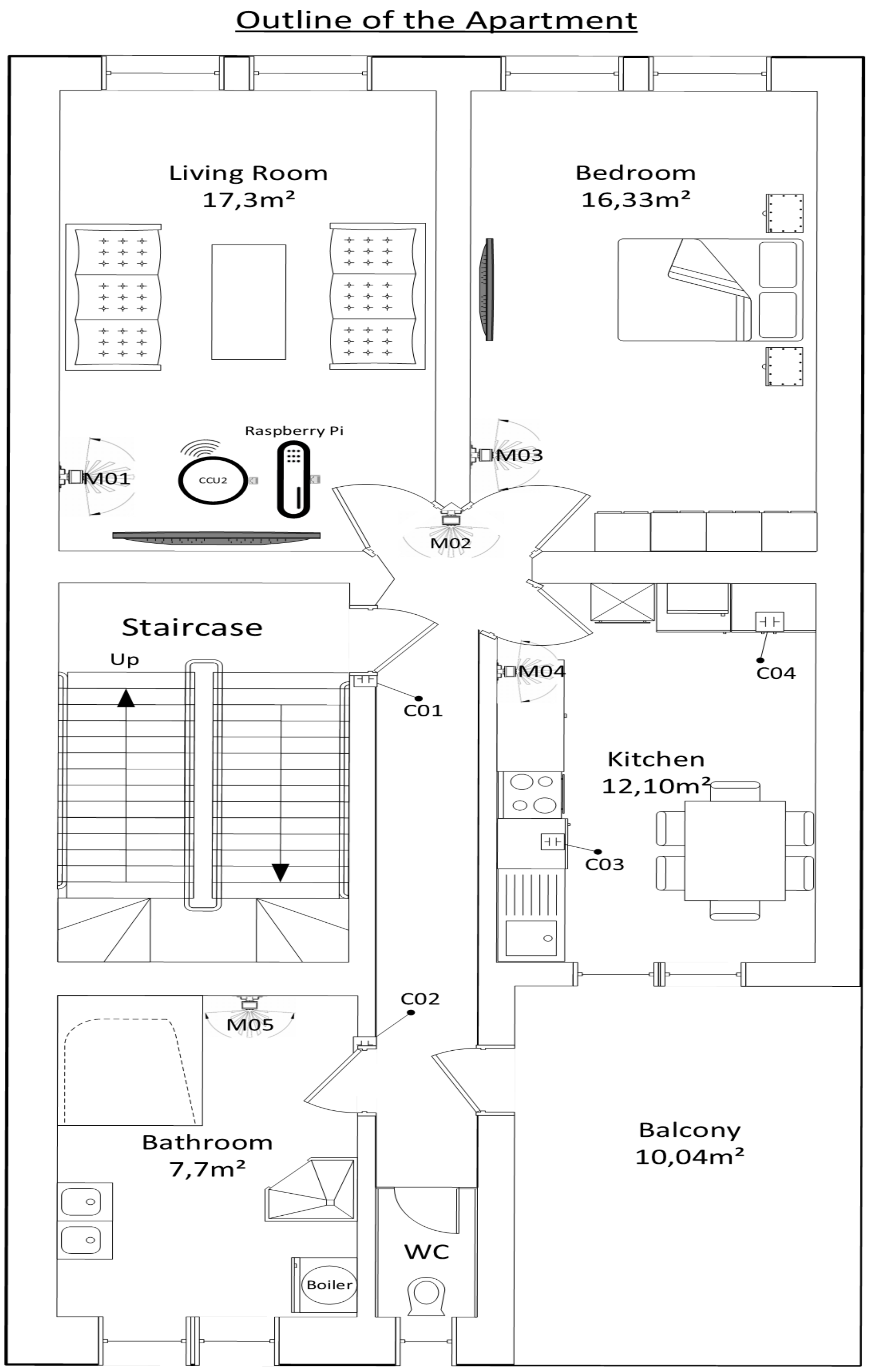


Figure 33: The outline of the Smart Home apartment.

Parameter	Definition	Parameter	Definition
rulefolder	Filepath where the target rules are exported to.	**actors**	Parameter that defines the number of different actuator labels.
ruletable	Filepath of the rule-table, containing basic informations about every exported rule.	**actors.X**	With $X \geq 0$ the labels of the various different actuators can be defined like elements in a list.
rulelog	Filepath of the log-file location. Required for the adaptation GUI.	**cvsplit**	A parameter that defines the distribution of training and evaluation data from all segmented sequences.
format	File extension of the target rule-based system in the SH.	**minsetnum**	A threshold that states the minimum size of segmented sequences required in order to learn rules.
logname	openHAB specific parameter. Defines the logger of openHAB's logging facility.	**performance**	A classpath to the implementation of the performance measure. Custom measures need to implement the PerformanceInterface.
interpreter	The classpath of the target rule interpreter. Will be used to transform the internal decision rules into external SH compatible rules.	**performance.min**	A threshold on the performance results to filter out bad performing rules.
db	The classpath of the database driver. Other Java compatible databases can be loaded with this parameter.	**arm**	The classpath of the ARM implementation. Custom approaches need to implement the ARMInterface.
db.url	The address of the database server.	**arm.support**	The support parameter of the ARM approach.
db.username	The username for database access.	**arm.confidence**	The confidence parameter of the ARM approach.
db.password	The password for database access.	**cla**	The classpath of the CL implementation. Custom approaches need to implement the ClassifierInterface.
db.name	The name of the database.	**cla.param**	The parameters given to the classifier.
engine	The classpath of the rule engine implementation. Custom engines need to implement the RuleEngineInterface.		

Table 11: A structured overview of the parameters required by the learning system.

Primary Literature

[1] G. D. Abowd, A. F. Bobick, I. A. Essa, E. D. Mynatt, and W. A. Rogers. The aware home: a living laboratory for technologies for successful aging. In *Proceedings of the AAAI-02 Workshop "Automation as Caregiver*, pages 1–7, 2002.

[2] E. Alpaydin. *Introduction to Machine Learning.* The MIT Press, 2014.

[3] A. Augello, A. Santangelo, S. Sorce, G. Pilato, A. Gentile, A. Genco, and S. Gaglio. Maga: a mobile archaeological guide at agrigento, 2006.

[4] C. Beierle and G. Kern-Isberner. *Methoden wissensbasierter Systeme - Grundlagen, Algorithmen, Anwendungen.* Springer-Verlag, Berlin Heidelberg New York, 5. aufl. Edition, 2014. ISBN: 978-3-834-82300-7.

[5] R. R. Bouckaert, E. Frank, M. A. Hall, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. Weka—experiences with a java open-source project. *J. Mach. Learn. Res.*, 11:2533–2541, Dec. 2010. ISSN: 1532-4435. URL: `http://dl.acm.org/citation.cfm?id=1756006.1953016`.

[6] K. Brooks. The context quintet: narrative elements applied to context awareness. In *The 10th International Conference on Human-Computer Interaction*, June 2003.

[7] J. Cendrowska. Prism: an algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349–370, 1987.

[8] S. L. Chiu. Extracting fuzzy rules from data for function approximation and pattern classification. *Fuzzy Information Engineering: A Guided Tour of Applications. John Wiley & Sons*, 1997.

[9] P. Clark and T. Niblett. The cn2 induction algorithm. *Mach. Learn.*, 3(4):261–283, Mar. 1989. ISSN: 0885-6125. DOI: `10.1023/A:1022641700528`. URL: `http://dx.doi.org/10.1023/A:1022641700528`.

[10] P. Compton and R. Jansen. Knowledge in context: a strategy for expert system maintenance. In *Proceedings of the Second Australian Joint Conference on Artificial Intelligence*, AI '88, pages 292–306, Adelaide, Australia. Springer-Verlag New York, Inc., 1990. ISBN: 0-387-52062-7. URL: `http://dl.acm.org/citation.cfm?id=89411.89756`.

[11] T. D. Corchado J.M. B. J. Alz-mas: alzheimer's special care multi-agent system. In part Proceedings of the workshop on Agents Applied in Health Care (ECAI 2006), pages 382–396. A. Moreno, U. Cortes, R. Annicchiarico and J. Nealon (Eds)., 2006.

[12] J. M. Corchado, J. Bajo, Y. d. Paz, and D. I. Tapia. Intelligent environment for monitoring alzheimer patients, agent technology for health care. *Decis. Support Syst.*, 44(2):382–396, Jan. 2008. ISSN: 0167-9236. DOI: 10.1016/j.dss.2007.04.008. URL: http://dx.doi.org/10.1016/j.dss.2007.04.008.

[13] S. Costantini, P. Inverardi, L. Mostarda, A. Tocchio, and P. Tsintza. User profile agents for cultural heritage fruition. *Proceedings of Artificial Societies for Ambient Intelligence (ASAMI07)*, 2007.

[14] K. Desai, D. Hoffman, and D. Le Sage. *Red Hat JBoss BRMS 6.0 Development Guide*. Red Hat Inc., 2014.

[15] M. J. Egenhofer and R. D. Franzosa. Point-set topological spatial relations. *International Journal of Geographical Information Systems*, 5(2):161–174, 1991. DOI: 10.1080/02693799108927841. eprint: http://dx.doi.org/10.1080/02693799108927841. URL: http://dx.doi.org/10.1080/02693799108927841.

[16] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C.-W. Wu, and V. S. Tseng. Spmf: a java open-source pattern mining library. *J. Mach. Learn. Res.*, 15(1):3389–3393, Jan. 2014. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2627435.2750353.

[17] E. Frank and I. H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 144–151, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc., 1998. ISBN: 1-55860-556-8. URL: http://dl.acm.org/citation.cfm?id=645527.657305.

[18] K. Gopalratnam and D. J. Cook. Active lezi: an incremental parsing algorithm for sequential prediction. *International Journal on Artificial Intelligence Tools*, 13(04):917–929, 2004.

[19] V. M. Guralnik. *Mining Interesting Patterns from Sequential Data*. PhD thesis, University of Minnesota, Minneapolis, Minnesota, 2001.

[20] K. Z. Haigh, L. M. Kiff, J. Myers, V. Guralnik, C. W. Geib, J. Phelps, and T. Wagner. The independent lifestyle assistant^{TM}(i.l.s.a.): ai lessons learned. In *Proceedings of the 16th Conference on Innovative Applications of Artifical Intelligence*, IAAI'04, pages 852–857, San Jose, California. AAAI Press, 2004. ISBN: 0-262-51183-5. URL: http://dl.acm.org/citation.cfm?id=1597321.1597335.

[21] K. Z. Haigh, L. M. Kiff, J. Myers, and K. Krichbaum. *The independent lifestyle assistant^{TM} (i.l.s.a.): deployment lessons learned.* In *AAAI Workshop - Technical Report.* Volume WS-04-05. 2004, pages 11–16.

[22] J. Han, M. Kamber, and J. Pei. 3 - data preprocessing. In J. H. Kamber and J. Pei, editors, *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 83–124. Morgan Kaufmann, Boston, third edition edition, 2012. ISBN: 978-0-12-381479-1. DOI: http://dx.doi.org/10.1016/B978-0-12-381479-1.00003-4. URL: http://www.sciencedirect.com/science/article/pii/B9780123814791000034.

[23] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 1–12, Dallas, Texas, USA. ACM, 2000. ISBN: 1-58113-217-4. DOI: 10.1145/342009.335372. URL: http://doi.acm.org/10.1145/342009.335372.

[24] E. R. Hruschka and N. F. Ebecken. Extracting rules from multilayer perceptrons in classification problems: a clustering-based approach". *Neurocomputing* ", 70"(1–3"):384–397", 2006". ISSN: 0925-2312". DOI: http://dx.doi.org/10.1016/j.neucom.2005.12.127". URL: http://www.sciencedirect.com/science/article/pii/S0925231206000403%22. Neural NetworksSelected Papers from the 7th Brazilian Symposium on Neural Networks (SBRN '04)7th Brazilian Symposium on Neural Networks ".

[25] B. Kang, P. Compton, and P. Preston. Multiple classification ripple down rules: evaluation and possibilities. In *Proceedings 9th Banff knowledge acquisition for knowledge based systems workshop*, volume 1, pages 17–1, 1995.

[26] R. M. Karp. *Complexity of computer computations: proceedings of a symposium on the complexity of computer computations.* In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors. Springer US, Boston, MA, 1972. Part Reducibility among Combinatorial Problems, pages 85–103. ISBN: 978-1-4684-2001-2. DOI: 10.1007/978-1-4684-2001-2_9. URL: http://dx.doi.org/10.1007/978-1-4684-2001-2_9.

[27] G. Marreiros, R. Santos, C. Ramos, J. Neves, P. Novais, J. Machado, and J. Bulas-Cruz. Ambient intelligence in emotion based ubiquitous decision making, 2007.

[28] R. Michalski. On the Quasi-Minimal Solution of the General Covering Problem. In *Vth International Symposium on Information Processing (FCIP)*, volume A3 of number 2, pages 125–128, Yugoslavia, Bled, Oct. 1969.

[29] T. M. Mitchell. *Machine Learning.* McGraw-Hill, Inc., New York, NY, USA, 1st edition, 1997.

[30] E. Olsson. *Heart Rate Variability in Stress-related Fatigue, Adolescent Anxiety and Depression and its Connection to Lifestyle.* PhD thesis, Uppsala University, Department of Psychology, 2010, page 66.

[31] I. OpenRules. *OPENRULES©- Open Source Business Decision Management System - Release 6.3.4 - User Manual.* OpenRules, Inc., 2015.

[32] W. H. Organization. *ICF : International classification of functioning, disability and health / World Health Organization.* English. World Health Organization Geneva, 2001, iii, 299 p. : ISBN: 9241545429.

[33] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, Mar. 1986. ISSN: 0885-6125. DOI: `10.1023/A:1022643204877`. URL: `http://dx.doi.org/10.1023/A:1022643204877`.

[34] R. Setiono. Extracting rules from neural networks by pruning and hidden-unit splitting. *Neural Computation*, 9(1):205–225, 1997.

[35] M. Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, July 1999. ISSN: 1559-1662. DOI: `10.1145/329124.329126`. URL: `http://doi.acm.org/10.1145/329124.329126`.

Secondary Literature

[36] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, 22(2):207–216, 1993.

[37] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc., 1994. ISBN: 1-55860-153-8. URL: `http://dl.acm.org/citation.cfm?id=645920.672836`.

[38] M. Alam, M. Reaz, and M. Mohd Ali. Speed: an inhabitant activity prediction algorithm for smart homes. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 42(4):985–990, July 2012. ISSN: 1083-4427. DOI: `10.1109/TSMCA.2011.2173568`.

[39] O. Alliance. OSGi Service Platform - OSGi Compendium. Release 4, Jan. 2012.

[40] J. C. Augusto. Ambient intelligence: the confluence of ubiquitous/pervasive computing and artificial intelligence. English. In A. Schuster, editor, *Intelligent Computing Everywhere*, pages 213–234. Springer London, 2007. ISBN: 978-1-84628-942-2. DOI: `10.1007/978-1-84628-943-9_11`. URL: `http://dx.doi.org/10.1007/978-1-84628-943-9_11`.

[41] S. E. Bibri. *The Human Face of Ambient Intelligence - Cognitive, Emotional, Affective, Behavioral and Conversational Aspects*. Springer, Berlin, Heidelberg, 2015. ISBN: 978-9-462-39130-7.

[42] K. Bouchard, A. Bouzouane, and B. Bouchard. Discovery of topological relations for spatial activity recognition. In *Computational Intelligence and Data Mining (CIDM), 2013 IEEE Symposium on*, pages 73–80, Apr. 2013. DOI: `10.1109/CIDM.2013.6597220`.

[43] S. T. M. Bourobou and Y. Yoo. User activity recognition in smart homes using pattern clustering applied to temporal ann algorithm. *Sensors*, 15(5):11953, 2015. ISSN: 1424-8220. DOI: `10.3390/s150511953`. URL: `http://www.mdpi.com/1424-8220/15/5/11953`.

[44] L. Chen, G. Okeyo, H. Wang, R. Sterritt, and C. Nugent. A systematic approach to adaptive activity modeling and discovery in smart homes. In *Biomedical Engineering and Informatics (BMEI), 2011 4th International Conference on*, volume 4, pages 2192–2196, Oct. 2011.

[45] D. J. Cook, J. C. Augusto, and V. R. Jakkula. Review: ambient intelligence: technologies, applications, and opportunities. *Pervasive Mob. Comput.*, 5(4):277–298, Aug. 2009. ISSN: 1574-1192. DOI: 10.1016/j.pmcj.2009.04.001. URL: http://dx.doi.org/10.1016/j.pmcj.2009.04.001.

[46] D. J. Cook, N. C. Krishnan, and P. Rashidi. Activity discovery and activity recognition: a new partnership. *Cybernetics, IEEE Transactions on*, 43(3):820–828, 2013.

[47] D. Cook, M. Youngblood, I. Heierman E., K. Gopalratnam, S. Rao, A. Litvin, and F. Khawaja. Mavhome: an agent-based smart home. In *Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference on*, pages 521–524, Mar. 2003. DOI: 10.1109/PERCOM.2003.1192783.

[48] O. Corp. Why are java applications blocked by your security settings? URL: https://www.java.com/en/download/help/java_blocked.xml (visited on 06/20/2016).

[49] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.-C. Burgelman. Scenarios for ambient intelligence in 2010. In IST Programme Advisory Group (ISTAG), 2001.

[50] M. Friedewald, O. D. Costa, Y. Punie, P. Alahuhta, and S. Heinonen. Perspectives of ambient intelligence in the home environment. *Telemat. Inf.*, 22(3):221–238, Aug. 2005. ISSN: 0736-5853. DOI: 10.1016/j.tele.2004.11.00l. URL: http://dx.doi.org/10.1016/j.tele.2004.11.00l.

[51] K. Garg and D. Kumar. Comparing the performance of frequent pattern mining algorithms. *International Journal of Computer Applications*, 69(25), 2013.

[52] V. Guralnik and K. Z. Haigh. Learning models of human behaviour with sequential patterns. In *Proceedings of the AAAI-02 workshop "Automation as Caregiver"*, pages 24–30, 2002. AAAI Technical Report WS-02-02.

[53] M. B. Hassine. Easy rules - overview. URL: http://www.easyrules.org/about/overview.html (visited on 05/04/2016).

[54] J. Hipp, U. Güntzer, and G. Nakhaeizadeh. Algorithms for Association Rule Mining &Mdash; a General Survey and Comparison. *SIGKDD Explor. Newsl.*, 2(1):58–64, June 2000. ISSN: 1931-0145. DOI: `10.1145/360402.360421`. URL: `http://doi.acm.org/10.1145/360402.360421`.

[55] Home of FHEM. URL: `http://fhem.de/fhem.html` (visited on 05/06/2016).

[56] ISTAG. Ambient Intelligence: from vision to reality. Technical report, Information Society Technologies Advisory Group (ISTAG), Sept. 2003.

[57] P. D. K. Jonas and M. Rademacher. MCLab. URL: `http://mc-lab.inf.h-brs.de/` (visited on 06/05/2016).

[58] D. Karakos, S. Khudanpur, J. Eisner, and C. E. Priebe. Unsupervised classification via decision trees: an information-theoretic perspective. In *Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.* Volume 5, v/1081–v/1084 Vol. 5, Mar. 2005. DOI: `10.1109/ICASSP.2005.1416495`.

[59] K. Kreuzer. Reinventing the Wheel. Nov. 2009. URL: `http://kaikreuzer.blogspot.com/2009/11/reinventing-wheel.html` (visited on 04/04/2016).

[60] K. Larson and R. Topping. PlaceLab: A House_n + TIAX Initiative, Nov. 2003.

[61] P. Manheller. Automatisierte erkennung von ungewöhnlichem verhalten im privathaushalt und reaktion in notsituationen. May 2016.

[62] P. Manheller. Herstellerübergreifende Heim-/Geäudeautomatisierung beim Einsatz von openHAB. Praxisprojektbericht im Studiengang Master Informatik, June 2015.

[63] openHAB. URL: `http://www.openhab.org/` (visited on 05/06/2016).

[64] openHAB UG (haftungsbeschränkt). Bindings - openhab/openhab wiki. GitHub. Aug. 2015. URL: `https://github.com/openhab/openhab/wiki/Bindings` (visited on 04/06/2016).

[65] openHAB UG (haftungsbeschränkt). Feature overview - openhab/openhab wiki. GitHub. Jan. 2015. URL: `https://github.com/openhab/openhab/wiki/Feature-Overview` (visited on 04/06/2016).

[66] openHAB UG (haftungsbeschränkt). openHAB - Features - Architecture. Jan. 2016. URL: `http://www.openhab.org/features/architecture.html` (visited on 04/04/2016).

[67] openHAB UG (haftungsbeschränkt). Persistence - openhab/openhab Wiki. Mar. 2016. URL: https://github.com/openhab/openhab/wiki/Persistence (visited on 04/05/2016).

[68] openHAB UG (haftungsbeschränkt). Transformations - openhab/openhab Wiki. Feb. 2016. URL: https://github.com/openhab/openhab/wiki/Transformations (visited on 04/05/2016).

[69] OSGi Alliance. Specifications – OSGi^{TM} Alliance. URL: https://www.osgi.org/developer/specifications/ (visited on 04/04/2016).

[70] C. Ramos. Argumentative agents with emotional behaviour modelling for participants' support in group decision-making meetings. 2008. URL: http://www.gecad.isep.ipp.pt/GECAD/Pages/Projects/ProjectDetails.aspx?id=32 (visited on 12/10/2015).

[71] A. p. Reynolds and B. Iglesia. *Evolutionary multi-criterion optimization: 4th international conference, emo 2007, matsushima, japan, march 5-8, 2007. proceedings.* In S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, and T. Murata, editors. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. Part Rule Induction for Classification Using Multi-objective Genetic Programming, pages 516–530. ISBN: 978-3-540-70928-2. DOI: 10.1007/978-3-540-70928-2_40. URL: http://dx.doi.org/10.1007/978-3-540-70928-2_40.

[72] Rudolf könig im interview – der erfinder von FHEM zum thema smart home. URL: http://www.meintechblog.de/2015/07/rudolf-koenig-im-interview-der-erfinder-von-fhem-zum-thema-smart-home/ (visited on 05/06/2016).

[73] D. Sampaio, L. Reis, and R. Rodrigues. A survey on ambient intelligence projects. In *Information Systems and Technologies (CISTI), 2012 7th Iberian Conference on*, pages 1–6, June 2012.

[74] P. Schoutsen. Architecture. URL: https://home-assistant.io/developers/architecture/ (visited on 05/06/2016).

[75] P. Schoutsen. Home assistant. URL: https://home-assistant.io/ (visited on 05/06/2016).

[76] Systemübersicht – FHEMWiki. URL: http://www.fhemwiki.de/wiki/System%C3%BCbersicht (visited on 05/06/2016).

[77] Tedburke. English: an idealised lead i electrocardiogram (ECG) section of a healthy person, with labeled features p, q, r, s, t and r-r interval. Oct. 1, 2007. URL: https://commons.wikimedia.org/wiki/File:Ecg.png (visited on 04/29/2016).

[78] Z. Vale, A. Machado, and C. Ramos. Sparse: an intelligent alarm processor and operator assistant. *IEEE Expert*, 12(3):86–93, May 1997. ISSN: 0885-9000. DOI: 10.1109/64.590086.

[79] S.-I. Yang and S.-B. Cho. Recognizing human activities from accelerometer and physiological sensors. In *Multisensor Fusion and Integration for Intelligent Systems, 2008. MFI 2008. IEEE International Conference on*, pages 100–105, Aug. 2008. DOI: 10.1109/MFI.2008.4648116.

[80] M. F. B. Zaiyadi. generation5 - Expert System for Car Maintenance and Troubleshooting. June 26, 2005. URL: http://www.generation5.org/content/2005/CarMaintenance.asp (visited on 04/02/2016).

[81] S. Zhang, S. McClean, B. Scotney, P. Chaurasia, and C. Nugent. Using duration to learn activities of daily living in a smart home environment. In *Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2010 4th International Conference on-NO PERMISSIONS*, pages 1–8, Mar. 2010. DOI: 10.4108/ICST.PERVASIVEHEALTH2010.8804.