ROYAL INSTITUTE
OF TECHNOLOGY

Smart assistants for smart homes

KATHARINA RASCH

Doctoral Thesis in Electronic and Computer Systems
Stockholm, Sweden 2013

TRITA-ICT/ECS AVH 13:16
1653-6363
KTH/ICT/ECS/AVH-13/16-SE
978-91-7501-837-9

KTH School of Information and
Communication Technology
SE 164-40 Kista
SWEDEN

Akademisk avhandling som med tillstånd av Kungliga Tekniska Högskolan framlägges till offentlig granskning för avläggande av teknologie doktorsexamen i elektronik och datorsystem den 11 Oktober 2013 klockan 13 i Sal E, Forum Isafjordsgatan 39, Kista, Kungliga Tekniska Högskolan.

Tryck: Universitetsservice US AB

Abstract

The smarter homes of tomorrow promise to increase comfort, aid elderly and disabled people, and help inhabitants save energy. Unfortunately, smart homes today are far from this vision – people who already live in such a home struggle with complicated user interfaces, inflexible home configurations, and difficult installation procedures. Under these circumstances, smart homes are not ready for mass adoption.

This dissertation addresses these issues by proposing two smart assistants for smart homes. The first assistant is a recommender system that suggests useful services (i.e actions that the home can perform for the user). The recommended services are fitted to the user's current situation, habits, and preferences. With these recommendations it is possible to build much simpler user interfaces that highlight the most interesting choices currently available. Configuration becomes much more flexible: since the recommender system automatically learns user habits, user routines no longer have to be manually described. Evaluations with two smart home datasets show that the correct service is included in the top five recommendations in 90% of all cases.

The second assistant addresses the difficult installation procedures. The unique feature of this assistant is that it removes the need for manually describing device functionalities (such descriptions are needed by the recommender system). Instead, users can simply plug in a new device, begin using it, and let the installation assistant identify what the device is doing. The installation assistant has minimal requirements for manufacturers of smart home devices and was successfully integrated with an existing smart home. Evaluations performed with this smart home show that the assistant can quickly and reliably learn device functionalities.

Acknowledgements

First and foremost I am grateful to my supervisor Professor Rassul Ayani for all the support and guidance that he has given me over the past five years. Without him, this dissertation would not have been possible. I am thankful to my supervisor Professor Christian Schulte for all the helpful comments and ideas I got from him for my work. I also want to thank Johan Schubert for providing valuable feedback in the final year of my studies.

Many thanks go to Professor Alois Ferscha for taking the time to be my opponent and for reading and commenting on this dissertation. A big thank you also goes to Marianne Hellmin and Sandra Gustavsson Nylén for helping me with all the bureaucratic hurdles. I am thankful to KTH and Sweden for giving me this opportunity.

This research was partially funded by the "Smart homes for all" project (SM4All, EU FP7 STREP Grant No. 224332) and I would like to thank all my colleagues from this project. In particular, I am grateful to Fei Li and Sanjin Sehic from Vienna University of Technology for our collaboration, which resulted in a number of joint publications that form the basis for this dissertation.

I am grateful to the all-knowing Thesaurus for never failing to show me alternative paths.

I want to thank Eric-Oluf for proofreading and for all those lunches in Gallerian. Thank you to my fellow PhD students Imran, Shahab, Nima, Kathrin, Voravit, Irfan, Natalja, Salman, Byron, and many others for our semi-regular fikas and for making my workplace a pleasant one.

Ett stort tack till Petter, Marianela, Edward, Annika och alla mina andra vänner i Sverige för mycket nöje och distraktion från jobbet.

Vielen Dank an Sandra, Tobias, Tanja, Sebastian, Luise und die Familie für die vielen Besuche und für (manchmal dringend nötige) Ablenkung und Freude.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The idea of creating a smarter home – a home that unobstrusively assists people in their everyday lives – is rather old. The term *smart home* itself became popular already in the 1980s [43]. And the dream of automating domestic tasks has been the driving force behind numerous technological developments for more than a century [43]. Indeed, many of our homes today would have been considered smart in the 1960s [22]:

> *"We do have remote controls for our TVs, we do have smoke detectors and passive infra-red burglar alarms, we do have timers on our central heating."* [43, page ix]

However, visions for the home of the future are far more exciting than today's status quo. Inspired by Mark Weiser's call for ubiquitous computing [109], researchers have been working for the past two decades to create environments that are "saturated with computing and communication capability, yet so gracefully integrated with users that it becomes a technology that disappears" [93, p. 1].

The smarter home learns how people live their lives and offers assistance whenever needed. This might be simply to turn on comfortable lighting when the inhabitant is reading a magazine, or it might be to make sure that an Alzheimer patient always turns off the stove when cooking and to notify caretakers when necessary. Since the home is *context-aware*, it always knows what is happening and what functionality might be needed.

1.1 Visions for the home of the future

Increase comfort

For most people, an increased level of comfort might be the most attractive feature of having a smart home. Comfort means being able to remote control living room lights when sitting on the couch watching TV, to let the home make certain that windows are closed when it rains, and to check from afar that the front door is locked. Several studies of potential smart home customers found that comfort tasks and home control are indeed the most interesting

applications [107, 34, 8]; e.g. remote control of heating, lights and windows, and help with cleaning tasks are commonly mentioned. Several inhabitants of smart homes remark that increased comfort is indeed one of their favorite aspects of living in such a home [13, 74].

Of course, the ability to remotely control devices does not automatically mean that the home can be called smart. The key necessity for increased comfort is usability: "things must be simpler to do than in a normal house" [85, p. 232]. Consequently, researchers stress that the smart home must be context-aware, i.e. it must be attentive to the current situation and the current needs of the inhabitant. The context-aware home is always ready to assist the inhabitant without needing complicated instructions.

Assist elderly people and people with disabilities

The demographics of many countries, in particular in Europe, the US and Japan, are changing dramatically. It is projected that 82 million people (20%) in the US will be over the age of 60 by the year 2025 [79]. In France, it is projected that the ratio of persons aged 16-65 compared to those over 65 will decrease from 3:1 (in the 1990s) to 2:1 (by 2040) [16]. With increasing age, people often experience physical and sensory impairments [43]. In 2000, 10% of elders in the US suffered from Alzheimer's disease [79].

It has been shown that a pleasant home environment is a significant factor for quality of life for elderly people [40, 64]. Many elderly choose to continue living independently and many are proud to have been living in the same home for a long time [40]. Researchers are calling for smart homes that enable *aging in place*, so that more people can avoid having to move to a care facility [79].

A number of impressive applications have been proposed for assisting elderly and disabled people. For example, the smart home can check if the inhabitant has performed all necessary activities of daily living (ADLs, e.g. eating, drinking, washing themselves) [79]. It detects changes in the behavior patterns of the inhabitant that might indicate declines in health [101]. It keeps remotely living caretakers up-to-date about the health status and the activity level of the inhabitant [79]. And, of course, the smart home can unobstrusively help users with any tasks they might be having trouble with, e.g. opening heavy drawers.

Save energy

There are also encouraging attempts to build smart homes that help people save energy. To achieve this, smart meters can be coupled with user interfaces to give people continuous feedback about their resource consumption. For example, the energy consumption of each home device can be monitored in mobile phone applications [84], and the overall consumption is shared with social contacts to entice competition [38]. In [57], LEDs are built into a shower head to give people a direct feedback of their water usage. Up to 5-15% reduction in energy consumption have been reported [25], but there are criticisms regarding the general lack of long-term studies in the field [14].

Smart thermostats take a very different approach; the heating is automatically regulated, based on the current occupancy status and predictions of when the inhabitants will return

home [65, 97]. Energy savings of up to 28% are reported, at a cost of approximately $US25 in sensors [65].

1.2 Actual user experiences

It is interesting to look at the experiences of people that already live in a smart home and see how far the visions for the home of the future have already been realized. A number of commercial smart home solutions are available today and several studies have evaluated people's experiences of living in such a home. These studies are discussed in detail in Chapter 2, main findings are shortly summarized here.

Generally, the studied homes lack many of the features that people expect from a smart home. None of the homes are context-aware and consequently the inhabitants struggle with **complicated user interfaces**. They expect that "it should never take longer than it did before" [85, p. 232]. Instead of increased comfort, users today must navigate through complicated menus to perform even simple actions. One user exasperatedly complained that "things must be simpler to do than in a normal house ... I don't want to work through a menu just to turn off the lights" [85, p. 232]. In particular guests that are unfamiliar with the system are often unable to use it at all [13, 74].

The **inflexible configuration** of today's smart homes is another common issue. Rules (e.g. at 8 AM turn on the kitchen lights) and so-called scenes (e.g. at the press of one button, all lights are turned off) must today be manually programmed into the system. Rules and scenes should reflect user routines, but often these routines are subconscious and can not described in the detail necessary to cover all variations [27]. Several smart home inhabitants complained about inconveniences caused by the inflexibilities of rules and scenes [85, 13, 74, 111]. They react by accepting the inconveniences and working around the system [74, 111] or by turning off some rules and scaling back on the automation [13].

A third critical point is the **complicated installation and upgrading** procedure that is currently necessary when equipping a home with smart technology. In each of the surveyed homes, there was either one person with lots of technical expertise that managed the installation or an expensive outside contractor was hired [111, 13, 74]. Such a contractor is needed not only for the initial setup of the smart home system, but also whenever the system is upgraded, e.g. when new devices are installed. As a result, outsourced smart home installations are less often updated than Do-It-Yourself (DIY) installations [13]

1.3 Smart assistants for smart homes

This dissertation proposes two smart assistants for smart homes that address these issues.

1.3.1 A recommender system for smart homes

The first assistant is a recommender system that enables smart homes to become more context-aware. The proposed system recommends *services*, i.e. actions that the system can perform for the user (e.g. turning on lights, closing windows, increasing the heating

or playing a movie). The recommender system interprets the user's current context and generates personalized service recommendations. The system tries to recommend services that would be beneficial for the user in some way, i.e. automate some action that the user would want to perform anyway.

For example, consider that the inhabitant is preparing dinner, opening cupboards and the refrigerator, and using the stove. A large number of services will be fairly useless in this context, such as opening the garage door or closing the skylight. Which service is most useful depends largely on the inhabitant's habits. If the inhabitant likes to listen to the radio while cooking, then a service for turning on the radio might be relevant. If dinner is commonly eaten in the dining area, the smart home may already turn on the lights above the table. The system may also automate some tasks that are problematic for the user, e.g. opening high-mounted cabinet doors. And if it starts raining, the previously irrelevant service for closing the skylight could suddenly become important.

The recommendations can be used to simplify user interfaces by removing complicated menus and instead allow the user to select from only the most relevant choices. They might also be used to improve recognition rates of alternative input methods such as speech recognition, or might even allow the smart home to act autonomously in some situations (e.g. close the skylight because it is raining without being prompted by the user). It is not necessary to manually describe user routines beforehand, which increases flexibility. Instead of having pre-defined rules and scenes, our recommender system observes the user behavior and applies machine learning techniques to identify user habits.

Research problems

- How can the assistant generate useful service recommendations based on current user context, user habits, and preferences?
- How can the assistant model and learn user habits and preferences?
- How can the recommendations be utilized, with the goal of improving the usability of the smart home?

Limitations

The proposed recommender system can currently only be used in single-person households. The reason is that the recommendations are personalized to one inhabitant's context, habits, and preferences. The final chapter of this dissertation considers how the assistant can be extended to multi-person households.

1.3.2 An installation assistant for smart homes

The second assistant is a smart installation assistant that enables the use of our recommender system with minimal effort for configuration. To be able to select a useful service, the recommender system must know what each service can do. Such descriptions of service functionalities can become very complex and often cannot be provided *a priori* by the device manufacturer since they must be localized for each specific smart home.

For example, when reasoning whether opening the window blinds can result in comfortable reading illumination, a number of aspects have to be considered, such as outside lighting, user location, layout of the house, and furniture positions. Having to manually create detailed and localized service descriptions further complicates the installation and upgrading process. Under such conditions it is doubtful that smart homes can appeal to a mass market.

The installation assistant addresses this problem by removing the need for any manual service descriptions. The idea is that the user simply plugs in a new device (e.g. motorized window blinds), starts using it, and lets the installation assistant figure out what the device is doing. The assistant observes what is happening when the device is used and tries to identify patterns that describe the offered service functionalities. After a brief learning period, the generated service descriptions are integrated into the recommender system to provide useful service recommendations.

Research problem

- How can the installation assistant automatically identify the functionalities of services, without the need for manually providing service descriptions?

1.4 Contributions

A number of the contributions of this dissertation have been developed in collaboration with Fei Li (marked in the following as FL) and Sanjin Sehic (marked as SS), both working at the Vienna University of Technology. My own contributions are marked with KR.

The dissertation is based on several publications in peer-reviewed conferences and journals. For each of the following contributions, the relevant publications are listed. A full publication list is given in Section 1.5. The publications do not directly correspond to chapters in this dissertation. Instead, considerable effort underwent to transform the source material into one unified monograph.

Contribution 1: Comparative survey of smart home studies

We compare ten studies of potential and actual smart home inhabitants and identify their general expectations and experiences. In particular, we identify reports where the expectations of potential inhabitants deviate markedly from actual experiences. The results of this survey motivate the need for developing smart assistants for the smart home.

Detailed contributions: All work performed by KR.

Contribution 2: Hyperspace Analogue to Context

We propose a formal model of context called Hyperspace Analogue to Context (HAC). This context model is the basis for all methods developed in this dissertation. HAC describes context in a multi-dimensional space and integrates the concepts of user context, context changes, and context-altering services in one model. A number of operations in the context

space are described using fast geometric abstractions. This dissertation consolidates the core HAC definitions together with HAC extensions that were proposed later (see below) into one consistent context model. Visualizations of important HAC concepts are provided to improve the ease of understanding.

Relevant publications: We first proposed HAC in [61] and it was also published in [86]. HAC was extended with service side effects in [87]. It was extended with information about elapsed times in user contexts in an article called "An unsupervised recommender system for smart homes" (submitted).

Detailed contributions: Idea, initial definitions and operations for HAC by FL. Mathematical inconsistencies in the definitions fixed by KR. Scope combination operation and extension of HAC with time information (elapsed time vector) by KR. Visualizations by KR.

Contribution 3: Smart home recommender system

We propose a recommender system for the smart home that generates personalized service recommendations, based on user context, habits, and preferences. The recommender system incorporates contributions 3a-3e.

Contribution 3a: Utilization of the recommendation results

We specify how the recommendation results can improve the usability of the smart home. We identify obstacles that prohibit the realization of an autonomously acting home, given today's state-of-the-art. We propose a compromise that incorporates aspects of the autonomously acting home and classic smart home control. We also identify performance requirements for the system from a usability perspective.

Detailed contributions: All work performed by KR.

Contribution 3b : Service filtering

We propose a novel algorithm for filtering out irrelevant services based on the user's current context. We show that a significant majority of previous filtering results can be reused whenever the user context changes. Since the proposed algorithm takes advantage of this fact, it can fulfill important performance requirements even in large-scale smart home environments.

Relevant publications: We first proposed the algorithm in [61] and it was also published in [86]. In the original publications, service filtering was performed in the same algorithm as service ranking. In this dissertation, these steps are separated.

Detailed contributions: Idea and initial algorithms by FL. Further revision of the algorithms by FL and KR. Implementation and evaluation by KR.

Contribution 3c: Ranking services based on user habits

We propose a method for learning and emulating an inhabitant's habits in the smart home. The method is unsupervised, i.e. no manual input from the inhabitant is required. The algorithm learns in a training phase, which user actions commonly occur in a given situation and which actions are related to each other. During service recommendation, this knowledge is used to predict which actions are probable given the user's current situation and recent actions. Evaluations on two smart home datasets show that the algorithm produces correct recommendations with 61% and 73% accuracy, respectively.

The material has been submitted as an article titled "An unsupervised recommender system for smart homes".

Detailed contributions: All work performed by KR.

Contribution 3d: Ranking services based on user preferences

We propose a second ranking method that focuses on fulfilling user preferences. We describe how to measure the current distance between users and their preferences, how to measure the impact of executing a service with respect to a preference, and how to use these measures for service ranking.

Relevant publications: We first proposed a method for service ranking based on user preferences in [61] and it was also published in [86]. The material presented here is an enhancement of the previously published method. We first published about the automatic mining of user preferences in [60].

Detailed contributions: Idea and initial algorithms to ranking based on user preferences by FL. Substantial additional development by KR. Implementation and evaluation by KR. Idea and implementation of automatic user preference mining by FL, evaluations by KR.

Contribution 3e: Unified framework of the recommender system

For the recommender system in particular, considerable effort underwent to present the source material in a unified manner. The initial publication [61] speaks of service discovery based on user preferences, whereas our recently submitted article proposes service recommendation based on user habits. When writing this dissertation, we wanted to highlight that these two approaches share a common goal – providing personalized service recommendations to the user. To this end, we devised a general framework for the recommender system with a common service filtering phase and two alternative approaches for the service ranking phase. All supporting material (utilization of recommendation results, identification of performance requirements) applies to both ranking approaches.

Detailed contributions: All work performed by KR.

Contribution 4: Smart installation assistant

We propose an installation assistant for the smart home that automatically learns service functionalities, without any need for manual service descriptions. We describe the data

that is collected by the assistant and propose three algorithms for learning the service preconditions, effects and side effects. We report how we successfully integrated the assistant into a smart home prototype. We evaluate the method using smart home data and synthetic data and show that it can quickly learn the service functionalities.

Relevant publications: We first proposed the method presented in this chapter in [87].

Detailed contributions: All work performed by KR. FL and SS contributed with minor comments.

1.5 Source material

Most of the material in this dissertation has been previously published in the following peer-reviewed conferences and journals:

> Fei Li, **Katharina Rasch**, Hong-Linh Truong, Rassul Ayani, Schahram Dustdar: Proactive Service Discovery in Pervasive Environments. Proceedings of the Seventh International Conference on Pervasive Services, pages 126 – 133 (2010)
>
> **Katharina Rasch**, Fei Li, Sanjin Sehic, Rassul Ayani, Schahram Dustdar: Context-driven Personalized Service Discovery in Pervasive Environments. World Wide Web 14(4), pages 295 – 319 (2011)
>
> Claudio Di Ciccio, Massimo Mecella, Mario Caruso, Vincenzo Forte, Ettore Iacomussi, **Katharina Rasch**, Leonardo Querzoni, Giuseppe Santucci, Giuseppe Tino: The homes of tomorrow: service composition and advanced user interfaces. ICST Trans. Ambient Systems 11(10-12): e2, pages 1 – 12 (2011)
>
> **Katharina Rasch**, Fei Li, Sanjin Sehic, Rassul Ayani, Schahram Dustdar: Automatic Description of Context-Altering Services through Observational Learning. Proceedings of the Tenth International Conference on Pervasive Computing, pages 461 – 477 (2012)
>
> Fei Li, **Katharina Rasch**, Sanjin Sehic, Schahram Dustdar, Rassul Ayani: Unsupervised Context-aware User Preference Mining. Workshop on Activity Context-aware System Architectures at Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI-13), to appear (2013)

Some of the material also appeared in:

> **Katharina Rasch**: Plug and play context-awareness for pervasive environments. Licentiate Thesis (2011)

One article is currently under review:

> **Katharina Rasch**, An unsupervised recommender system for smart homes, submitted (2013)

1.6 Dissertation structure

In **Chapter 2**, the problem domain is introduced in more detail. The chapter starts with a short history of ubiquitous computing and an overview of the technologies that contribute to smart home research. A survey of user expectations and experiences motivates the need for smart assistants. Important ethical aspects of this dissertation are also debated.

Chapter 3 proposes a context model that will serve as the basis for all methods developed in this dissertation. The chapter presents the datasets that are used to evaluate the assistants and it is shown how the data can be converted according to the proposed context model.

Chapter 4 introduces the smart home recommender system. Several ways of utilizing the recommendation results are discussed and performance requirements are identified from a usability perspective. The second part of the chapter presents an algorithm for effectively filtering out irrelevant services, based on the user's current context.

Chapter 5 is dedicated to the first ranking strategy, ranking services based on an inferred model of user habits.

Chapter 6 is dedicated to the second ranking strategy, ranking services based on user preferences.

Chapter 7 presents the smart installation assistant. It is demonstrated how the assistant can be integrated into smart home environments and the proposed learning algorithms are evaluated.

Chapter 8 expands the general overview of smart home research that was given in Chapter 2 by looking specifically at research that is closely related to this dissertation. The chapter covers previous research efforts in the fields of context modeling, service recommendation/service discovery, user behavior prediction and automatic service description.

Chapter 9 gives a summary of the dissertation, discusses limitations of the proposed assistants and identifies opportunities for future work.

Chapter 2

Background

This chapter presents the problem domain in more detail. A short introduction to ubiquitous computing is followed by an overview of the technologies that contribute to smart home research. A survey of user expectations and experiences is used to identify several common problems that motivate the need for smart assistants for smart homes. Finally, ethical aspects of this dissertation are debated.

2.1 Ubiquitous computing

In 1991 Mark Weiser formulated his vision for future computing experiences: "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it." [109, p. 1]. Realizing this vision essentially transforms computing from the "one person with one computer paradigm [..] towards a ubiquitous and pervasive computing landscape" [37, p. 380]. The terms *ubiquitous computing* (Weiser's original term) and *pervasive computing* are today used nearly interchangeably for this paradigm [91].

Initial attempts at ubiquitous computing in the early to mid 1990s disappointed because their technological requirements could not yet be fulfilled. The most prominent was the need for small, cheap microprocessors that could be integrated into everyday objects, so-called actuators and sensors. Actuators offer services for carrying out some action, typically for controlling the environment. A TV actuator, for example, may offer services for turning the TV on or off, or to switch channels. Sensors can retrieve information about the environment, such as temperature, position or medical data.

The second big requirement was the need for wireless communication technologies that allow sensors and actuators to communicate with each other and the supporting software. Even though the processing and communication technology is readily available today, research efforts are ongoing for making it disappear silently into the background.

Context-awareness

Context-awareness is one of the key factors for achieving this aim: "A pervasive computing system that strives to be minimally intrusive has to be context-aware. In other words, it must be cognizant of its user's state and surroundings, and must modify its behaviour based on this information" [93, p. 7].

The term context-awareness was first introduced by Schilit and Theimer in 1994 for a system that "adapts according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time" [95, p. 22]. The first context-aware application even predates this term; in 1992, office employees at Olivetti were equipped with so-called Active Badges that report the wearer's location. For example, the system was used when trying to find a specific person or for routing calls to a phone near the person. Employees were at first reluctant to use the badges due to privacy concerns, but were soon convinced by the benefits of the system (e.g. the rate of misconnected phone calls dropped significantly) and voluntarily continued wearing the badges after the trial period.

Other proposed context-aware applications include guide systems for tourists [1, 19] and reminder systems that display notes on a user's handheld PC when specific context conditions are met, e.g. if the user is at a specific location [12]. A comprehensive survey of context-aware applications can be found in [5].

2.2 Smart homes

Researchers quickly moved to bring the ubiquitous computing paradigm also into the home. One of the first prominent prototypes is Mozer's adaptive house, described in 1998 [77]. Other important smart home projects are MavHome and Casas from the University of Texas, Arlington [26], the MIT PlaceLab [54] and the Gator Tech Smart House at the University of Florida [49]. We will encounter a number of research contributions from these projects in the next section when we present essential smart home technologies.

Generally, research on smart homes can also be found under several alternative terms. Most common is the name *ambient intelligence*. *Domotics* is used mainly in Latin languages [16]. *Smart environments* incorporates also research in other environments, such as smart offices and smart hospitals.

Smart home typology

A typology of smart homes was proposed by Randal et al., sorted by increasing level of intelligence [85]:

Contains intelligent objects The house contains single, stand-alone objects that function with some intelligence, e.g. a smoke detector.

Contains intelligent, communicating objects The intelligent objects function on their own, but can also communicate with each other to increase functionality, e.g. a set of inter-connected smoke detectors that all sound alarm if smoke is detected by one of them.

Connected home Through the use of internal and external networks, the house can be interactively controlled from within the home or remotely.

Learning home The home system records usage information and applies machine learning techniques to the accumulated data. The extracted knowledge is used to anticipate users' needs.

Alert home The home records information about user activities and uses this information to anticipate users' needs.
We consider user activity to be just another type of data that can be recorded by the learning home and feel that there is no need for the distinction between the learning home and the alert home.

As will be seen in Section 2.4, today's commercially available smart home solutions typically fall into the third category of above typology, the connected home. Inhabitants of these homes can use a smart home interface to control for example the lighting, heating and window blinds in their house. Many of these solutions also offer the possibility to control the home remotely. However, these houses have barely any learning capabilities, i.e. home automation has to be manually programmed. A notable exception are some homes that have a thermostat that can learn user routines.

2.3 Contributing technologies

Cook et al. identify five research areas that contribute to ambient intelligence: sensing, reasoning, acting, Human Computer Interaction (HCI), security and privacy [22]. This section concentrates on the first four of these research areas and provides a high-level overview of the state-of-the-art. Research efforts that are closely related to this dissertation are examined in more detail in Chapter 8. Security and privacy are considered separately in Section 2.5 as part of a discussion of the ethical implications of our work.

2.3.1 Sensing

Sensors provide the smart home with important data about the status of the house and its inhabitants. Sensor data can provide an indication of how comfortable the inhabitants are and help detect undesirable situations. In smart homes, environmental sensors (e.g. temperature, humidity) and state sensors (door open/closed) are the most common; an extensive list of common sensors is presented in Section 3.1.1.

Sensor reliability

Unfortunately, sensors do not always function correctly. Sensor data might be wrong or might not be transmitted at all due to a number of issues. For instance, there can be hardware failures, packet collisions, human blockage or interference from microwaves [62]. A number of algorithms try to detect anomalies in sensor data; for an overview of the field see [115]. If anomalies are detected, the system might choose to disregard data coming from the

affected sensor or use an average of the most recent non-faulty sensor values instead. It is also possible to deploy several redundant sensors to reduce the uncertainty by using sensor fusion algorithms (see [67] for a survey).

Indoor positioning

User location has been considered a first-class attribute of user context even in the earliest works on context-awareness. In smart homes, location-awareness can be used to create a comfortable environment whereever the user is currently situated. Location information can also be important to identify which inhabitant is performing some activity in the home in a multi-person household. Since the Global Position System (GPS) does not work inside of buildings, a large body of research has investigated indoor positioning systems. A comparative survey of 17 indoor positioning system is given in [45] and drawbacks of the used technologies are discussed.

Most popular today are systems that make use of existing Wireless Local Area Network (WLAN) infrastructures, as first proposed in the RADAR system [4]. The user is wearing a sensor that can read the signal strength and signal-to-noise ratio of all wireless access points that are installed in the area. During a setup phase, the building is mapped by taking sensor readings at different locations in the building and adding them to a database. To locate a user, the system matches the user's current sensor readings with this database. In the SM4All prototype smart home (see Section 3.7.3), the commercial Ekahau system[1] is deployed to provide location information, which can reach up to one meter resolution. Curran et al. state that location systems "have the constraint of providing lower accuracy over a wide coverage area or providing high accuracy in a small area" [23, p. 61].

Relation to this dissertation

Sensing technology and the reliability of sensors are outside of the scope of this dissertation. The proposed context model supports user location as one attribute of user context, however the proposed assistants do not presuppose that user location is available.

2.3.2 Reasoning

The second technology necessary for building smart homes is reasoning, i.e. to obtain higher level information about system state and user behavior from the low-level sensed data.

Activity recognition

Activity recognition is the process of identifying the high-level activity that a user is currently performing, e.g. preparing a meal, washing the dishes or watching TV. Applications for activity recognition are found mainly in health care scenarios, for instance to tell caretakers whether the inhabitant has performed all the necessary activities of daily living and to make

[1] http://www.ekahau.com

certain that the activities were performed correctly (e.g. the user did not forget to turn off the stove) [79], or to identify declines in the health of the inhabitant [101].

A recent and comprehensive survey of the field is given by Chua in [20, Chapter 2]. Most common are supervised algorithms for activity recognition. During the training phase, the inhabitant lives in the smart home and performs activities just as normal. Sensor data is collected throughout the training phase and annotated with the activity that was performed. Then a model of the user activities is trained using various learning algorithms, e.g. using Naïve Bayes [101], decision trees [63] or Hidden Markov Models [118].

The problem with these approaches is that a large amount of annotated training data is needed to be able to recognize user activities in all possible variations. Obtaining this training data requires either an expensive reviewing and annotation process or involves regularly disturbing users by querying which activity they are currently performing. Semi-supervised activity learning approaches address this problem by trying to adapt models learned in one home to another home [105] or expand activity models to also recognize related activities [119]. Unsupervised methods work without any manual data annotation. The activity learner proposed in [113] mines full-text descriptions of activities from the web, extracts phrases that describe the objects that are being used and maps these to the names of sensors that are attached to common house-hold objects. Other approaches try to find similar patterns in the sensor data to identify ongoing activities, using clustering [48] or compression methods [20].

As Chua points out, comparing these algorithms is difficult since various datasets are used for evaluation [20]. Often researchers use their own datasets that are collected in a laboratory setting or are not publicly available. To our knowledge, no efforts have been made for comparing different approaches on publicly available real-world datasets.

Behavior prediction

If the smart home knows not only where users currently are and what activities they are performing, but also has some idea of what might happen next, then it might be able to proactively automate some of the user's next actions.

A number of researchers concentrate on predicting the future user location, i.e. which room or larger zone in the home will become occupied in the near future. Mozer presents the anticipator as part of the Adaptive Control of Home Environment (ACHE) [78]. The anticipator uses a neural network to predict future locations based on data from recent location data, door status, sound level and time of day. Das et al. present a method called LeZiUpdate that collects a history of user movements and build a probabilistic model in form of a tree of found sub-sequences [26]. Using this model, LeZiUpdate calculates how probable a future location is, given an inhabitants most recent locations.

More powerful methods try to predict arbitrary future events without concentrating on user location. Aipperspach et al. model sensor events as words in a restricted language and use techniques known from natural language processing to predict the next word (i.e. the next sensor event), achieving a prediction accuracy of up to 51% on a smart home test dataset [2]. Gopalratnam and Cook improve LeZiUpdate and apply their ActiveLeZi

algorithm to predict arbitrary user actions, achieving a prediction accuracy of up to 47% on a different dataset [44].

Mayrhofer presents in [71] a framework for context prediction that also integrates pre-processing steps such as feature extraction and classification of sensor data. The focus in this work lies not on providing a specific prediction algorithm, but to allow to use and compare different available prediction algorithms. In contrast to the other works described in this section, Mayrhofer aims to support the prediction of higher-level activities (e.g. meeting, in a concert), rather than of low-level sensor data.

Relation to this dissertation

Chapter 5 presents an algorithm for predicting context changes, a mechanism which is closely related to predicting sensor events. In contrast to the existing methods, this dissertation also proposes a framework that utilizes these predictions to improve the usability of the smart home by transforming predictions into service recommendations. Activity recognition is not addressed, however user activity is supported by the context model. This means that the assistants can make use of activity data if it is provided by any of the available recognition methods.

2.3.3 Acting

The autonomously acting home is part of nearly every futuristic scenario described by smart home researchers, although in reality few have taken up the challenge.

The ACHE system presented by Mozer contains the *controller* that regulates light sources in the smart home [78]. The controller bases its decisions on current and predicted location, current movement patterns, as well as the expected cost incurred by the operation. Costs include the energy costs caused by the lights and the discomfort that is incurred if a device's settings have to be manually adjusted by the inhabitant.

In the iDorm project, a fuzzy controller is used to perform actions on behalf of the inhabitant using ten actuators (lamps, blinds, heater, word processor, media player), based on values from seven sensors (internal/external light and temperature, chair and bed pressure, time) [32]. The system collects data whenever an actuator is used, recording sensor states before the action and actuator states after the action. Fuzzy rules are extracted from this data and enacted when known situations arise.

In the work by Vainio et al., fuzzy rules are used in a similar manner to automatically control the lighting in a home using ceiling lamps and window blinds, based on output from three sensors (outdoor lighting, user location, time) [104].

The lack of long-term evaluations and user-acceptance tests of these methods is problematic. A limited evaluation was performed in the iDorm project using data collected over three days. The authors found that the system performs the correct action with a Root Mean Squared Error of 0.11. For the other two projects, only anecdotal evidence of the system accuracy is reported. Perhaps the evaluation of the behavior prediction algorithms presented in the previous section can serve as an approximation of the currently achievable

accuracy. If the decisions were based on these methods, then the system might be executing the correct action only 50% of the time.

Relation to this dissertation

The feasibility of an autonomously acting home is further discussed in the context of the smart home recommender system in Chapter 4.

2.3.4 Human-Computer Interaction

New and unobtrusive forms of human-computer interaction are needed to make technology disappear into the background. Cook et al. list several natural interfaces for controlling a smart environment [22]. For example, the inhabitant might choose to speak [116], to whistle [35] or to use gestures [83] to give commands to the smart home.

Other interfaces aim to inform the user of the current state of the home. The eWatch [70] is a computerized wristwatch that can be used to provide tactile, audio, and visual notification, and can display the current status of smart home sensors. The Digital Family Portrait [79] was designed to keep caretakers informed of health trends, activity levels, and emotional well-being of their elderly relatives. A computerized picture frame is used to visualize the relative's day at home using available sensor information. Counter Intelligence is a kitchen designed to help people cook more easily and safely [9]. Information such as recipes are projected on objects and surfaces, and everyday objects are augmented with additional data, e.g. the kitchen sink is augmented with a small lamp that makes the water appear red or blue, depending on its temperature.

Several authors propose more intuitive ways of programming the home. Alfred is a macro recorder that can be used in voice-controlled smart homes to record a sequence of commands and store this sequence as a new command [41]. In aCAPpella, an inhabitant demonstrates an activity by performing it several times and lets the system learn (with help from some manual annotation) how to perform the activity autonomously [31]. Finally, two prototypes explore programming with use of physical objects. In [53], jigsaw puzzles represent sensor information and system actions that can be combined to write simple programs. For example, the puzzle pieces *grocery alarm*, *add to shopping list* and *send SMS* together form a program that sends a shopping list of missing items per SMS to the inhabitant's phone. In a design study, small refrigerator magnets with words printed on them are used in a similar fashion to express small programs such as "record 1 picture every 4 minute Billy bed room every night until morning stop" [103] .

Relation to this dissertation

Chapter 4 discusses how service recommendations can help improve smart home user interfaces.

2.4 User expectations and experiences

Several studies examine the expectations and experiences of actual and potential smart home users. In the following these studies are reviewed from several different angles: fear of loss of control, complicated user interface, flexibility, reliability, privacy and security, and installation. The studies are divided into two sets. The first set consists of six studies performed with inhabitants of normal, non-smart homes, we will call these participants potential smart home users. The second set consists of four studies that were performed with actual smart home users.

2.4.1 Studies of potential smart home users

Table 2.1 gives an overview of the studies of persons that do not yet have any experience with smart homes. Most of these studies apply an ethnographic approach. Ethnography is used commonly in anthropology and sociology research to explore the social life of humans. The methodology includes observation of the study participants, interviews and surveys. In these studies, participants are commonly observed in their own home, to identify the characteristics of their normal life.

Authors	Year	Participants	Research questions
Venkatesh et al. [107]	2001	25 households	How is technology currently used in the home and how is life impacted? Which attitudes do users have to future smart homes?
Eggen et al. [34]	2003	Unknown number of families with children	What do people expect from a Smart Home and how would they specify their ideal future home?
Röcker et al. [89]	2005	55 persons	How do users react to four fictional smart home scenarios? What is their ideal home?
Davidoff et al. [27]	2006	12 families	How could a smart home give users more control over their lives?
Allouch et al. [8]	2009	1221 persons	Are users interested in smart homes? How accepting would they be of different smart home applications?
Bonino and Corno [10]	2011	262 persons (mostly technical)	What would users ask their home if it were intelligent?

Table 2.1: Overview of studies of potential smart home users

Interview questions are typically open ended, e.g. Bonino and Corno simply let their participants discuss "What would you ask your home if it were intelligent?" [10, p. 1]. Phrasing questions in such a way removes some of the interview bias that is inherent if researchers ask very specific questions on those smart home technologies they themselves are interested in. Instead the study participants can freely associate about what "Smart home" means to them. Exceptions are the study by Allouch et al., where only a survey was feasible due to the large number of study participants [8], and the work by Röcker et al., where users were presented with four fictional smart home scenarios [89].

Potential users are attracted by the overall convenience that is promised by smart home technologies [107, 34, 8]. Study participants hope that such technologies could save them time and increase their comfort. At the same time there is a fear of becoming lazy [89] or being replaced by the technology [107, 34]. Very common were also sentiments regarding loss of control, which will be discussed further on.

The participants express most interest in applications that allow remote control or automation of home components such as lighting, temperature, or windows [107, 89, 8, 10]. Automation of household tasks such as cleaning [34] and the integration of media devices [89, 8] are also perceived as positive. There seems to be very little interest in specific appliances such as smart TVs or smart refrigerators [107, 8]. According to [8], the perceived loss of control and privacy are highest for these two applications (they were described as autonomously ordering groceries/media content over the Internet) which had a negative affect on the participant's attitude towards these technologies.

Limitations

If users are allowed to freely associate, they are commonly restricted by their own understanding of what would actually be technically feasible [34]. One professional smart home planner remarks "The whole issue of home automation is still so remote. For cars, everyone knows what's possible." [74, p. 156]. Bonino and Corno report in their 2011 study that 64% of the requirements that participants have for their intelligent home can be met already today, but are perceived as distant and futuristic [10].

2.4.2 Studies of actual smart home users and smart home test labs

Table 2.2 gives an overview of a second set of studies which have been performed with actual smart home users or in smart home test labs. All listed studies use ethnographic approaches. The last three studies in the table are especially interesting because they are reporting valuable long-term experiences of actual smart home users. These three studies are shortly summarized here, concentrating on the types of technologies that the participants opted to install in their homes and their experiences.

The participants in Woodruff's study are using smart homes for religious reasons (Orthodox Jews are prohibited from manually turning on or off electronic devices on Sabbath day) [111]. The authors report that some of the families have been using some form of home automation for decades. This group has very specific requirements which can be easily met by making appliances timer-controlled, e.g. the kitchen lights are automatically

Authors	Year	Participants	Research questions
Randall [85]	2003	3 families, short term residency in test smart home (max. 2 weeks)	How are users experiencing the test smart home?
Woodruff et al. [111]	2007	20 families living in smart homes	What are the families real-life, long-term experiences with their smart home?
Brush et al. [13]	2011	14 families/singles living in smart homes	What kind of technologies are installed and what was their cost? What are the families real-life, long-term experiences with their smart home?
Mennicken and Huang [74]	2012	7 professional smart home planners, 7 families living in smart homes, 3 families currently building smart homes	How does the smart home develop – from initial idea, to installation and configuration, to actual usage?

Table 2.2: Overview of studies of actual smart home users

switched on at breakfast time on Sabbath morning. Only some households apply more intelligent technology, e.g. in one household the closing of a skylight is triggered by a rain sensor. The participants were generally satisfied with their systems. However, they state that their primary motivation is religious and that they would probably not buy such systems otherwise. Nonetheless, several households report that they are using their smart home technology also on other days of the week, mostly for controlling lights remotely.

The participants in the study by Brush et al. have been living in their smart home for at least a year, with two households having ten years of smart home experience [13]. The authors identified that there is one person in all of the participant families who is interested in technology and who was the driving force for integrating smart technologies into the family home. Depending on the expertise of this person, both Do-It-Yourself installations and installations done by contractors are common. The most commonly installed technologies were lighting control (11 of 14 households), often with scenes (e.g. turn off all lights, turn on all lights on the first floor), integrated media functionality (11 households), security (10 households) and automatized control of the heating system (8 households). Eight households had remote access to their home installed. For thirteen of the households, convenience was their favorite aspect of living in a smart home. Eleven households liked the security aspects and the peace of mind given by a remote access to home functionalities. For five households the possibility to centrally control home functionalities was very valuable. However, nearly

all the families felt that the technology was not yet ready for broad adoption, due to various reasons that will be discussed.

The study by Mennicken and Huang concentrates on experiences gained while building a smart home. Two of the households have been living in their smart home for six months and the others for at least three years. The authors found three motivations for having a smart home among their participants [74]: one or several persons are interested in technology with home automation as a hobby (4 of 7 households), the desire to live in a modern home (1 household), and potential energy savings (2 households). Additionally, some households extended their installed system after positive experiences (3 households). Again, DIY installations and professional installations were common. All of the households had advanced climate control, several had light or shades controlled by sensors or timers. Five households had remote access to the home and five had programmable scenes.

One interesting outcome of the study is that even though the participants find that their level of comfort increased, they to do not experience the technology as having a huge impact on their life, but rather as minor conveniences. One might argue that these sentiments are exactly what ubiquitous computing strives to achieve – the technology should not majorly change user behavior and routines, but instead unobtrusively assist users in their everyday life.

Limitations

All participant's homes can be called connected. However, none of them can be called learning. One form of automation that can be found is the use of pre-defined scenes that have to be manually activated by the user (e.g. switch off all lamps, fill bathtub and dim bathroom lights). The other form are rules, either timer-based (i.e. perform some action at some specific time) or based on motion detectors and simple environmental sensors (e.g. turn on light if somebody is in the room, close window if it is raining). The only form of learning was reported in [13] where several households had installed a smart thermostat that learns inhabitant routines.

2.4.3 Study results

Fear of loss of control

The main fear of potential users is that they could lose control over their home in some way [107, 34, 89, 8]. Often potential users are not comfortable with the idea that the system autonomously performs some action on their behalf. Eggen et al. strongly stress that "people want control over when and how things are done, and to what degree the home takes over" [34, p. 7]. Other participants worry that there may be no way to manually override any automation [107]. Applications that were perceived with a large control loss were found less attractive than other applications [8].

Somewhat contrary to these studies are the experiences of actual smart home users, where complaints about the home acting autonomously were related mostly to the reliability of the technology, which will be considered later on. But first let us discuss the possible

reasons for why actual smart home users seem to be much more comfortable with the general idea of a smart home than potential users.

The first potential reason is quite obvious – these people are early adopters that consciously decided to buy smart homes, even though the technology is still in its infancy and rather expensive. There usually is one person involved that is very interested in technology and pursues home automation as a hobby [13, 74]. One can argue that these people are more aware of what technology can and cannot do, and are thus more at ease with technology controlling their home. A second reason might be that none of the participant's houses are learning homes, i.e. the home will never be perceived as acting autonomously. It is always tangible to the user, *why* something is happening. They are still in control, because they are the ones that defined those rules and programmed the system accordingly.

User interface

Randall et al. point out that, paradoxically, a user interface that is designed to give users a lot of control can instead make them feel less in control [85]. While their study participants liked the ability to control lights from the couch, the sheer overhead of having to go through a menu to do so was a constant source of irritation. One participant remarked that "there's not an ordinary tap in the house and it drives you mad. You can't control the water volume and it's inconsistent. I really disliked the lack of control." [85, p. 233]. Users expect that "things must be simpler to do than in an ordinary home" [85, p. 232].

Eight participants in [13] cite complex user interfaces as one of the main downsides of their smart home. Especially guests that are unfamiliar with the system are often unable to use it at all; one smart home inhabitant describes how his/her mother sat in the dark during her visit because she was too scared to touch the controls [13]. Similar sentiments are also reported in [74].

Several of the households in [13] use augmentation, i.e. they keep traditional interfaces such as light switches, but also enable inhabitants to remotely control the devices. This method seems to work out as a good compromise, especially for guests. Similar manual controls were also requested by the user's in Randall's study to avoid having to go through the complicated menu whenever they wanted to switch on a light [85].

Flexibility

The smart homes in the studies are based heavily on knowledge of user routines. For scene-based automation, users must decide which devices should be involved in a scene and which actions should be performed. For timer- and sensor-based rules it must be decided how the system should react to any event. Both types of configuration assume that there is some standard user behavior that can be hard-coded into the system.

Routines are indeed important for organizing home life, especially for families with children [102, 27]. However, these routines are not as fixed as is assumed by scene-based and rule-based systems, but instead are highly variable [27]. For example, a cooking routine will vary based on the meal that is being prepared. Davidoff et al. argue that routines are

often subconscious and that they can rarely be described in the detail necessary to cover all variations [27]. They advise that routines can for these reasons not be described *a priori*.

Several problems arose in the studied smart homes because of inflexible rules and scenes. One user discussed, how he scaled back from his initial ideas of a wake scene with music playing and lights switching on automatically because his life is much less structured than implied by the setup of the scene [13]. He concluded "So I don't think the routineness of automation is what I was really wanting." [13, p. 6]. Another participant remarked "I came to discover that you can't really create hard rules to describe every single situation that you might want to automate." [13, p. 6]

One inhabitant mentioned that "It bothers me when it turns on the light ten times and I actually don't need it" [74, p. 156]. Another simply accepts the problems "I just accept that the shades are down and then I just go to the door to look outside" [74, p. 156]. Woodruff et al. even describe, that for participants of their study "the home automation system reflected and shaped the routines, expected behaviors, and social relations of family life" [111, p. 7]. A system that has such influence on the inhabitants to fit their lives into regulated structures will be highly undesirable for many people.

Reliability

Rode et al. formulate the pressing need for reliability in domestic settings:

> *"When domestic technology goes awry it is often more invasive than office technology; not only do we expect our homes to provide a haven of calm and security, but breakdowns in domestic technology can actually prevent us from meeting our basic needs."* [90, p. 1]

One such issue arose in the test smart home in [85] when a cupboard door in the bathroom became stuck open and prevented the bathroom door from being opened. However, the control device displayed that all doors were closed. The family could not enter the bathroom at all. Some long term smart home inhabitants report other instances of system failure. Most smart homes in Woodruff et al. have been in place for years and are generally reliable. Occasional errors did occur, but were treated by the users with calm acceptance, in the spirit of their Sabbath day celebrations[111].

Fourteen participants in [13] report that reliability issues caused unexpected behavior of their smart home. In particular, rule-based systems were described as unreliable, e.g. lights suddenly went off without an apparent reason. In several cases the users felt that the system was hard to debug and they either lived with the problems or switched off the rules.

Privacy and security

Concerns about a loss of privacy rarely came up in the studies. Bonino et al. note that none of the 262 study participants include any privacy considerations when freely associating about intelligent homes [10]. The authors argue that the users are probably unaware of any privacy issues and that "at least in Italy, the home is rarely identified with something that can be open to privacy breaches." [10, p. 14]. Nevertheless, researchers should be aware

of potential privacy issues, in particular when employing algorithms for user profiling and personalization.

Remote access to the smart home is popular with actual smart-home inhabitants [13, 74]. This technology allows users to remotely control household devices and check the status of the home. It is used mostly to deter burglars by making the house seem occupied when traveling and to ease the mind of the inhabitants. However, seven of the eight households in [13] that have remote access worry about security threats. Some inhabitants chose to disable technologies they deemed unsafe, such as remote access to the front door of the house.

Installation

Smart home technology is most commonly set up when building or re-modeling a house [13, 74]. At that point it is easiest and cheapest to install the necessary wiring. At the moment, only a minority of households have retrofited installation that utilize wireless technologies [13]. Cost for DIY installations (min=$US200, max=$US50,000, median=$US5,000) is typically less than for outsourced installations (min=$US13,500, max=$US120,000, median=$US40,000) [13]. Especially DIYers were reluctant of buying all technology from one specific vendor, for fear of being locked in to that vendor [13]. Instead, people want flexible, future-proof solutions [74]. However, trying to integrate technology from different vendors is problematic as well. Eight of fourteen households in [13] report that they have some devices that they could not get connected to the main system.

The installations of smart homes typically happens in iterations [13, 74]. In the beginning the configuration changes quite often and people report being frustrated with the initial chaos [74]. Outsourced installations were more static than DIY installations after the initial setup [13]. Inhabitants report that they rarely call in a consultant but that they do not mind the extra costs involved [13]. However, some users are frustrated with their inability to fix the system on their own [13]. In DIY installations, one inhabitant often pursues home automation as a hobby and regularly makes changes to the installation [13, 74]. Unfortunately it is unclear from the studies whether any iterations after the initial setup are concerned more with hardware issues (e.g. adding devices, moving devices) or with configuring the system (e.g. new rules, modify rules).

2.4.4 Discussion

Complicated user interfaces and inflexibility vs fear of control loss

The issues of complicated user interfaces, inflexibility, and fear of control loss are inextricably linked with each other. Today's smart homes are inflexible because users have to program rules and scenes in a process that they have control over. The interfaces are complicated because the smart home gives inhabitants full control and the full selection of services at any given moment.

The more control users are willing to give up, the more flexible the home can become. Activity recognition and behavior prediction techniques can automatically learn habits and

needs of the inhabitants. User interfaces also can be simplified, showing only the functionality needed right now. With enough information about the user situation and habits, the home might even start acting autonomously, foregoing user interfaces completely. Interestingly, inflexibility and complicated user interfaces rarely concern potential smart home users. The study participants seem to assume a truly smart, self-learning and autonomously-acting home, which triggers their fear of control loss.

The smart home recommender system addresses these issues by following the strategy outlined in the previous paragraph. Since the system automatically learns user habits, no manual description of routines is necessary, which increases the flexibility. The system continuously produces service recommendations that are tailored to the user's current context situation and updated whenever the context changes. We do not prescribe one specific way of utilizing the recommendation results, but instead several such strategies are presented, so that users themselves can decide how much control they are willing to give up for an increase in simplicity and comfort.

Complicated installation

The complicated installation procedure is a critical obstacle that must be overcome before smart home technologies can appeal to a mass market. Surprisingly, the proposed installation assistant is one of the very few approaches that address this problem. This is despite the fact that Edwards and Grinter have identified installation issues as the first three of seven major challenges for smart home research, in their widely cited article from 2001 [33]:

(i) Edwards and Grinter argue that most consumer homes will be "accidentally smart", i.e. a smart home that is not thoroughly planned from the beginning (like most prototype homes are) but instead where new technology is added bit-by-bit over a long time-frame.

(ii) They stress the need for interoperability between different device vendors.

(iii) They point out that there are no systems administrator in homes and that it cannot be expected that consumers can (or want to) administrate their smart home.

The installation assistant addresses challenges *(i)* and *(iii)*. Any smart home solution needs to have some knowledge about the sensors and actuators that are deployed in the environment. In particular, it is important that the smart home knows which actions (i.e. services) it can perform. The recommender system needs information about service functionalities to be able to reason whether a service can be executed in the current situation and whether this service could be beneficial for the user. Unfortunately, these functionalities often have to be localized for a specific smart home and cannot be provided by device manufacturers.

It is unreasonable to expect that consumers manually describe functionalities or call a technician whenever they install a new device in their home. The proposed installation assistant addresses this problem by automatically learning the functionalities. With the installation assistant, people can upgrade their homes bit-by-bit, without having to act as system administrators. The main objective for this assistant is to simplify the installation of

the recommender system. However, the generated descriptions can also be used by other products that need information about service capabilities. This information is needed, for example, to compose several simple services in the smart home to achieve some higher-level goal [56].

The assistant does not address any low-level integration issues between different smart home protocols (challenge *(ii)* from above). However, it poses only minimal requirements on device manufacturers and some standard smart home protocols already fulfill these requirements.

2.5 Ethical considerations

2.5.1 Privacy

Privacy concerns invariably arise in ubiquitous environments where tiny sensors silently collect data about the user. In smart homes, data can be provided by numerous types of sensors. Some sensors sound harmless on their own (e.g. status sensors that detect if windows, doors, etc are open), but several of these together can be used to build very detailed models of user activities and habits. Other smart home sensors inherently evoke privacy concerns, e.g. medical sensors that measure blood pressure and heart rate.

Smart home data is utilized by the proposed assistants in the following manner:

- The recommender system always keeps track of the current user context (which incorporates information coming from all installed sensors and actuators). It needs only the most current and the second most current user context, i.e. it does not require storing the user context in a raw format.
- The recommender system also uses the current context to develop a model of the user's habits. While it does not store raw context data, the stored model contains very detailed information about the inhabitant's behavior at home. Making this model available to anyone else could be seen as highly invasive of the inhabitant's privacy.
- The installation manager keeps a history of user context and context changes that happen whenever a device is being used. This means that snapshots of raw context data are stored whenever there is activity in the home. This data could potentially be used to reconstruct a full picture of the inhabitant's daily life. Again, any outside access to this data could be seen as highly invasive.

Meyer and Rakotonirainy identify two major strategies for achieving privacy in ubiquitous computing environments [75]: *(i)* heavily restrict the amount of information that gets outside the system or *(ii)* from the beginning limit the amount of information that is being collected and stored. From the listing above it is obvious that the second strategy is not an option in our case; even though the assistants do not store the full raw data, they do store very detailed data that needs to be actively protected.

Our general privacy policy is that the data collected by the assistants should be confined to one home and should not be accessible from outside the home. In particular, we do not

apply any transfer-learning paradigm, i.e. combining data collected in several homes with the intention of creating better models. In fact, it is questionable if collaboration between different homes can be a useful strategy in our case since the assistants are fitted specifically to one home and (at the moment) one inhabitant.

However, privacy concerns do not stop at the doorstep. Already the idea of being observed at all times in their own home will feel strange to many people. The discomfort will be even larger if the home stores the sensor data, possibly over the course of several years. Additionally, at least some information about typical user habits will invariably be shared with other inhabitants and with guests. However, ideas of what is private and what is public differ across cultures, persons, context, social roles [47, 69]. For this reason, privacy settings should be customizable by the users.

We use a framework proposed by Belotti and Selen [7] to identify how different privacy needs can be accommodated by the smart assistants. Central in the framework are the issues of *feedback* (letting users know what data is being captured, stored, how it is used, who can access it) and *control* (allowing users to set their own preferences for these matters). In Table 2.3 this framework was applied to the proposed assistants. Please note that the feedback and control mechanisms listed in the table are theoretical considerations of how to ensure privacy in the context of our work; they have not yet been implement in the prototypes. Finally, it should be noted that this analysis only considers the actual inhabitants of the home. It is in their responsibility to inform their guests that sensor data is being captured and stored.

2.5.2 Addressing important human needs

Smart home technology can help elderly people and people with disabilities to achieve tasks that they previously could not do [29]. The minor conveniences experienced by today's smart home inhabitants can for some people be the key to a more independent life.

However, one should be alert to not fall into the trap that Dewsbury calls the *technologization of needs* [30]. There are a number of basic human needs that simply cannot be met by technology, such as emotional, social, and psychological needs. The smart home can only be one part of an overall health strategy that addresses all user needs. However, "individualisation and personal preferences can be lost as care providers respond to financial constraints or localised restrictions" [29, p. 8]. This can lead to issues such as the *illusion of independence* [68] – if patients are mobile and can perform all activities of daily living, they are seen as more independent then they actually are, even though basic human needs are not met.

2.6 Summary

This chapter positioned the dissertation within existing ubiquitous computing and smart home research. Based on eleven smart home studies, a number of usability issues that trouble smart home inhabitants were identified and it was outlined how the proposed smart assistants can address some of these issues. Finally, the chapter debated important ethical considerations.

Feedback about	**Control over**
Capture – what kind of sensor and status data is being collected by the home?	
Device manufacturers should inform users what kind of data is provided by the device, including the data resolution. The smart home system must allow users to view the data the system has currently captured (i.e. the current user context).	Device manufacturers should include a mechanism disallowing the device to collect data. If such a mechanism is not available, the user must be able to instruct the smart home system to ignore data coming from the device.
Construction – in what form is the collected data stored by the home?	
Provide users an overview of the data that is stored in the system, include high-level statistics. Allow users to browse all of the stored data.	Allow users to exclude specific data from being stored (e.g. health data), to delete some stored items or purge all old data.
Purpose – how do the assistants utilize the stored data?	
When installing the smart home system, inhabitants should be informed of the basic functionality of the assistants. A graphical user interface for the recommender system could include additional information about why the assistant recommends a service.	Allow users to completely turn off the assistants. Allow them to exclude specific stored data from utilization.
Accessibility – who can access the stored data and any generated high-level information?	
Inform user about current access rights for inhabitants and guests.	Allow user to configure, which inhabitant can browse the stored data. Allow also to configure, which users interfaces are personalized with recommendations; otherwise a generic interface (i.e. guest interface) is shown.

Table 2.3: Feedback and control for four privacy concerns
(using framework proposed by Belotti and Selen in [7])

Chapter 3

Hyperspace Analogue to Context (HAC)

In the introduction it was stressed that smart homes must be context-aware, i.e. they must be attentive to the current situation and the current needs of the inhabitant. This chapter presents a model for user context called Hyperspace Analogue to Context (HAC). This context model serves as the basis for all methods developed in the remainder of this dissertation. This chapter also introduces the smart home datasets that will be used to evaluate the recommender system and the installation assistant.

3.1 Requirements

The chapter starts off with a requirements collection, aimed at identifying the concepts and relationships that a context model for the smart home must be able to express. We look at the sources of context information in the smart home, discuss the variety of the data provided by them, and explain how the user context is formed from this information. It is also examined how context changes are triggered and how services that can automate user actions relate to the context.

3.1.1 Context sources

Sensors

Sensors are the main source of context information in smart homes. An overview of common smart home sensors is given in Table 3.1. This list was compiled by Alam et al. [3] [1], their work is simply reproduced here with a different sensor categorization.

Environmental sensors can be installed inside and outside the house. Placed inside, they can give some idea of the comfort level of the inhabitant, e.g. is it too cold, too loud or too dark? Environmental data collected from sensors placed outside the house can influence the decision making for the indoor environment, such as deciding to close the windows

[1] Section IV A and Table 1 in [3], pages 1196-1197

Environmental
Sensors for *luminosity*, *temperature*, *humidity*, *rain*, *noise*, *smoke*, etc
Home usage
Switch sensors detect if doors, windows, drawers etc. are open or closed
Water flow sensors detect if shower, washing machine, etc. are in use
Electrical current sensors identify, which lamps, electrical devices or power outlets are in use
Object usage sensors in form of RFID (Radio-frequency identification) tags attached to home objects (e.g. a pan), user needs to wear an RFID glove to detect which objects are in use; alternatively the objects are placed near an RFID receiver, if they can not be detected, they are implicitly in use
User location
Indoor positioning system as described in Section 2.3.1
Medical information
Sensors for user's *body temperature*, *weight*, *pulse*, *blood pressure*, etc

Table 3.1: Types of sensors used in smart homes, reproduced from [3]

because it is too cold outside. The sensors in the home usage category can be indicators of user activities. For example, if the pan is in use and the stove is drawing a lot of current, the smart home may infer that the user is cooking. User location can be an additional source of information about activities. Sensors for medical information are most relevant in assisted living environments, to keep a log of the patient's health and to identify when medical assistance is necessary.

Actuators

Any knowledge about the inhabitant's situation can only be useful if the system has the means to react to it in some way. The smart home must have some kind of control over parts of the home infrastructure and other devices. A list of smart home actuators that offer such control is presented in Table 3.2.

Home infrastructure control
Lighting turn lights on, off or dimm them
Doors/windows/blinds etc. motorized opening and closing
Climate control heating and air conditioning
Other device control
Entertainment devices turn on/off, change channel, play/pause, change the volume, etc
Bed motorized bed allows to change sleeping/sitting positions

Table 3.2: Types of actuators used in smart homes

The first category contains any mechanical or electrical devices that are fitted to the home infrastructure itself. This category includes remote-controlled motors that are installed

such that they can automatically open doors, windows, drawers or other mechanical home fittings. Another example are remote-controlled light switches that are mounted inside the light fittings or electrical outlets. The second category contains any other devices that are not part of the home infrastructure. These are mostly entertainment devices, such as TVs, radios and media centers. Some specialized devices, such as motorized beds, are common in assisted living environments.

The internal state of actuators can be an additional source of context information. For example, a motorized door opener may save an internal state, registering whether the door is currently open or closed; the internal status of a TV could be the channel that is currently being watched. Several industry standard protocols for smart homes such as X10 [114] and UPnP [55] allow devices to publish their internal state. This information can complement the data collected by smart home sensors.

Other context sources

Additional context information can come from external sources, for example a web service that provides information about the current weather in the area. Time is another important aspect of user context, since often user activities are correlated with the time of the day or the day of the week. High-level context data such as user activity can be reasoned from low-level context data (see Section 2.3.2).

Sensors
environmental, home usage, user location, medical information
Internal state of actuators
environmental, home usage
External sources
environmental current weather and weather forecast, e.g. outside temperature, chance of rain
location phone GPS signal (outdoors)
current time
High-level context
user activity reasoned from low-level context data or provided by the user

Table 3.3: Summary of smart home context sources

Table 3.3 summarizes the different context sources and lists the types of context information that can be provided by them. There is some overlap between the sources, e.g. weather information can be provided by an external web service as well as by environmental sensors installed outside the smart home. Often, the level of detail and accuracy differ, e.g. a locally installed sensor will give more accurate temperature information than a remote web service.

Data types

The data from the context sources is either numeric (e.g. temperature) or drawn from a set of nominal values (e.g. the rooms of the house). Sometimes it is necessary to transform

the raw data into a more meaningful data format. For example, information about the day of the week or the time of the day (e.g. morning, lunch, afternoon, evening, night) will be more useful in a smart home setting than the raw current date and time information. Another example is blood pressure; some levels such as *low, normal, high, critical* may be interesting in addition to the raw values. The context model must be able to integrate the different data types and must be able to accommodate different representations of the same data.

3.1.2 User context and context changes

To describe the current context of a user, information from all installed context sources is aggregated. Generally, the more context sources are available, the more detailed is the knowledge that the home system can gather about the user's comfort level and ongoing activities. More detailed knowledge improves in turn the system's ability to identify and react to any undesired situations.

The user context changes frequently. Many context changes happen without any user influence, in particular for environmental and medical context sources. For example, when the sun rises, a luminance sensor will report increased brightness.

Context changes can also be triggered by user actions[2]. Opening a window could cause several context changes: if the window publishes its internal state, this state will now read "open", alternatively this information may also be provided by a switch sensor. Any temperature sensors nearby the opened window will typically also register some changes in the temperature. Changes in noise level or humidity are also likely.

It is important to realize that user actions cannot be observed directly by the smart home system. Instead, only the context changes caused by the action can be observed. If the user context is sufficiently detailed, the system may use the observed changes to reason whether any action occurred and which action it was.

3.1.3 Service precondition and effect

The installed actuators offer services for automating some of the user actions. The execution of a service may trigger context changes, we say that the service has some specific effect. Wu et al. first coined the term "world-altering" for services that have some effect on the world [112]. We are adopting this term with some minor adjustment and call services that have an effect on the user context "context-altering" services. Often, there will be a direct mapping between a user action and a context-altering service, e.g. manually opening a window will result in the same context changes as calling a service to automatically open it.

Information about service effects is needed to identify useful services given the current situation, i.e. identify those services that change the user context in the desired way. For example, if the user wants to read in the living room, but the room is dark, then any service that results in increased light could be desirable. Additionally, the smart home must be able

[2]Please note the difference between *user action* and *user activity*: Any user activity is comprised of a number of shorter user actions, e.g. a meal preparation activity is made up of several small actions: opening the refrigerator, taking out a prepared meal, opening the microwave, putting in the meal, closing the microwave, etc.

to filter out any services that can currently not be executed. Switching on a lamp is only an option, if the lamp is currently turned off. We say that the service has a precondition, i.e. a context condition that must be fulfilled, otherwise the service is not a viable option.

3.1.4 Summary of the requirements

Based on these observations we identify several requirements for a context model for the smart home. Such a context model must

- integrate various types of context information,
- support descriptions of the current user context and context changes,
- allow to formulate preconditions and effects of services in terms of user context,
- provide context operations, e.g. for checking if service preconditions are fulfilled

In a following a context model is presented that addresses these requirements. The model describes context as a multi-dimensional space that integrates various types of context data. User context and context changes are multi-dimensional points in the space, service preconditions and effects are subspaces of the original space. The relationships between context descriptions are characterized by geometric structures, which greatly improves the performance when reasoning about situations. We start by describing fundamental concepts of the model in Sections 3.2-3.4. We then build on these concepts to define context-altering services (Section 3.5) and context operations (Section 3.6).

3.2 Hyperspace Analogue to Context (HAC)

The proposed context model, which is called Hyperspace Analogue to Context (HAC), models context as a multidimensional space. HAC was inspired by Hyperspace Analogue to Language (HAL) [66], which defines a multi-dimensional language model [3].

Acknowledgements

The idea, initial definitions and operations for HAC are courtesy of Fei Li from the Vienna University of Technology and were first proposed in a paper co-authored by me [61].

Definition 1 (HAC) *An n-dimensional HAC $\mathbb{H}$ is a multi-dimensional space formed by an array of context dimensions $\mathbb{H} = [\mathbb{D}_0, \ldots, \mathbb{D}_{n-1}]$, where each dimension $\mathbb{D}_i$ corresponds to one context attribute.*

Definition 1 formally defines HAC as a multi-dimensional context space. Each dimension is the meta data to describe the data type and value set for a context attribute. Dimensions can be numeric or nominal. For numeric dimensions, the values are drawn

[3] In HAL, each word that occurs constitutes one dimension, a co-occurrence vector for a word marks how often this word occurs nearby any of the other words. Geometric operations can be used to identify words with similar meaning.

from some numeric interval, e.g. the FM frequency of a radio can be represented as dimension $\mathbb{D}_{fm\_frequency} = [88.0, 108.0]$ (MHz). For nominal dimensions, only a limited set of non-numeric values are allowed, e.g. $\mathbb{D}_{blinds} = \{open, close\}$. A special bottom element $\perp$ represents undefined values. The bottom element will later be used to mark missing or irrelevant context information.

Some examples for context dimensions that are described in this manner can be found in Table 3.4. The table contains dimensions formed on raw data (e.g. $\mathbb{D}_{temp1} = [10.0, 30.0]$), as well as dimensions formed on transformations of the same data (e.g. $\mathbb{D}_{temp2} = \{cold, chilly, comfortable, warm, hot\}$). Both dimensions $\mathbb{D}_{temp1}$ and $\mathbb{D}_{temp2}$ can be part of the same HAC.

location
$\mathbb{D}_{location1} = \{bedroom, kitchen, bathroom, hallway, garden\}$
$\mathbb{D}_{location2} = \{indoors, outside\}$
time
$\mathbb{D}_{time\_of\_day} = \{morning, lunch, afternoon, evening, night\}$
$\mathbb{D}_{day\_off\_week} = \{monday, \ldots, sunday\}$
temperature (indoor)
$\mathbb{D}_{temp1} = [10.0, 30.0]$
$\mathbb{D}_{temp2} = \{cold, chilly, comfortable, warm, hot\}$
radio
$\mathbb{D}_{fm\_frequency} = [88.0, 108.0]$
$\mathbb{D}_{radio\_channel} = \{off, ch1, ch2, ch3, ch4, ch5, ch6\}$
$\mathbb{D}_{radio\_volume} = [0, 20]$
window blinds
$\mathbb{D}_{blinds} = \{open, close\}$
$\mathbb{D}_{blinds\_open\_percentage} = [0, 100]$

Table 3.4: Examples of context dimensions

Running example

Throughout this chapter, important concepts are visualized on an example HAC with six dimensions. Figure 3.1 illustrates this HAC as a parallel coordinate system. Each of the vertical axes in the figure represents one context dimension. The first four dimensions (user location, status of kitchen window, status of kitchen blinds and status of radio) are nominal, the values that the dimension can take are marked as boxes on the axis. The final two dimensions (outside temperature, temperature in the kitchen) are numeric and are represented as a scale from the minimum value to the maximum value recorded for this dimension. A formal declaration of the example context space according to Definition 1 is given in the caption to Figure 3.1.

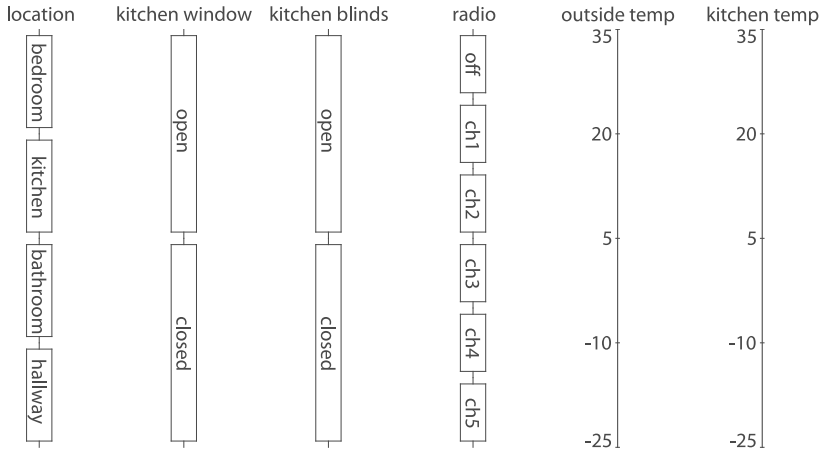


Figure 3.1: Example HAC with six dimensions

$\mathbb{H} = [\ \mathbb{D}_{location} = \{bedroom, kitchen, bathroom, hallway\},$
$\mathbb{D}_{kitchen\_window} = \{open, closed\}, \mathbb{D}_{kitchen\_blinds} = \{open, closed\},$
$\mathbb{D}_{radio} = \{off, ch1, ch2, ch3, ch4, ch5\}, \mathbb{D}_{outside\_temp} = [-25, 35],$
$\mathbb{D}_{kitchen\_temp} = [-25, 35]\ ]$

Notation

Following notation will be used throughout this chapter for all context dimensions $\mathbb{D}_i$:

Numeric $\mathbb{D}_i$: the boundaries of $\mathbb{D}_i$ can be accessed using $\mathbb{D}_i.min$ and $\mathbb{D}_i.max$
e.g. $\mathbb{D}_{outside\_temp}.min$=-25
$v \in \mathbb{D}_i$ denotes that a value v lies in $\mathbb{D}_i$, $v \in \mathbb{D}_i \Leftrightarrow \mathbb{D}_i.min \leq v \leq \mathbb{D}_i.max$
e.g. $20 \in \mathbb{D}_{outside\_temp}$ and $70 \notin \mathbb{D}_{outside\_temp}$

Nominal $\mathbb{D}_i$: $v \in \mathbb{D}_i$ denotes that a value v is an element of the set defined by $\mathbb{D}_i$
e.g. $bedroom \in \mathbb{D}_{location}$ and $moon \notin \mathbb{D}_{location}$

3.3 Context points

3.3.1 Definition

A context point is a vector in the space $\mathbb{H}$ that assigns specific values to some of the available dimensions. Let p_i be the value that is assigned to dimension i. The bottom element $\perp$ marks missing/undefined values, i.e. assign $p_i \leftarrow \perp$ instead of a specific value, if dimension i is not relevant for the context point. Then a context point c can be defined according to Definition 2.

Definition 2 (Context point) *A context point c is a vector in an n-dimensional* $\mathbb{H}$*;* $c = [p_0, \ldots, p_{n-1}]$*, with* $\forall\ 0 \leq i < n : p_i \in \mathbb{D}_i$ *or* $p_i \leftarrow \perp$.

For the example HAC with dimensions [$\mathbb{D}_{location}$, $\mathbb{D}_{kitchen\_window}$, $\mathbb{D}_{kitchen\_blinds}$, $\mathbb{D}_{radio}$, $\mathbb{D}_{outside\_temp}$, $\mathbb{D}_{kitchen\_temp}$] a context point $c^1 = [\perp, closed, \perp, \perp, \perp, 22]$ expresses that the kitchen window is closed and the kitchen temperature is 22°C. No values are assigned to any of the other dimensions. A second context point $c^2 = [\perp, closed, \perp, \perp, \perp, 55]$ is not valid, since the kitchen temperature must fall into range $[-25, 35]$, according to the definition for $\mathbb{D}_{kitchen\_temp}$.

Alternative representation

Often the resulting array will be sparse, i.e. it contains mostly missing values. It can also be difficult for the reader to identify, which value in the array belongs to which context dimension. For these reasons, this dissertation mainly uses an alternative context point notation as an associative array, i.e. a collection of *key:value* pairs. For example, the context point $c = [\perp, closed, \perp, \perp, \perp, 22]$ in form of an associative array would be $c = \{kitchen\_window: closed, kitchen\_temp: 22\}$. Addressing by index ($c[1] = closed$), as well as addressing by dimension name ($c[kitchen\_window] = closed$) are supported for a context point. A summary of notations and addressing can be found in Table 3.5.

Notation using list array
$c = [\perp, closed, \perp, \perp, \perp, 22]$
Notation using associative array
$c = \{kitchen\_window: closed, kitchen\_temp: 22\}$
Addressing individual elements
$c[0] = c[location] = \perp$
$c[1] = c[kitchen\_window] = closed$

Table 3.5: Summary of context point notations and addressing

3.3.2 Current user context

The current context of a user aggregates the current values reported by all context sources in the home. For example, say that the user is in the kitchen, window and window blinds are closed, the radio is set to channel 2, outdoor temperature is -14°C and indoor temperature is 22°C. Then user context $c^u = \{location: kitchen, kitchen\_window: closed, kitchen\_blinds: open, radio: ch2, outside\_temp: -14, kitchen\_temp: 22\}$ is a context point in $\mathbb{H}$. This user context is visualized in Figure 3.2; the vector in $\mathbb{H}$ that is described by c^u is illustrated as a line through the context space. The point of intersection between line and an axis marks the current value for the dimension represented by the axis.

3.3.3 Changing the user context

A context change moves the user from one context to another. We commonly use Δc to denote a context change and set in Δc the new values for all changed dimensions. For example, let the user turn off the radio, then $\Delta c = \{radio: off\}$.

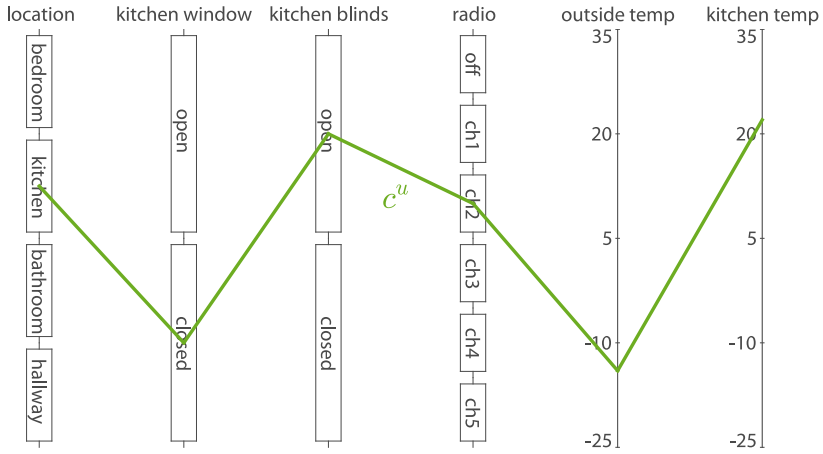


Figure 3.2: The current user context c^u as a vector in $\mathbb{H}$
$c^u = \{location: kitchen, kitchen\_window: closed, kitchen\_blinds: open, radio: ch2, outside\_temp: -14, kitchen\_temp: 22\}$

Definition 3 (Context change operation $\times\Delta c$) *Let $\mathbb{H}$ be an n-dimensional HAC and c and Δc be context points in $\mathbb{H}$. The result of the operation $c \times \Delta c$ is context point c' in $\mathbb{H}$ with $\forall\, 0 \leq i < n:$ if $\Delta c[i] = \perp$ then $c'[i] \leftarrow c[i]$ else $c'[i] \leftarrow \Delta c[i]$.*

The context change operation $\times\Delta c$ in Definition 3 is used to update the user context. The context change operation overwrites for any changed dimensions the values in the current user context with the updated values. Figure 3.3 demonstrates, how the user context from Figure 3.2 changes, when the radio is turned off. The new user context c'^u is given in the caption of Figure 3.3.

3.3.4 Modeling time information

In Chapter 5 it is also necessary to know, how long the user has been in the current situation. An additional vector, the elapsed time vector Δt^u, is used to record for each dimension how much time has passed since the user context changed on this dimension. The elapsed time vector is formalized in Definition 4.

Definition 4 (Elapsed time vector) *Let c^u be a context point representing the current user context in an n-dimensional $\mathbb{H}$. The elapsed time vector Δt^u is an array of length n, $\Delta t^u = [\delta t_0, \ldots, \delta t_{n-1}]$; with $\forall\, 0 \leq i < n:$ if $c^u[i] \neq \perp$ then δt_i is the time elapsed since $c^u[i]$ has changed, else $\delta t_i \leftarrow -1$*

The elapsed time vector should only be interpreted together with the user context. For example, in Table 3.6 a c^u and a Δt^u together express that the user has been in the kitchen for 220 seconds, the kitchen window has been closed for 4000 seconds, the kitchen blinds

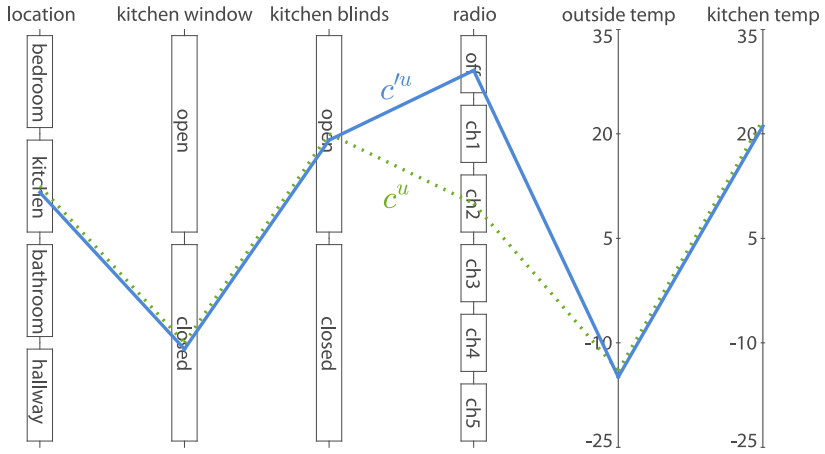


Figure 3.3: Old context c^u (dotted line) and changed context c'^u (solid line)
$\Delta c = \{radio: off\}$
$c'^u = \{location: kitchen, kitchen\_window: closed, kitchen\_blinds: open,$
$\mathbf{radio: off}, outside\_temp: -14, kitchen\_temp: 22\}$

been open for 1340 seconds, the kitchen radio has just been turned off 5 seconds ago. Additionally, outside temperature has been stable at -14 °C for 630 seconds and kitchen temperature has been 22 °C for 380 seconds. The elapsed time vector can also be written as associative array and addressing by index as well as addressing by dimension name are supported, see Table 3.6.

User context
$c^u = \{location: kitchen, kitchen\_window: closed, kitchen\_blinds: open$
$radio: off, outside\_temp: -14, kitchen\_temp: 22\}$
Elapsed time vector (in seconds)
$\Delta t^u = [220, 4000, 1340, 5, 630, 380]$
Alternative notation for elapsed time vector)
$\Delta t^u = \{location: 220, kitchen\_window: 4000, kitchen\_blinds: 1340$
$radio: 190, outside\_temp: 630, kitchen\_temp: 380\}$
Addressing individual time vector elements
$\Delta t^u[0] = \Delta t^u[location] = 220$

Table 3.6: Example for elapsed time vector and summary of notations and addressing

There is not preset time resolution, instead the person that performs the context modeling and creates the HAC can freely choose the best fitting resolution. Throughout this dissertation, the elapsed time vector is measured in seconds. The reason for this decision is that the elapsed time vector is mainly used to describe user behavior. A resolution in minutes has the risk of being too coarse-grained to properly model the temporal relationship

between two user actions. On the other hand, a resolution in milliseconds is much too fine-grained for a human-centric computing environment. The issue of time resolution will be revisited in Section 5.3.2.

3.4 Context scopes

A context scope is a subspace of the full multi-dimensional HAC. The context scope may make use of only a subset of all dimensions. For each of the dimensions contained in the subspace, the context scope can restrict the allowed value range. For a dimension $\mathbb{D}_i$, a restriction r_i is:

Numeric $\mathbb{D}_i$:	r_i is an interval $[r_i.min, r_i.max]$, $\mathbb{D}_i.min \leq r_i.min$ and $r_i.max \leq \mathbb{D}_i.max$
	$v \in r_i$ denotes that a value v lies in r_i, $v \in r_i \Leftrightarrow r_i.min \leq v \leq r_i.max$
Nominal $\mathbb{D}_i$:	r_i is a subset of the original value set of $\mathbb{D}_i$, $r_i \subseteq \mathbb{D}_i$ and $r_i \neq \emptyset$
	$v \in r_i$ denotes that a value v is an element of the set r_i

The symbol $\perp$marks that a dimension is not part of the subspace. Definition 5 then formally defines a context scope C.

Definition 5 (Context scope) *A context scope C is a subspace of an n-dimensional* $\mathbb{H}$*;* $C = [r_0, \ldots, r_{n-1}]$*, with* $\forall\, 0 \leq i < n : r_i$ *is a restriction of* $\mathbb{D}_i$ *or* $r_i \leftarrow \perp$.

For example, a context scope $C^1 = [\{kitchen\},\ \perp,\ \perp, \{ch1, ch2\}, \perp, [12, 28]]$ forms a subspace that contains only three dimensions (location, radio, kitchen_temp), and each of the three dimensions are restricted to some subrange of their original range. For the same reasons as before, mainly an associative array notation will be used. In this notation $C^1 = \{location : \{kitchen\}, radio : \{ch1, ch2\}, kitchen\_temp : [12, 28]\}$. Again, addressing by index, as well as addressing by dimension name are supported. An overview of the different notations and addressing possibilities is given in Table 3.7.

Notation using list array
$C^1 = [\{kitchen\}, \perp, \perp, \{ch1, ch2\}, \perp, [12, 28]]$
Notation using associative array
$C^1 = \{location : \{kitchen\}, radio : \{ch1, ch2\}, kitchen\_temp : [12, 28]\}$
Addressing individual elements
$C^1[0] = C^1[location] = \{kitchen\}$
$C^1[1] = C^1[kitchen\_window] = \perp$

Table 3.7: Summary of context scope notations and addressing

Figure 3.4 visualizes the scope C^1 compared to the full space $\mathbb{H}$. The three dimensions that are marked with $\perp$ and thus not part of the subspace are completely grayed out. For each of the three dimensions that are in C^1, the data ranges that form the subspace are marked black, all other data ranges are grayed out.

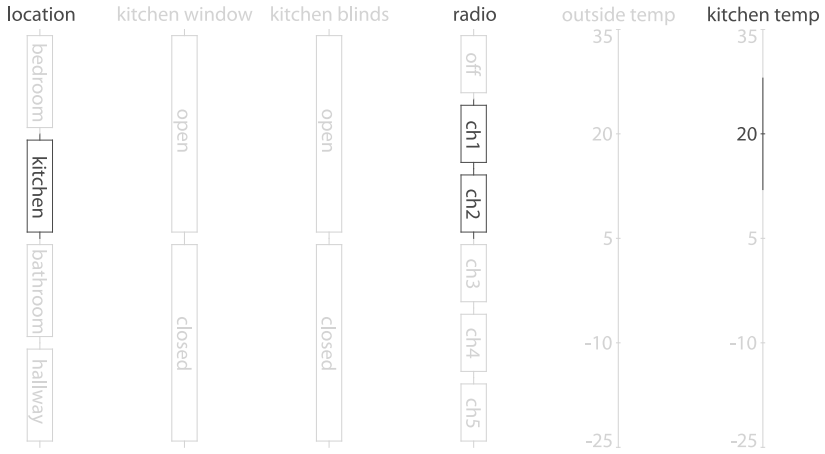


Figure 3.4: A context scope C^1

$C^1 = \{location : \{kitchen\}, radio : \{ch1, ch2\}, kitchen\_temp : [12, 28]\}$

3.5 Context-altering services

Context-altering services were previously introduced as services that have some context preconditions and effects. This section now discusses the relationship between services and user context in detail and defines context-altering services formally.

3.5.1 Service preconditions

The preconditions of a service are any context conditions that must be fulfilled before the service can be executed. Take as example a service s for opening the kitchen window. For a user performing this action manually it is obvious that the window can only be opened if it is currently closed. In order to allow the smart home to perform the same reasoning, this information must be made explicit.

For service s there could be additional preconditions besides the current status of the window. For example, there may be window blinds installed that, when closed, restrict opening the window. The window may also be furnished with a child-proof lock and only if unlocked can the window actually be opened. Of course, descriptions of service preconditions can only be useful if all relevant data is provided by some context source. If there is no context source for the status of the window blinds, then the smart home is not able to determine whether the window can actually be opened.

We model service preconditions using context scopes, i.e. the precondition forms a subspace in $\mathbb{H}$. For the window opening service s, one can express the condition that the window must be closed and the blinds must be open as $s^{pre} = \{kitchen\_window : \{closed\}, kitchen\_blinds : \{open\}\}$. Only if the current user context falls within the subspace defined by s^{pre}, then s is a possible option.

Figure 3.5 demonstrates visually this notion of c^u being contained in s^{pre}. The figure shows the example HAC, overlayed with the context scope s^{pre} and the current user context c^u. Only the two dimensions *kitchen_window* and *kitchen_blinds* that are part of s^{pre} are relevant when checking whether c^u is contained in s^{pre}. Here, $c^u[kitchen\_window] = closed$ and $c^u[kitchen\_blinds] = open$, so the precondition is fulfilled.

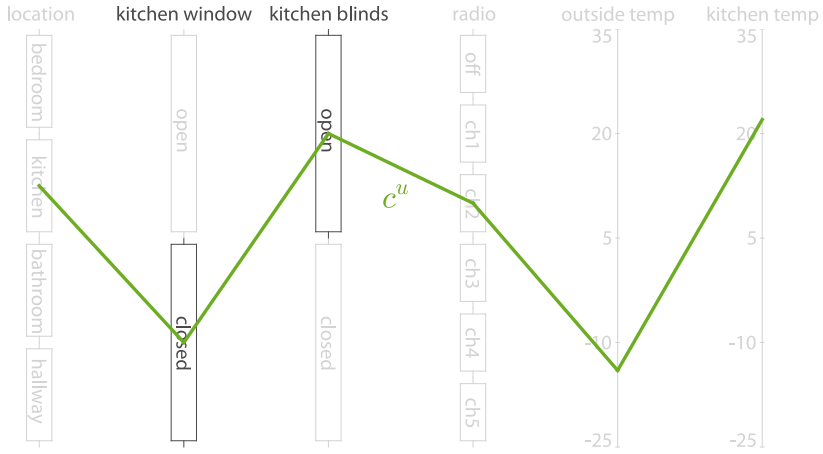


Figure 3.5: Service precondition s^{pre} is fulfilled by user context c^u

$s^{pre} = \{kitchen\_window : \{closed\}, kitchen\_blinds : \{open\}\}$

$c^u = \{location: kitchen, \mathbf{kitchen\_window: closed}, \mathbf{kitchen\_blinds: open}, radio: ch2, outside\_temp: -14, kitchen\_temp: 22\}$

In contrast, Figure 3.6 shows a user context that is not contained in s^{pre}. Again, only the *kitchen_window* and *kitchen_blinds* dimensions are relevant and in this case $c^u[kitchen\_blinds] = closed$, i.e. the precondition is not fulfilled. The operation for checking whether a context point is contained in a context scope will be formalized in Section 3.6.3.

3.5.2 Service effects

The execution of a context-altering service causes some changes in the user context. For the *open window* service, one obvious context change $\Delta c = \{kitchen\_window: open\}$ can be expected. Given that for the usual c^u, it is rather cold outside (*outdoor_temp*=-14), but warm inside (*kitchen_temp*=22), one might also expect that the indoor temperature decreases when opening the window. However, we do not know exactly, how cold it will actually be in the kitchen after opening the window and the temperature changes will happen gradually. The cooling down process is influenced by a large number of factors, such as the physical setup of the house and the type and placement of heating sources, and is thus very hard to model realistically. For this reason, we have instead opted for a more simplified model, which merely states that at some point the kitchen temperature will decrease.

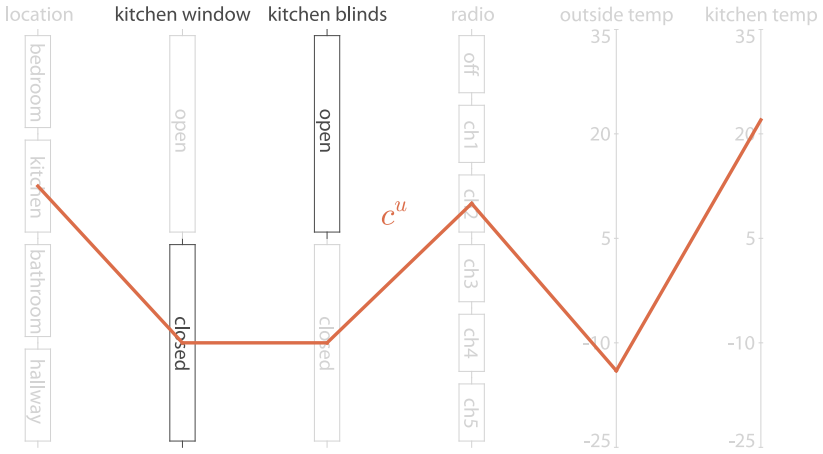


Figure 3.6: Service precondition s^{pre} is not fulfilled by user context c^u
$s^{pre} = \{kitchen\_window : \{closed\}, kitchen\_blinds : \{open\}\}$
$c^u = \{location: kitchen, \mathbf{kitchen\_window: closed}, \mathbf{kitchen\_blinds: closed},$
$radio: ch2, outside\_temp: -14, kitchen\_temp: 22\}$

Service effects can be formalized using a context scope, which allows to define some range of possible context changes. The effect s^{eff} of the *open window* service s might be described as context scope $s^{eff} = \{kitchen\_window : \{open\}, kitchen\_temp : [-25, 22]\}$. Figure 3.7 shows this s^{eff}, together with the user context c^u before execution of s and the changed context c'^u five minutes after the execution of s. In this example, after s was executed, the window is indeed open, and the kitchen temperature has decreased to 15°C. No context changes were recorded for any of the other dimensions, but of course unrelated context changes caused by some environmental change or another user actions could occur as well.

Service main and side effects

One critical issue with this model of s^{eff} remains – it is not possible to express that changes in kitchen temperature happen only under certain conditions. The kitchen temperature will decrease only if it is cold outside and if there are no other heating sources that could counteract the coldness. If, on the other hand, it is hot outside, then one will likely see an increase in temperature instead. For this reason it is necessary to distinguish between effects that happen every time a service is executed (called the main effects of a service) and effects that happen only if some additional conditions are fulfilled (the side effects of the service). For the *open window* service, there is one main effect $s^{eff} = \{kitchen\_window : \{open\}\}$, and several side effects s^{side} that state how the kitchen temperature changes under different circumstances.

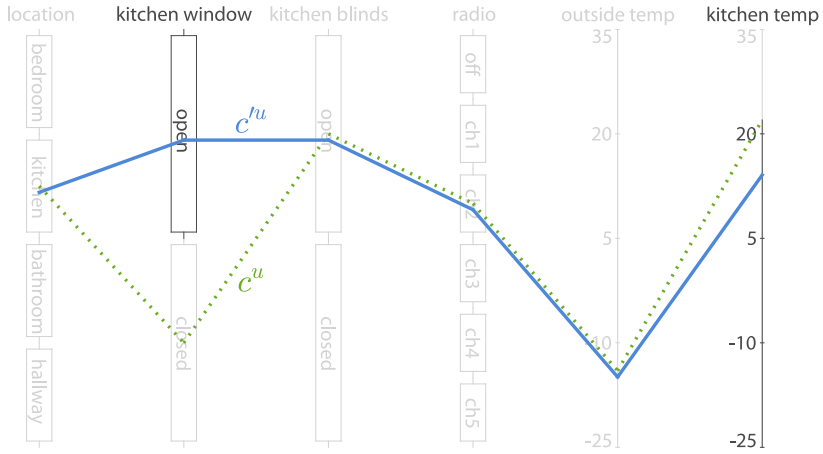


Figure 3.7: Service effect s^{eff}, c^u before execution of s (dotted line), and c'^u five minutes after execution of s (solid line)

$$s^{eff} = \{kitchen\_window : \{open\}, kitchen\_temp : [-25, 22]\}$$
$$c^u = \{location: kitchen, kitchen\_window: closed, kitchen\_blinds: open, radio: ch2, outside\_temp: -14, kitchen\_temp: 22\}$$
$$c'^u = \{location: kitchen, \textbf{\textit{kitchen\_window}}: \textbf{\textit{open}}, kitchen\_blinds: open, radio: ch2, outside\_temp: -14, \textbf{\textit{kitchen\_temp}}: \textbf{\textit{15}}\}$$

3.5.3 Formal definition of context-altering services

Following these observations, we can formally describe context-altering services according to Definition 6. Precondition, main effect and side effects are modeled as context scopes in $\mathbb{H}$. The precondition is used to check whether the service can be executed in the current user context c^u; the main effect describes how the service changes the c^u when s is executed. Side effects are defined as a set of additional precondition→effect pairs. A side effect only occurs, if its precondition is fulfilled by c^u. The overall effect of s is a combination of the main effect and all side effects that apply.

Definition 6 (Context-altering service) *A context-altering service is situated in $\mathbb{H}$ and is described with:*

- s^{pre}*, a context scope in $\mathbb{H}$ that represents the precondition of s; when c^u fulfills s^{pre}, then s is a possible execution choice.*
- s^{eff}*, a context scope in $\mathbb{H}$ that represents the main effect of s; executing s will change c^u according to s^{eff}.*
- $s^{side} = \{(s^{spre_0} \rightarrow s^{seff_0}), \ldots, (s^{spre_{m-1}} \rightarrow s^{seff_{m-1}})\}$*, the set of m side effects of s. Each pair $(s^{spre_i} \rightarrow s^{seff_i})$ for $0 \leq i < m$, describes one side effect of the service, where s^{spre_i} and s^{seff_i} are context scopes in $\mathbb{H}$. If c^u fulfills s^{spre_i}, then executing s will change c^u according to s^{eff} and s^{seff_i}.*

3.6 Operations in HAC

Since context points and context scopes are geometrical objects situated in the same multi-dimensional space, operations on these objects can now be easily defined. This section introduces four context operations in HAC that are important for filtering services and identifying the most beneficial services for a given situation. *Context description* is used from now on as an umbrella term for context points and context scopes.

3.6.1 Basis

The basis of a context description is a bit-array that identifies, which dimensions are relevant for the description, i.e. which dimensions are not set to $\perp$. An element in the bit-array $B(c)$ is set to 0 if the corresponding dimensions is $\perp$, otherwise the element is set to 1.

Table 3.8 demonstrates the basis operation for a context point c and a context scope C, both situated in the example six-dimensional $\mathbb{H}$. The basis $B(c)$ is an array of length six, where the first element corresponds to the first dimension in $\mathbb{H}$, the second element corresponds to the second dimension, etc. $B(c) = [0,1,0,0,0,1]$ expresses that the first dimension (*location*) is missing, the second dimension (*kitchen_window*) is set to some value, the following three dimensions (*kitchen_blinds*, *radio*, *outside_temp*) are missing, and the last dimension (*kitchen_temp*) is again set to some value. The basis of the context scope C is constructed analogously. Addressing the basis by index as well as addressing by dimension name are supported, examples are given in the table.

Dimensions of $\mathbb{H}$
$[\mathbb{D}_{location}, \mathbb{D}_{kitchen\_window}, \mathbb{D}_{kitchen\_blinds}, \mathbb{D}_{radio}, \mathbb{D}_{outside\_temp}, \mathbb{D}_{kitchen\_temp}]$
Basis of a context point
$c = \{kitchen\_window: closed, kitchen\_temp: 22\}$
$B(c) = [0,1,0,0,0,1]$
Basis of a context scope
$C = \{location : \{kitchen\}, radio : \{ch1, ch2\}, kitchen\_temp : [0,22]\}$
$B(C) = [1,0,0,1,0,1]$
Addressing individual basis elements
$B(c[0]) = B(c[location]) = 0$
$B(C[0]) = B(C[location]) = 1$

Table 3.8: Examples for the basis of context points and context scopes

The basis operation is formalized for context points in Definition 7 and for context scopes in Definition 8

Definition 7 (Basis of context point c) *Let c be a context point in an n-dimension HAC $\mathbb{H}$. Then the basis $B(c) = [b_0, \ldots, b_{n-1}]$ is a bit-array of length n with $\forall\, 0 \leq i < n :$ if $c[i] = \perp$ then $b_i \leftarrow 0$ else $b_i \leftarrow 1$.*

Definition 8 (Basis of context scope C) *Let C be a context scope in an n-dimension HAC $\mathbb{H}$. Then the basis $B(C) = [b_0, \ldots, b_{n-1}]$ is an bit-array of length n with $\forall\, 0 \leq i < n$: if $C[i] = \perp$ then $b_i \leftarrow 0$ else $b_i \leftarrow 1$.*

The basis can be used to very quickly determine whether two context descriptions share any dimensions. For the example c and C from Table 3.8, the result of a bitwise *and* operation on their bases $B(c)\&B(C) = [0,1,0,0,0,1]\&[1,0,0,1,0,1] = [0,0,0,0,0,1]$ reveals that the two context descriptions share only the last dimension. The basis can be converted into the decimal representation of the binary array, e.g. $\texttt{decimal}(010001) = 17$. Then two context descriptions share a dimension if $\texttt{decimal}(B(c)\&B(C)) > 0$. This mechanism will be used in Section 4 to rapidly filter service descriptions.

3.6.2 Projection

Let c^1 and c^2 be two context points in the example $\mathbb{H}$:

$c^1 = \{location\text{: } bathroom, kitchen\_window\text{: } closed, kitchen\_blinds\text{: } open\}$
with $B(c^1) = [1,1,1,0,0,0]$
$c^2 = \{location\text{: } kitchen, kitchen\_temp\text{: } 17\}$
with $B(c^2) = [1,0,0,0,0,1]$

The projection operation $c^1 \times B(c^2)$ removes from c^1 all dimensions that are not used in c^2, i.e. it sets the dimensions *kitchen_window* and *kitchen_blinds* to $\perp$. The result of $c^1 \times B(c^2)$ is a new context point $c'^1 = \{location\text{: } bathroom\}$ with $B(c'^1) = [1,0,0,0,0,0]$. The projection operation is needed as a preparation step for other context operations, e.g. when checking if the user context fulfills a service precondition; an example will be given at the end of the next subsection. The projection operation for a context point is defined formally in Definition 9.

Definition 9 (Projection $\times B$ for a context point) *Let $\mathbb{H}$ ben an n-dimensional HAC, c be a context point in this $\mathbb{H}$ and B the basis of a context point or context scope in $\mathbb{H}$. The result of $c \times B$ is a context point $c' = [v_0, \ldots, v_n]$ in $\mathbb{H}$ with $\forall\, 0 \leq i < n$: if $B[i] = 0$ then $c'[i] \leftarrow \perp$ else $c'[i] \leftarrow c[i]$.*

The projection operation can analogously be applied to a context scope by setting some dimensions to $\perp$, according to a reference basis B (Definition 10).

Definition 10 (Projection $\times B$ for a context scope) *Let C be a context scope in an n-dimensional HAC $\mathbb{H}$ and B the basis of a context point or context scope in $\mathbb{H}$. The result of $c \times B$ is a context scope $C' = [v_0, \ldots, v_n]$ in $\mathbb{H}$ with $\forall\, 0 \leq i < n$: if $B[i] = 0$ then $C'[i] \leftarrow \perp$ else $C'[i] \leftarrow C[i]$.*

3.6.3 Containment

The containment operation $c \in C$ tests, whether a context point c is contained in a context scope C, i.e. whether c lies within the subspace defined by C. This operation has been informally introduced in Section 3.5 as a method for checking if the user context fulfills the preconditions of a service. The containment operation tests for every dimension of $\mathbb{H}$, whether c lies within C on this dimension. For one dimension i, the following four cases must be considered:

$B(c[i])$	$B(C[i])$	$c[i]$ lies within $C[i]$?
0	0	Yes
0	1	No
1	0	No
1	1	If $c[i] \in C[i]$

If the bases of $c[i]$ and $C[i]$ are both 0, then dimension i is irrelevant in both context descriptions; in this case $c[i]$ lies within $C[i]$. If the bases differ, then i is contained in one of the context descriptions and irrelevant in the other, thus $[i]$ does not lie within $C[i]$. If both bases are 1, then i is relevant for both context descriptions and $c[i]$ lies within $C[i]$ if $c[i]$ is contained in the range defined by $C[i]$. Definition 11 expresses the containment operation in a formal manner.

Definition 11 ($c \in C$) *Let $\mathbb{H}$ be a HAC with n dimensions, c a context point in $\mathbb{H}$ and C a context scope in $\mathbb{H}$. Then* $c \in C \Leftrightarrow \forall\, 0 \leq i < n : \quad (B(c[i]) = 0 \wedge B(C[i]) = 0) \vee ((B(c[i]) = 1 \wedge B(C[i]) = 1 \wedge c[i] \in C[i])$.

Table 3.9 shows results of the containment operation for three examples.

Context scope to check against
$C = \{radio : \{ch1, ch2\}, kitchen\_temp : [19, 22]\}$
Example 1
$c^1 = \{location: kitchen, radio: ch1, kitchen\_temp : 19\}$
$c^1 \notin C$ (because $B(c^1[location]) \neq B(C[location])$)
Example 2
$c^2 = \{radio: ch1, kitchen\_temp : 17\}$
$c^2 \notin C$ (because $c^2[kitchen\_temp] \notin C[kitchen\_temp]$)
Example 3
$c^3 = \{radio: ch1, kitchen\_temp : 19\}$
$c^3 \in C$

Table 3.9: Examples for containment operation

Using projection and containment to check if service precondition is fulfilled

Figure 3.5 in Section 3.5 showed a user context c^u that fulfills the *open window* service precondition. Following c^u and s^{pre} were used in the figure:

$$s^{pre} = \{kitchen\_window : \{closed\}, kitchen\_blinds : \{open\}\}$$
$$c^u = \{location: kitchen, kitchen\_window: closed, kitchen\_blinds: open,$$
$$radio: ch2, outside\_temp: -14, kitchen\_temp: 22\}$$

When applying the containment operation directly on c^u and s^{pre}, the result is that $c^u \notin s^{pre}$. The reason is that there are four dimensions for which the bases of c^u and s^{eff} differ. When checking if the service precondition is fulfilled, only those dimensions that have a basis 1 in s^{pre} should be considered. The projection operation $c^u \times B(s^{pre})$ eliminates the unnecessary dimensions from c^u. The result of the operation is a new context point c'^u with only two dimensions, $c'^u = \{kitchen\_window: closed, kitchen\_blinds: open\}$. For this new context point, the containment operation is successful, $c'^u \in s^{pre}$.

3.6.4 Scope combination

The overall effect of a service is a combination of the main effect and all side effects that apply. This combined effect can be calculated using the scope combination operator $\biguplus$. For example, the kitchen window service s might have $s^{eff} = \{kitchen\_window : \{open\}\}$ and one side effect $s^{seff_0} = \{kitchen\_temp : [-25, 22]\}$ that applies in the current user context. Then the combined effect of this service s^{comb_eff} can be calculated using $s^{comb\_eff} = s^{eff} \biguplus s^{seff_0} = \{kitchen\_window : \{open\}, kitchen\_temp : [-25, 22]\}$.

For combining two scopes C^1 and C^2 on one dimension i, following four cases have to be considered:

$B(C^1[i])$	$B(C^2[i])$	$C^1[i] \biguplus C^2[i]$
0	0	$\perp$
0	1	$C^2[i]$
1	0	$C^1[i]$
1	1	Nominal dimension: $C^1[i] \cup C^2[i]$ Numeric dimension: $[\texttt{min}(C^1[i].min, C^2[i].min), \texttt{max}(C^1[i].max, C^2[i].max)]$

Definition 12 declares how $C^1 \biguplus C^2$ can be calculated by combining the scopes on each context dimension.

Definition 12 ($C^1 \biguplus C^2$) *Let $\mathbb{H}$ be a HAC with n dimensions and C^1 and C^2 context scopes in $\mathbb{H}$. The result of the operation $C' = C^1 \biguplus C^2$ is a context scope C' with $\forall\ 0 \leq i < n$: $C'[i] = C^1[i] \biguplus C^2[i]$.*

Algorithm 1 calculates the overall service effect for a service s in user context c^u, by combining the main effect of the service with any applicable side effect. The combined effect s^{comb_eff} is first set to the main effect of s. Then it is checked for every side effect, whether the side effect condition is fulfilled in c^u. If this is the case, then s^{comb_eff} is merged with the side effect using the combine operator $\biguplus$.

Algorithm 1 Calculate the overall service effect

```
1: procedure COMBINEEFFECTS(c^u, s)
2:     s^comb_eff = s^eff
3:     for ∀(s^spre_i → s^seff_i) ∈ s^side do
4:         if (c^u × B(s^spre_i)) ∈ s^spre_i then
5:             s^comb_eff ← s^comb_eff ⊎ s^seff_i
6:         end if
7:     end for
8:     return s^comb_eff
9: end procedure
```

3.7 Smart home datasets used for evaluation

The smart assistants are evaluated using several datasets that were collected in prototype smart home installations.

3.7.1 van Kasteren houseA and houseB

The houseA and houseB datasets are made available by van Kasteren et al. [106, 105]. These datasets were originally recorded for evaluating activity recognition algorithms. The houseA dataset contains observations of the inhabitant of a smart home over a period of 25 days. 14 binary sensors detect the open/close status of front door, kitchen cabinets, bedroom door, refrigerator, freezer, dishwasher and washing machine, plus whether the toilet flush is on or off.

The houseB dataset was recorded over the course of 28 days (the house was only inhabited for 13 of these days), in a different smart home. In houseB, 23 binary sensors are installed: pressure mats for the bed and and a chair; open/close status of doors, window; sensors indicating the usage of drawers, toaster, microwave, stove, toilet and sink; infrared occupancy sensors in bedroom, kitchen and bathroom.

3.7.2 Domus

The domus dataset has recently been made available by the MultiCom research group in Grenoble, France [42]. The dataset contains recordings of nearly 1 year of smart home activity. In contrast to the houseA and houseB datasets, the house was not inhabited by one single person during this time. Instead, several experiments were carried out in the house

and it appears that the house was not consistently occupied. Nevertheless, the dataset is a tremendous resource, with data from numerous sensors and actuators.

Overall, more than 100 devices are installed in the home. From the documentation it is not always evident, whether a device is an actuator or a sensor; for example it is not clear if window blinds are motorized and publish their internal state, or if open/close status information is provided by a sensor. Data is coming from lamps, windows and doors, window blinds and shutters, power outlets and presence detectors. Additionally, there is data about water and electricity usage, indoor temperature and humidity in different rooms, and the outside weather (temperature, air pressure, humidity, etc). Most devices produce binary data, however there are also some devices with more than two states (e.g. shutters have four different opening states). Additionally, several sensor produce numeric data, e.g. temperature, humidity, water usage.

3.7.3 SM4All smart home

We also had access to a smart home prototype, owned by Fondazione Santa Lucia[4] and situated in Rome, Italy. Most of the installed devices were fitted to the home as part of the demonstration of the middleware developed in the SM4All (Smart homes for all) project[5]. The SM4All home contains 13 actuators: bed, curtains, window, front door, lamps, alarm, stereo, TV. Each of the actuators publishes its internal status (all internal states are binary: on/off, open/closed, etc). Additional data is provided by an indoor location system, a smoke sensor and four sensors in the refrigerator that indicate whether food items are placed at the respective positions in the refrigerator.

We do not have a dataset with recordings of sensor and actuator data available for the SM4All home. Instead we had the opportunity to test the installation assistant directly in the prototype home.

3.7.4 Converting the datasets into HAC

The datasets contain the smart home data in form of a sequence of events published by different context sources. Each event is given as *(timestamp,name,value)*, e.g. *(25-Feb-2008 00:20:14,bedroom door, open)*. Formally, the dataset can be represented as sequence of $k+1$ events $[(ts_0, name_0, value_0), \ldots, (ts_k, name_k, value_k)]$. This section describes how these datasets can be transformed into a HAC-based dataset, so that that they can later be used in the evaluations.

Create the HAC

In the first step, each different *name* that occurs in the event list is mapped to a context dimension. For example, we may have seen 20 events for *bedroom door*, of which 10 events signal that the door was opened and 10 events signal that the door was closed. Based on this information, a context dimension $\mathbb{D}_{bedroom\_door} = \{open, closed\}$ can be declared.

[4] `http://www.hsantalucia.it/`

[5] SM4All was funded by the 7th Framework program of the European Union, FP7 STREP Grant No. 224332

The general process for creating a context dimension for *name* is to *(i)* find all events for *name*, *(ii)* extract from the events all different values that were ever reported for *name*, *(iii)* decide if *name* is a numeric or nominal dimension and create the dimension accordingly.

Most of this process can be done automatically by a script. Manual intervention can be necessary when deciding whether the dimension should be numeric or nominal. This is because often context sources report nominal data using discrete numeric values. For example, in the domus dataset the four settings of a window blind are reported as $0.0, 1.0, 2.0, 3.0$. For such cases, mapping instructions must be provided manually. Output of this step is a HAC $\mathbb{H}$ with n dimensions.

Extract user contexts and context changes

No information is known about values of the context sources before the first event, e.g. it is not known whether the bedroom door initially is open or closed. The user context before the first event is thus empty, all dimensions are set to $\perp$. After the first event $(ts_0, name_0, value_0)$ it is known that $name_0 = value_0$ and the initial user context can be calculated:

Initial user context after event $(\mathbf{ts_0}, \mathbf{name_0}, \mathbf{value_0})$
$c_0^u = \{name_0\text{: } value_0\}$
initialize elapsed time vector as $\Delta t_0^u = [-1, \ldots, -1]$ and set $\Delta t_0^u[name_0] = 0$

Each subsequent event is interpreted as a context change and the user context is updated:

After event $(\mathbf{ts_i}, \mathbf{name_i}, \mathbf{value_i}), \mathbf{1 \leq i \leq k}$
Context change $\Delta c_i = \{name_i\text{: } value_i\}$
is applied to previous context to obtain new context $c_i^u = c_{i-1}^u \times \Delta c_i$
and the elapsed time vector is updated:
$\Delta t_i^u[name_i] = 0$,
for other dimensions x with $c_{i-1}^u[x] \neq \perp$: $\Delta t_i^u[x] = \Delta t_{i-1}^u[x] + (ts_i - ts_{i-1})$

Output of this step is a sequence of $k+1$ user contexts $[c_0^u, \ldots, c_k^u]$, a sequence of $k+1$ elapsed time vectors $[\Delta t_0^u, \ldots, \Delta t_k^u]$ and a sequence of k context changes $[\Delta c_1, \ldots, \Delta c_k]$.

Identify context-altering services

The datasets do not contain any information about context-altering services and their preconditions and effects. In fact, we don't even always know if devices are actuators or sensors. Therefore, we apply following rules:

- Windows, doors, drawers, lamps, blinds, shutters, curtains, refrigerator, freezer, microwave, dishwasher, washing machine, motorized bed, toilet flush: all of these devices could potentially be remote controlled, if an actuator was installed. We describe context-altering services for these devices.
- All other data is considered to be coming from sensors and we do not describe any context-altering services.

Since most actuators produce binary data, we typically define two services. For example, for a *lamp* that publishes status updates with values *on* and *off* we describe a service $s = switch\_on\_lamp$ with $s^{pre} = \{lamp : \{off\}\}$ and $s^{eff} = \{lamp : \{on\}\}$ and vice versa for service *switch_off_lamp*. We also have to look at other context dimensions such as *luminosity* to check whether this service has any side effects. In some cases, an actuator may only have one service. For example, an actuator *toilet_flush* will only have one service *activate flush*, since deactivation happens automatically.

Build the dataset

The result of the conversion process is a HAC-based smart home dataset as summarized in Table 3.10. There is some redundancy for the sequence of user contexts and the sequence of context changes, since each c_i^u can be calculated using $c^u = c_{i-1}^u \times \Delta c_i$, for $1 \leq i \leq k$. Both sequences are included in the dataset since it simplifies some of the evaluations.

n	number of context dimensions
$\mathbb{H}$	n-dimensional HAC
$[c_0^u, \ldots, c_k^u]$	sequence of $k+1$ user contexts
$[\Delta t_0^u, \ldots, \Delta t_k^u]$	sequence of $k+1$ elapsed time vectors
$[\Delta c_1, \ldots, \Delta c_k]$	sequence of k context changes
S	set of context-altering services

Table 3.10: HAC-based smart home dataset

Datasets after conversion to HAC

A summary of the datasets after the conversion to HAC is given in Table 3.11.

	houseA	**houseB**	**domus**	**SM4All**
Number of context dimensions n	14	22	105	16
Number of user contexts $k+1$	≈2400	≈38000	$>$ 3 million	-
Number of services $\|S\|$	27	26	53	26

Table 3.11: Datasets after conversion to HAC

3.8 Summary

This chapter presented a context model for the smart home. The context model is grounded on observations of the diversity of context sources in smart homes and the relationship between user context and context-altering services. The proposed context model

- forms a multi-dimensional context space $\mathbb{H}$ that supports numeric and nominal context data,
- expresses the user context c^u and context changes Δc as vectors in the context space,
- allows to define subspaces in the context space,
- and uses these subspaces to express service precondition, main effect and side effects.

Since all context descriptions are geometrical objects that are situated in the multi-dimensional space, a number of operations on context descriptions could be defined:

- The context change operation $\times\Delta c$ updates the user context to reflect context changes.
- The basis B marks which dimensions are relevant for a context description and can be used to quickly check if two context descriptions share some dimensions.
- The projection operation $\times B$ transforms a context description according to a reference basis B.
- The containment $\in$ operation allows to check whether a context point is contained in a context scope.
- The combine operation $\uplus$ combines two context scopes.

We also present the datasets we use for evaluating the smart assistants and describe how they can be converted into the HAC format. With the proposed context model we now have the necessary tools for developing a context-aware, learning home.

Chapter 4

A recommender system for smart homes

In Chapter 2 we have seen two opposing strategies of smart home control. The first strategy uses pre-defined rules and scenes that automate frequently needed functionality. After initial configuration, the user hands over control to the system. The aim is to increase user convenience, but in reality the strategy cannot cope with the variability of user routines. The second strategy gives the user free reign over the smart home through an interface that allows to manually select which service to execute. This strategy puts the user in full control of the smart home, but introduces so much overhead that many users would prefer to disregard the smart home and perform the desired actions manually.

With the material presented in this chapter and the following two chapters we want to address these issues. We propose a recommender system that interprets the user's current context and generates personalized service recommendations. The system tries to recommend services that would be beneficial for the user in some way, i.e. can automate some action that the user would want perform anyway. We present several methods for utilizing the recommendations to increase the usability of the smart home.

In this chapter

This chapter starts by demonstrating the recommender system in an example scenario. We discuss different ways of utilizing the recommendations and identify performance requirements for the system from a usability perspective. We then move on to a more technical presentation of the recommender system. After an overview of two phases of the system, the first phase is presented in detail at the end of this chapter. The second phase is subject of the two following chapters.

4.1 Example scenario

Figure 4.1 introduces the smart home recommender system with help of an example scenario. In the initial situation, the user is reading in the living room, there is good lighting, a pleasant temperature and there are no issues reported by the smart home that need user attention. Some services are more likely to be relevant in this situation than others, in particular

services offered by actuators in spatial proximity to the user. However, since the user is comfortable, the smart home system cannot make any specific recommendations.

If the situation changes, previously calculated recommendations may no longer be relevant. This means that **whenever the user context changes, the recommendations must be re-computed**. The first context change that triggers the recommendation process is an increase in the indoor temperature by 0.1°C. The recommendation algorithm is run on the updated user context, but since the change is minimal, still no specific recommendations are necessary.

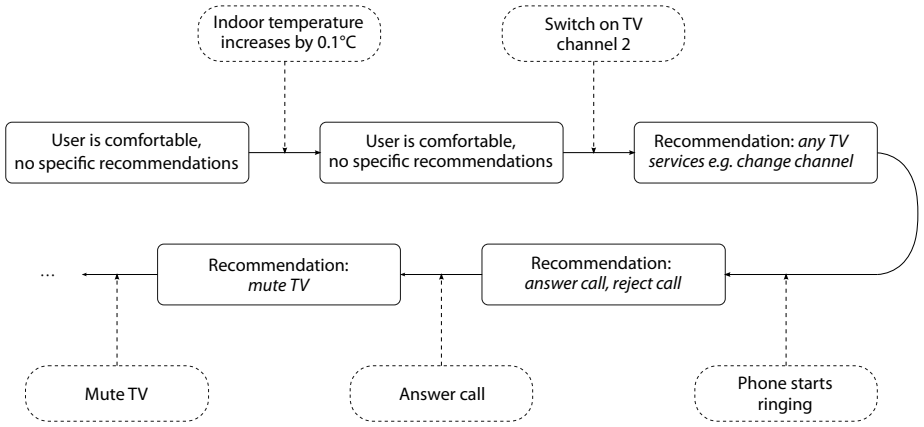


Figure 4.1: Context changes trigger the computation of service recommendations
Initial situation: user is reading in the living room; lighting, temperature and all other smart home settings are to user's liking

Next, the user turns on the TV and several context changes happen, e.g. the TV status is set to *on* and the sound volume in the room increases. The recommender system recognizes that now any services that can control the TV, such as changing the channel and increasing or decreasing the volume, could likely be needed by the user.

After a while, a context change signals that the phone has started ringing. The recommender system knows that the user likes to immediately give attention to a ringing phone, and therefore recommends two specific services *answer call* and *reject call*. When the user answers the phone, the recommender system recognizes that the updated user context contains a conflict (TV sound is on while the user is talking on the phone) and advises to mute the TV.

The example scenario shows the wide variety in possible recommendations. Sometimes no recommendations can be made, the system may only be able to give some general information about which services are more likely. In other situations the recommender system can produce very specific recommendations.

4.2 Utilizing the recommendation results

Today's smart home users complain that the process of selecting services from the user interface (UI) introduces so much overhead that it is easier to just execute the desired action manually. Additionally, the UIs are often unsuitable for guests that are unfamiliar with the system. Inhabitants today have full control over their home, but the UI offers a low level of usability and convenience. We present in the following several strategies for utilizing the recommendation results and discuss their impact on the issues of control loss and convenience.

4.2.1 Active context-awareness

Chen and Kotz [18] make the distinction between active and passive context-awareness. An active context-aware application "automatically adapts to discovered context, by changing the application's behavior" [18, p. 3]. An active context-aware smart home continuously senses the user context, interprets it using the recommender system and autonomously acts on the recommendation results by executing the best recommendation. Active context-awareness requires that very precise service recommendations are generated. The smart home must be able to decide *(a) if a service should be executed in the current situation*, *(b) which service should be executed* and also *(c) exactly when to execute it*.

Pinpointing the best time of execution may be even more difficult than identifying the correct service to execute. For example, imagine that the smart home inhabitant has just opened the refrigerator to take out some breakfast ingredients. It is not very hard to predict that one of the inhabitant's next actions will be to close the refrigerator and hence to identify the corresponding service *close refrigerator* as the best service recommendation. However, the home should avoid closing the refrigerator before the inhabitant is finished with retrieving the needed groceries. Even if the chosen execution time is based on additional data (e.g. the user is not near the refrigerator anymore), the user may get annoyed if the refrigerator closes prematurely (actually the user was just transporting some ingredients to the stove and planned to return to the refrigerator).

There is a risk that active context-awareness elicits feelings of control loss even if recommendations and timings are perfect. With imprecise recommendations and timings, this loss of control may no longer be compensated by an increased level of convenience. The inhabitant could quickly become frustrated and switch off the system.

This dissertation mainly address issue *(b)* by developing algorithms for computing the service recommendations. Chapter 5 also investigates some first ideas for issues *(a)* and *(c)*. Generally, we feel that active context-awareness is not yet not a viable option for the smart home. Currently this strategy should only be used in a few selected situations, such as ensuring a comfortable indoor temperature.

4.2.2 Passive context-awareness

Instead, we are turning towards what Chen and Kotz call passive context-awareness [18]. With this strategy the smart home also continuously senses the user context and interprets it

using the recommender system. However, the home does not autonomously execute some service, but instead waits for input from the user.

To achieve passive context-awareness, the service recommendations are used to build a very simple graphical user interface (GUI) that shows only the currently most relevant choices. In case the user wants to perform some other, less typical action, a list of all services may be hidden behind an additional button. This solution increases convenience but keeps the user still fully in control.

More natural interaction

Even with a simplified graphical user interface (GUI), standard touch-screen operated devices such as tablets or smart phones will likely involve too much overhead for most users. In many cases it will be much simpler to perform the desired action manually than to retrieve the device from the pocket or wherever it was left previously, switch it on, and select the correct service. Such an input device may be most interesting for users that have trouble performing some tasks manually, e.g. opening heavy drawers. It may also be useful in comfort situations, e.g. for controlling the home while resting on the couch.

We think that wearable computing solutions would be better suited as input devices for smart home environments. Wearable devices such as head-mounted displays and computerized wristwatches are becoming commercially available now and could remove much of the current overhead when interacting with the smart home. These devices are readily accessible whenever the user wants to send commands to the system. Since display space is often reduced in such devices, it becomes even more important to reduce the selection choice to only those services that are currently relevant for the user. Head and hand gestures may be used to scroll through the service list and select a service for execution.

Speech recognition allows for an even more natural way of interacting with the smart home. A context-aware speech recognition system might use service recommendations to improve recognition rates. Whenever recognition confidence is low and there are several potential service matches for the user's command, the system can cross-check with the recommendations to decide which of the options is most probable given the user's situation.

4.2.3 Mixed strategy

If there are several alternative services that all fulfill some similar goal, a GUI might show only the most recommended of these services and indicate that there are alternatives. One might even combine passive context-awareness with some aspects of active context-awareness. Instead of letting the user select from the alternatives, the GUI presents a summary of the choices and the smart home chooses which alternative to execute.

For example, the smart home senses that it is too dark in the room and recommends that one or several lamps should be switched on. Instead of listing specific recommendations, the GUI simply displays a virtual light switch. When the user selects this option, the smart home decides whether to turn on the lamp next to the couch (since the user is watching TV), or the one above the dining table (since dinner is being prepared in the kitchen). This mixed strategy avoids the timing issues of active context-awareness, since the cue for executing a

service comes from the user. Comfort is increased, but there is also some loss of control (because the user can not be sure which light will be turned on).

The mixed strategy also lends itself very well for the use with speech and gesture recognition systems. Instead of having to specify exactly which action should be executed, the user can give imprecise commands, e.g. *"light!"*. Further details are not necessary, the smart home can use the service recommendations to autonomously choose which lamp to switch on.

4.3 Performance requirements

The recommendations must at all times reflect the current context of the user. Updated recommendations must be made available nearly instantly after a context change, so that the user does not notice any delays. For example, assume that the recommendations are displayed in a GUI. When the user selects the recommended service *mute TV*, a context change is caused, which in turn triggers a new round of the recommendation process. The new recommendations will not contain the service *mute TV*, since the TV is already muted. If the recommendation process is slow, then the user will continue to see the now useless service *mute TV*, which will make the GUI appear sluggish.

Delays in the recommendation process

Figure 4.2 shows, how delays may happen at every step of the recommendation process. The only delay that we can influence is the time needed to calculate the new recommendations. The other network and computational delays depend on the technologies used in the smart home, e.g. communication protocol and system architecture.

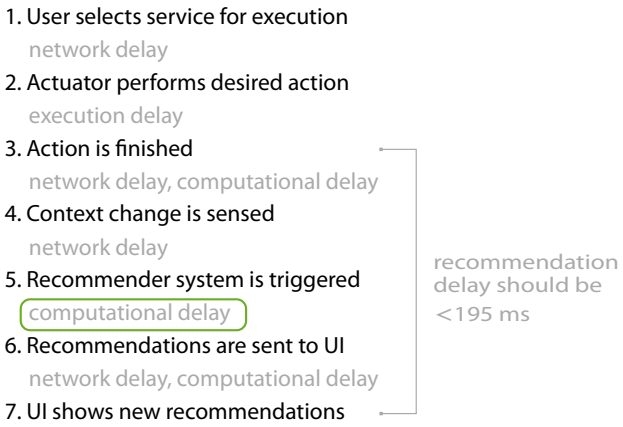


Figure 4.2: Possible delays in the recommendation process
(only the delay marked with a box is under our control)

Another delay that is outside of our control is the *execution delay* of the actuators. The execution delay encompasses the time between the start of an actuator action (e.g. starting to open the window) and the finish of the action (window is fully open). The execution delay can vary considerably between the actuators. In the SM4All smart home testbed we measured execution delays between 30 milliseconds (turning on a lamp) and 21 seconds (raising the motorized bed into a sitting position). The users can in many cases observe the progress of the executed action and can therefore anticipate the execution delay. As soon as the action is finished, however, the UI should reflect the updated situation. We hence define the recommendation delay as the time between the end of the actuator action and the update of the UI.

Targeted maximum delay

Dabrowsky and Munson performed an experiment to find out at which point computer users notice delays in a GUI [24]. The 21 test subjects performed several tasks in a test application using keyboard and mouse. During the experiment, the reactions of the applications were increasingly delayed and the subjects were asked if they noticed the delay. Most similar to our use case are the *button-bar* and the *dialog-box* tasks, where users clicked on a button and waited for the response of the system. Dabrowsky and Munson found that the subjects did not notice delays of less than 195 milliseconds for these tasks. We have not found any similar evaluations for other GUI interaction modes, such as touch-screens. Based on these results, we are targeting a maximum recommendation delay of 195 milliseconds.

4.4 Overview of the recommender system

We can now start presenting the proposed recommender system in more technical detail. Figure 4.3 gives an overview of the two phases of the system.

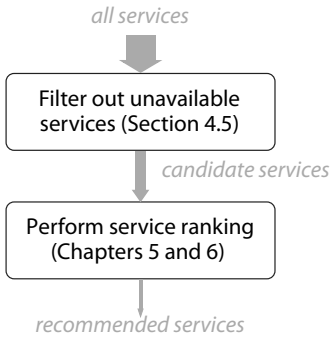


Figure 4.3: The two phases of the recommender system

Phase 1: Filter out unavailable services

A service is a valid recommendation only if its preconditions are fulfilled by the current user context. We say that a service with fulfilled preconditions is *available* in the current user context, otherwise it is *unavailable*. For example, the *switch to channel 5* service is only available, if the TV is actually switched on and not already set to channel 5. If *switch to channel 5* is not available, then it should not be shown to the user as a recommended service. The first phase of the recommendation process is therefore to filter out unavailable services from the list of all services. The filtering process will be covered in detail in Section 4.5.

Phase 2: Perform service ranking

The result of the filtering step is an unordered set of candidate services. In the second phase, these services are ranked according to how beneficial they are to the user in the current situation. The highest ranked services form the service recommendations. Two different approaches for service ranking are proposed in this dissertation, each of which will be detailed in a dedicated chapter:

(i) Ranking based on user habits The first method focuses on *user habits*. In a training phase, the proposed algorithm learns, which user actions commonly occur in a given situation and which actions are related to each other. During service ranking, this knowledge is used to predict which actions are likely given the user's current situation and recent actions. A service is beneficial and will receive a high ranking if it can automate one of the predicted actions. The method is described in detail in Chapter 5.

(ii) Ranking based on user preferences The second approach focuses on *preferred situations*. A user preference is some context situation that is favored by the user. Such preferences can, for example, be used to express comfortable environmental settings, such as temperature and lighting. A beneficial service is one that can bring the user closer to a preferred situation. This method is described in detail in Chapter 6.

Both ranking mechanisms address slightly different needs. Ranking based on user habits models the dynamics of home life – support inhabitants in performing activities and using devices. Ranking based on user preferences focuses on static and background aspects of the home – allow inhabitants to perform these activities under comfortable conditions. The integration of the two mechanisms to create one overall service ranking is not addressed in this dissertation. This issue will be revisited in the discussion of future work in Chapter 9.

4.5 Filtering unavailable services

The remainder of this chapter deals with the first phase of the recommender system, i.e. how to filter out services that are unavailable in the current user context. We start by presenting a straightforward filtering algorithm that simply checks the availability of each service given the current user context. We then argue how the performance of this algorithm can

be improved by reusing previous filtering results and propose an updated algorithm. Both algorithms are evaluated and compared with each other.

Acknowledgements Fei Li had the initial idea for proactive service filtering based on current user context and proposed the initial algorithm. Further development of the algorithm was done in collaboration between Fei Li and me.

4.5.1 An algorithm for service filtering

Algorithm 2 takes as input the current user context c^u and the set of all services S. The set of candidate services S_{cand} is initialized as an empty set. For every service $s \in S$ it is then checked, whether the user context fulfills the service preconditions. In Line 4, first the user context is projected into the subspace of s^{pre}, i.e. $c^u \times B(s^{pre})$. Then the containment operator $\in$ tests whether the projected c^u is contained in the context scope defined by s^{pre}. If this is the case, then the preconditions of the service are fulfilled and the service is added to the set of candidate services. The algorithm returns the unordered set of candidate services S_{cand}.

Algorithm 2 Filter out unavailable services

```
1: procedure FILTERSERVICES(c^u,S,)
2:     S_cand = ∅
3:     for ∀s ∈ S do
4:         if (c^u × B(s^pre)) ∈ s^pre then
5:             S_cand ← S_cand ∪ {s}
6:         end if
7:     end for
8:     return S_cand
9: end procedure
```

4.5.2 Reuse of previous filtering results

Context changes affect only a minority of services, i.e. most services that were available in one user context will also be available after the context changed. This is because service preconditions are dependent only on a small number of dimensions and only if one of these dimensions changes, then the availability of the service may change. For example, the *switch to channel 5* service is dependent on only a few dimensions that describe the TV status. A context change on any other dimension does not affect the availability of the service in any way. For three smart home datasets we found that on average less than 10% of the services are affected by a context change (see Section 4.6).

We can take advantage of this fact by reusing most of the previous candidate set when calculating the updated candidate set. Instead of checking for every service if preconditions are fulfilled, it is instead tested if a service is affected by the context change, i.e. if service

precondition and context change have at least one dimension in common. Only a quick comparison of the bases of service precondition and context change is needed to find out if a service is affected. Then the availability check only has to be performed for the affected services. This new strategy can substantially decrease the runtime of the service filtering step, in particular in large smart homes with many context dimensions and services (see evaluations in Section 4.6). Since less time is needed for service filtering, more of the 195 milliseconds maximum delay remain for finding the best service ranking.

Formally, a service s is affected by a context change Δc if $\texttt{decimal}(B(\Delta c)\&B(s^{pre})) > 0$, i.e. the precondition of s has at least one dimension in common with Δc. An available service does not necessarily become unavailable if it is affected by a context change, e.g. after a minor change on a numeric dimension the service precondition s^{pre} might still be fulfilled. Analogously, an unavailable service does not necessarily become available if it is affected by a context change. For this reason it is necessary to check for every affected service whether the service preconditions are fulfilled by the new user context.

4.5.3 Updated algorithm with reuse of previous results

Algorithm 3 implements the filtering of unavailable services with reuse of previous results. The algorithm takes as input the current user context c^u, the context change Δc that lead to c^u, the set of all services S, and the previous candidate set S_{prev}. As the first step, S_{cand} is initialized as an empty set.

Algorithm 3 Filter out unavailable services with reuse of previous filtering results

```
1:  procedure FILTERSERVICESWITHREUSE( c^u, Δc,S, S_prev)
2:      S_cand = ∅
3:      for ∀s ∈ S_prev do                ▷ check which services from S_prev are still available
4:          if decimal(B(Δc)&B(s^pre)) = 0 or (c^u × B(s^pre)) ∈ s^pre then
5:              S_cand ← S_cand ∪ {s}
6:          end if
7:      end for
8:      for ∀s ∈ S\S_prev do              ▷ check which previously unavailable services are available
9:          if decimal(B(Δc)&B(s^pre)) > 0 and (c^u × B(s^pre)) ∈ s^pre then
10:             S_cand ← S_cand ∪ {s}
11:         end if
12:     end for
13:     return S_cand
14: end procedure
```

In lines 3-7 the algorithm looks at the services in the previous candidate set. The expression $\texttt{decimal}(B(\Delta c)\&B(s^{pre}))$ performs a bit-wise *and* operation on the bases of Δc and s^{pre} and converts the resulting bit array into its decimal representation. A result 0 signifies that the context change and the service precondition have no dimension in common, i.e the service is not affected by the context change and can directly be added to the new candidate set. If the result of the bit-wise and is not 0, then context change and

service precondition have at least one dimension in common, and it must be checked if the precondition is still fulfilled by the updated context.

In lines 8-12 the algorithm looks at those services that were previously not in the candidate set, but may now be available after the context change. For every service not in S_{prev}, line 9 tests, whether the service could have been affected by the context change. If the basis comparison shows that Δc and s^{pre} have no dimension in common, s was not affected and is still not available in the updated context. Otherwise, it must be evaluated, whether the updated user context fulfills the service precondition; if this is the case s is added to the candidate set. The algorithm returns the set of candidate services.

4.5.4 Runtime analysis

In the following, let $T_{fulfills}$ be the runtime of one application of operation $(c^u \times B(s^{pre})) \in s^{pre}$ that checks whether c^u fulfills the preconditions of s. Let T_{basis} be the runtime of one application of the basis comparison $\texttt{decimal}(B(\Delta c)\&B(s^{pre})) = 0$ that checks if Δc and s^{pre} have dimensions in common. Also, $|S|$ is the total number of services and $|S_{affected}|$ is the number of affected services, $|S_{affected}| \leq |S|$.

The runtime $r_{without\_reuse}$ for Algorithm 2 for service filtering without reuse of previous results is linear with respect to $|S|$:

$$r_{without\_reuse} = |S| * T_{fulfills} \tag{4.1}$$

The runtime r_{with_reuse} of Algorithm 3 for service filtering without reuse of previous results is also linear with respect to $|S|$:

$$r_{with\_reuse} = |S| * T_{basis} + |S_{affected}| * T_{fulfills} \tag{4.2}$$

$$(\text{Upper bound for } |S_{affected}| = |S| \rightarrow r_{with\_reuse} \leq |S| * T_{basis} + |S| * T_{fulfills})$$

The next section shows that on average $S_{affected} < 0.1 * |S|$ and $r_{with\_reuse} < r_{without\_reuse}$.

4.6 Evaluation of service filtering

This section reports results of the performance evaluation we performed with both filtering algorithms. Smart home datasets as well as synthetic datasets are used to evaluate filtering runtimes and the scalability of the algorithms.

4.6.1 Experimental setup

Smart home datasets

The houseA, houseB and domus smart home datasets are used in the experiments. From these datasets, only the set of services S, the initial user context c_0^u, and the sequence of context changes $[\Delta c_1, \ldots \Delta c_k]$ are needed.

Synthetically generated datasets

To evaluate how the filtering algorithms scale in larger installations, we implemented a generator for synthetic datasets. The generated datasets are modeled after the smart home data. Context changes are generated such that only one dimension changes at a time and services have preconditions that depend only on one or two dimensions. The generator takes as input the desired number of context dimensions, the number of services and the average ratio of affected services. It produces S, c_0^u and the context changes accordingly.

4.6.2 Inspecting the rate of affected services in smart home datasets

Reuse of previous filtering results is most effective if a substantial amount of services are not affected by the changing context. The aim of the first experiment is to get an idea of how many services are on average affected by context changes in the test smart homes. For each change Δc_i we count the number of services s with $\texttt{decimal}(B(\Delta c_i)\&B(s^{pre})) > 0$. This number is averaged and the 90% confidence interval is calculated. The results are shown in Table 4.1.

dataset	number of affected services	
houseA	1.88 ± 0.02	(of 27 services, $\sim 7\%$)
houseB	0.16 ± 0.02	(of 26 services, $\sim 0.6\%$)
domus	0.18 ± 0.003	(of 53 services, $\sim 0.3\%$

Table 4.1: Average number of affected services for three smart home datasets

In the houseA dataset, every context dimension except *ToiletFlush* can be mapped to two binary services. For example, the front-door actuator offers the two services *open door* and *close door*. When the status of the front-door changes, then the availability of these two services flips. None of the other services are affected by the context change. For this dataset there are therefore on average two services affected by the context change.

The houseB dataset has some additional sensors that can not be mapped to services, e.g. several motion detectors. Changes on these dimensions do not affect any of the available services. Since context changes on these dimensions happen frequently, on average less than 1% of the services are affected by a context change, i.e. it is possible to re-use the 99% of filtering results.

The domus dataset has many more additional sensors, e.g. for temperature and humidity. Since changes on these dimensions do not affect service availability, we can again re-use 99% of filtering results.

4.6.3 Runtime of the filtering algorithms on smart home datasets

This experiments compares the two algorithms with respect to the time needed to perform one round of service filtering.

The algorithms are evaluated using the following procedure:

1. Calculate initial candidate set $S_{cand,0} = \texttt{FilterServices}(c_0^u, S)$
2. For all context changes Δc_i with $1 \leq i \leq k$
   (a) Calculate new user context $c_i^u = c_{i-1}^u \times \Delta c_i$
   (b1) $S_{cand,i} = \texttt{FilterServices}(c_i^u, S)$
   or
   (b2) $S_{cand,i} = \texttt{FilterServicesWithReuse}(c_i^u, \Delta c_i^u, S, S_{cand,i-1})$

For both algorithms the test procedure is applied on the whole dataset with k context changes. Only step (b1) or only step (b2) is used, depending on the studied algorithm. The results of the experiment are listed in Table 4.2. For these small datasets, service filtering with reuse of previous results is slightly more than twice as fast as without reuse. However, the differences are negligible, both algorithms can perform service filtering in well under one millisecond. The contribution of the filtering phase to the total recommendation delay is insignificant for these datasets.

dataset	**without reuse [ms]**	**with reuse [ms])**
houseA (14 dimensions, 27 services)	$0.05 \pm 1 \times 10^{-4}$	$0.02 \pm 1 \times 10^{-4}$
houseB (22 dimensions, 26 services)	$0.06 \pm 2 \times 10^{-4}$	$0.02 \pm 1 \times 10^{-4}$
domus (105 dimensions, 53 services)	$0.19 \pm 1 \times 10^{-4}$	$0.04 \pm 1 \times 10^{-4}$

Table 4.2: Average runtime of one round of service filtering for three smart home datasets

4.6.4 Scalability of the filtering algorithms

Future smart homes could have a wider range and larger amount of sensors and actuators installed than todays testbeds. This means that the number of dimensions as well as the number of services will increase. In the following, we study the scaling properties of the two algorithms using synthetic datasets with many dimensions and services. For each replication we generate a HAC with the desired number of dimensions, one random user context and one random context change, and the desired number of services, with 10% of these services being affected by the change. Both algorithms are run on the same data and the total run-times are measured. This procedure is performed for 100 replications.

Figure 4.4 shows the averages of the measured run-times for up to 1000 services and dimensions. As formally analyzed in Section 4.5.4, both algorithms scale linearly with respect to the number of services. However, the growth rate of Algorithm 3 with reuse is much smaller than the growth rate of the original Algorithm 2. This is because `FilterServicesWithReuse` only has to perform the slow `fulfills` operation for 10%

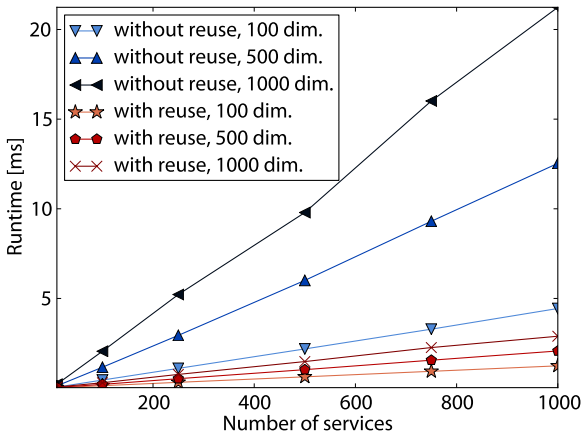


Figure 4.4: Run-time for one round of service filtering when scaling the number of dimensions, 10% of services are affected

of the services. For 90% of the services, the faster basis comparison already shows that the service can be reused from the previous filtering round.

In fact, it turns out that `FilterServicesWithReuse` with 1000 dimensions is faster than `FilterServices` with 100 dimensions. Even when applied to the largest dataset with 1000 dimensions and 1000 services, `FilterServicesWithReuse` can perform the filtering within less than 3 milliseconds. `FilterServices` needs 20 milliseconds for this dataset, i.e. 10% of the 195 milliseconds maximum recommendation delay are already spent on service filtering.

Increasing the ratio of affected services

In a second experiment it was evaluated how increasing the ratio of affected services influences the run-times. For Figure 4.5 the amount of affected services was increased from 10% to 100% in several steps. With a higher amount of affected services, algorithm `FilterServicesWithReuse` has to perform the `fulfills` operation for more services and run-times increase. However, even for 80% of affected services it is still slightly faster than `FilterServices`. If 100% of services are affected, `FilterServicesWithReuse` performs the basis as well as the containment operation for every service and is consequently slower than filtering without reuse.

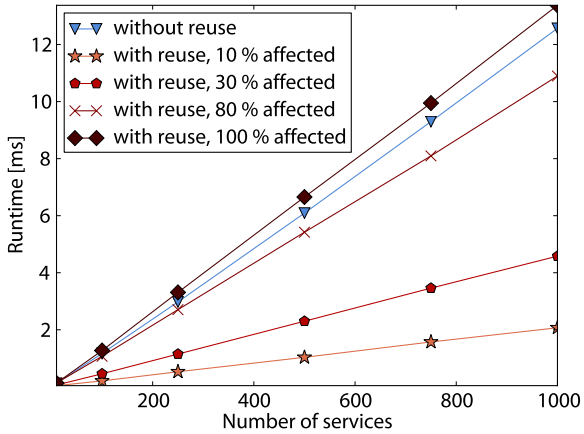


Figure 4.5: Filtering run-times for different percentages of affected services, for HAC with 500 dimensions

4.7 Summary

We introduced in this chapter the first smart assistant, a recommender system for the smart home. We identified that there is a wide variety of possible recommendations and that smart homes must have several strategies for utilizing the recommendation results. Several such strategies were presented. A large part of the chapter is dedicated to filtering unavailable services, as the first phase of service recommendation. The performance of the first, straightforward filtering algorithm was improved by reusing previous filtering results. The evaluation shows that both algorithms are fast enough for small smart home environments, but only the improved algorithm with service reuse is suitable for the larger installations that we may see in future smart homes.

Chapter 5

Ranking services based on user habits

The first approach to service ranking is focused on *user actions*. The idea is to learn, which user actions commonly occur in a given situation and which actions are related to each other. This knowledge can then be used to recommend services that can automate the most probable user actions in a given situation.

5.1 Example scenario

We demonstrate this idea based on the houseA dataset. For Figure 5.1 we have extracted from the dataset the four most common actions the user performs within five minutes of closing the front door of the smart home and applied a smoothing function to the raw data (see Section 5.3).

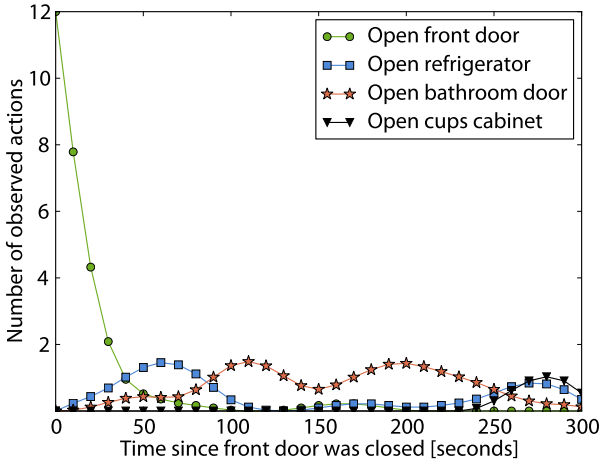


Figure 5.1: Typical actions after closing front door

Surprisingly, the most common user action is to immediately open the front door again. However, this action is only common within the first 40 seconds of closing the door – possibly the user has forgotten something in the car or can not carry all bought groceries inside at once. If a longer time has passed since closing the door, the user often heads either to the bathroom or for the refrigerator. If one looks even further into the future, say 20 minutes since closing the front door, no dominating actions can be found. Similar observations can be made in this dataset for many other user routines. For example, it can be clearly seen how different kitchen actions often occur together. The data supports rather detailed observations, e.g. after opening the dishwasher, actions concerning the plate and the cups cabinet are more likely than other kitchen actions.

We propose in this chapter a method for extracting these temporal relationships from a user's smart home history and building a model of the user's habits. Using this model, the method can then produce service recommendations that try to emulate the user's behavior. This means that for some given user context, the system recommends services that can automate those user actions that are most probable in this situation. The proposed method is unsupervised, i.e. no manual annotation of the smart home data is needed.

5.2 Overview of the proposed method

Training phase

The aim of the training phase is to build a model of the user's typical behavior in the smart home. However, the smart home can not directly observe the user behavior, i.e. it does not know which actions the user is performing. Instead, only the context changes that were caused by a user action can be observed. The proposed method extracts temporal relationships between user context and context changes from a collected smart home history. The extracted relationships are of the form "*if user is in situation x since δt_x time, then context changes y_1,y_2 and y_3 are common*. For example, the recommender system might learn that *"if the front door has been closed since 100-200 seconds, then a context change bathroom_door=open is common"*. The context change can be interpreted as circumstantial evidence that the user often heads for the bathroom shortly after closing the front door.

Service ranking

Based on the learned model of the user habits, our method can now identify probable context changes for a given user context and elapsed time vector. The predicted context changes are then matched to services, i.e. the method tries to find services that can automate the action causing this context change. The more probable the context change is, the higher ranked is the matching service. The ranking is based on the current user context and the elapsed time vector, i.e. it has to be recalculated whenever either of these change.

Restriction to nominal dimensions

The method proposed in this chapter is at the moment restricted to nominal dimensions, any numeric dimensions are omitted. This restriction leads to some limitations when modeling the user behavior. First, some situations in which user actions occur cannot be expressed using only nominal dimensions, e.g. it is not possible to formulate relationships such as "whenever it is colder than 18°C, the user turns on the heating". The situation-centric approach to service recommendations that will be described in Chapter 6 is much better suited for modeling this type of knowledge.

Second, the restriction to nominal data also means that it is not possible to model context changes on numeric dimensions. However, the limitations imposed by this restriction are minimal. The user actions that we want to model typically involve changing the internal status of some device or home component, which can be expressed as a nominal dimension. The user can rarely directly alter any numeric dimensions. For example, the indoor temperature can not be altered directly; any changes on this dimension are instead side effect of actions that involve other smart home infrastructure, e.g. heating or windows.

5.3 Training phase

During training, our method builds a model of the temporal relationships between user context and context changes. Since learning is performed on the basis of observed context changes, the recommender system can learn the user behavior from manually performed user actions as well as from service invocations. This means that users can continue to live in their home just as normal and do not have to switch to a smart home user interface for all of their interactions with the home.

Input to the training is the sequence of $k+1$ user contexts $[c_0^u, \ldots, c_k^u]$, the sequence of $k+1$ elapsed time vectors $[\Delta t_0^u, \ldots, \Delta t_k^u]$ for these user contexts and the sequence of k context changes $[\Delta c_1, \ldots, \Delta c_k]$. The underlying $\mathbb{H}$ has n context dimensions.

Output is the extracted behavior model

Procedure The training phase consists of two steps:

1. extract context change observations from a user's smart home history
2. sort the observations into time intervals and count the observations

5.3.1 Extracting observation tuples

Typically there will not be enough training data to exhaustively cover all possible vectors. With only ten binary context dimensions, already 2^{10} different user contexts could occur and enough training data will be available only for some of these vectors. For this reason, we proceed analogously to established algorithms such as Naïve Bayes and look at each context dimension individually. For example, we will extract information about which context changes are common if $c^u[\mathit{frontdoor}] = \mathit{closed}$, which changes are common if $c^u[\mathit{frontdoor}] = \mathit{open}$, if $c^u[\mathit{cabinet}] = \mathit{open}$, etc. The individual dimensions will be aggregated when calculating the service ranking (Section 5.4).

The first step of the training phase is to extract context change observations from the training data. Figure 5.2 shows as an example a short excerpt of six context changes, which could have been recorded while the user prepares some light snack.

Figure 5.2: Ten minutes of smart home activity

In this example, the refrigerator is opened at about 10:06, i.e. $c^u[\mathit{refrigerator}] = \mathit{open}$. After the refrigerator has been open for 30 seconds ($\Delta t^u[\mathit{refrigerator}] = 30$), it is closed again ($\Delta c = \{\mathit{refrigerator}\colon \mathit{closed}\}$). This information can be summarized as one observation tuple:

$$(c^u[\mathit{refrigerator}] = \mathit{open},\ \Delta t^u[\mathit{refrigerator}] = 30,\ \Delta c = \{\mathit{refrigerator}\colon \mathit{closed}\})$$
or shorter: $(\mathit{refrigerator}{=}\mathit{open}, 30, \mathit{refrigerator}{=}\mathit{closed})$

While the refrigerator is closed, one can extract four further such tuples:

$(\mathit{refrigerator}{=}\mathit{closed},\ 25,\ \mathit{cabinet}{=}\mathit{open})$
$(\mathit{refrigerator}{=}\mathit{closed},\ 235,\ \mathit{dishwasher}{=}\mathit{open})$
$(\mathit{refrigerator}{=}\mathit{closed},\ 340,\ \mathit{cabinet}{=}\mathit{closed})$
$(\mathit{refrigerator}{=}\mathit{closed},\ 360,\ \mathit{dishwasher}{=}\mathit{closed})$

The first tuple expresses that the cabinet door was opened 25 seconds after the refrigerator was closed. The other three tuples express that 235 seconds after closing the refrigerator the dishwasher was opened, that the cabinet was closed 340 seconds after the refrigerator was closed, and that the dishwasher was closed 360 seconds after the refrigerator was closed.

Formal extraction process

Algorithm 4 extracts observation tuples from the training data. The algorithm loops through the sequence of context changes. For each change Δc_i, c^u_{i-1} is the user context where this change occurs and Δt^u_{i-1} the time vector for this context. For each context dimension j the algorithm first calculates a context point c'^u that keeps only the dimension j, since we want to extract observation tuples individually for each dimension. The tuple $(c'^u, \Delta t^u_{i-1}[x], \Delta c_i)$ is then added to the list of tuples. For each Δc_i, up to n observation tuples can be extracted (since there are n nominal context dimensions).

To simplify the notation, observation tuples will in the following be written as $(x, \delta t_x, y)$, where x is a user context with only one dimension, δt_x describes how long the user has been in situation x and y is the context change that occurred.

Algorithm 4 Extract observation tuples

```
1:  procedure EXTRACTTUPLES(n, [c_0^u, ..., c_k^u], [Δt_0^u, ..., Δt_k^u], [Δc_0, ..., Δc_k])
2:      tuples = []
3:      for 1 ≤ i ≤ k do                          ▷ loop through all k context changes
4:          for 0 ≤ j < n do                      ▷ loop through all n context dimensions
5:              if c_{i-1}^u[j] ≠ ⊥ then
6:                  c'^u_{i-1} = {j: c_{i-1}^u[j]}        ▷ keep only dimension j in c'^u
7:                  tuples.append((c'^u_{i-1}, Δt_{i-1}^u[x], Δc_i))
8:              end if
9:          end for
10:     end for
11:     return tuples
12: end procedure
```

Set of all possible context settings

Situation x and context change y are context points with only one dimensions. In the following, we refer to a one-dimensional context point as a *context setting*. Since only nominal context dimensions are allowed, the set θ can be defined as as an enumeration of all possible context settings. For the example with three binary dimensions $\mathbb{D}_{refrigerator}$= $\{open, close\}$, $\mathbb{D}_{cabinet}$= $\{open, close\}$ and $\mathbb{D}_{dishwasher}$= $\{open, close\}$:

$$\theta = \{x_0 = \{refrigerator\text{: } open\}, x_1 = \{refrigerator\text{: } closed\}, x_2 = \{cabinet\text{: } open\},$$
$$x_3 = \{cabinet\text{: } closed\}, x_4 = \{dishwasher\text{: } open\}, x_5 = \{dishwasher\text{: } closed\}\}$$

5.3.2 Dealing with temporal data

We showed earlier that any prediction about the next context change must be based not only on the current user situation, but also on the time that the user has been in this situation. Thus, for any given context setting x we want to know how likely a change y is, given that the user has been in x since δt_x time. We define a δt_{max}, which is the maximum δt_x for which we still collect such temporal information. This maximum reflects the loss of correlation between current context and subsequent context changes if the user has been in the situation for a long time.

When extracting temporal relationships between user situations and typical context changes, we need to avoid over-fitting to the training data. There will always be some variations in user activities that should be tolerated by the algorithm. For example, it should not matter whether the user re-opens the front door after 3 or after 5 seconds. On the other hand, there can be a risk of over-generalizing the user behavior, e.g. if no difference is made between opening the front door instantly or after 5 minutes. The smart home datasets show the need to be more precise for small δt_x and more general for large δt_x. To this end the interval $[0, \delta t_{max}]$ is divided into a number of intervals with increasing width.

For example, for the houseA dataset we use $\delta t_{max} = 300$ and an interval width of 10 seconds for $\delta t_x \in [0, 60]$ and 30 seconds for $\delta t_x \in (60, 300]$.

A function $b(\delta t_x)$ is defined to look up the correct interval for a given δt_x (Equation 5.1).

$$b(\delta t_x) = \begin{cases} -1 & \text{if } \delta t_x > \delta t_{max} \\ \text{index of interval which includes } \delta t_x, & \text{otherwise} \end{cases} \tag{5.1}$$

5.3.3 Counting the observations

The final step of the training phase is to aggregate the extracted tuples $(x, \delta t_x, y)$ into the defined intervals. To make the intervals comparable with each other, the number of occurrences in each interval is divided by the interval's width. Moving average smoothing with window size 10 is applied to the intervals to further reduce the risk for over-fitting. Table 5.1 then defines a number of look-up functions for the smoothed counts.

$\text{count}(y\|x)$	How often change y has been observed in situation x
$\text{count}(y\|x,b)$	How often change y has been observed in situation x in time interval with index b; if $b = -1$ (*i.e.* $\delta t_x > \delta t_{max}$): $\text{count}(y\|x,-1)$=$\text{count}(y\|x)$

Table 5.1: Look-up functions for the smoothed counts

These functions return the number of observations of change y in a situation x in total and in the different time intervals. For large δt_x, temporal information is not useful, so the system falls back onto the total counts. Finally, Table 5.2 defines some sums of the counts to simplify later calculations.

$\text{csum}(x)$	Total number of observations for situation x, $\text{csum}(x) = \sum_{\forall y \in \theta} \text{count}(y\|x)$
$\text{csum}(x,b)$	Total number of observations for situation x in interval b $\text{csum}(x,b) = \sum_{\forall y \in \theta} \text{count}(y\|x,b)$
maxtotal	Maximum number of total observations for any situation x $\text{maxtotal} = \arg\max_{\forall x \in \theta} \text{csum}(x)$
maxtemp	Maximum number of observations for any situation x in any interval $\text{maxtemp} = \arg\max_{\forall x \in \theta, \forall b \neq -1} \text{csum}(x,b)$

Table 5.2: Important sums for the smoothed counts

5.3.4 Incremental learning

So far it was only discussed how the training process is applied to some recorded smart home data. In reality, the smart home should continuously learn from the observed user behavior. The proposed algorithm lends itself very well to incremental learning, i.e. it is easy to continuously add new observations to the model. In incremental learning mode, the algorithm retains the un-smoothed counts and adds new observations to the correct intervals. Smoothing is performed either after an observation was added or at regular time intervals. To allow the model to adapt to changed behavior patterns, the algorithm could regularly remove older observations.

5.4 Calculate the service ranking

Input to service ranking is the set of candidate services S_{cand}, the learned behavior model, the current user context c^u and the context change Δc that lead to c^u. The underlying $\mathbb{H}$ has n context dimensions.

Output is the list of ranked services, plus some measures of uncertainty and conflict

Procedure The service ranking is performed in two steps:

1. calculate probable context changes given the current user context c^u and the elapsed time vector Δt^u
2. match available services to the predictions

5.4.1 Context change prediction

Dempster-Shafer theory of evidence

Dempster-Shafer theory is a framework for achieving a consensus between several different information sources. It is a generalization of Bayesian probability theory. In contrast to Bayesian theory, it allows to attribute belief mass not only to individual hypotheses, but also to combinations of hypotheses. This makes Dempster-Shafer theory a valuable tool in human-centered environments, which are typically governed by variations – and thus uncertainties – in the subject's behavior. In the following we give just a short introduction to the theory and how we use the concepts in our work, more details can be found in [98].

Frame of discernment

In Dempster-Shafer theory, the frame of discernment Θ is the set of all hypotheses of interest. In our case, the frame of discernment is made up of all context changes that could occur given the user's current situation. This means that Θ changes if c^u changes and we thus denote the frame for c^u as Θ_{c^u}.

The frame Θ_{c^u} can be calculated by first setting $\Theta_{c^u} = \theta$, i.e. initialize the frame as the set of all possible context settings. Then for all context dimensions i with $c^u[i] \neq \perp$, remove $c = \{i\colon c^u[i]\}$ from Θ_{c^u}, i.e. remove all current settings since they can not occur as a context change.

Information sources

Each part of the current user context is considered a source of information. If the HAC has n nominal dimensions, then there will be typically n information sources. There can be less than n sources if one or several of the nominal dimensions are set to $\perp$, i.e. no value is known for this dimension.

Formally, a source is $(x, \delta t_x)$, where x is the current context setting for one dimension and δt_x expresses how long the setting has been active. For example, $c^u = \{$*refrigerator*: *closed*, *cabinet*: *open*, *bedroom*: *closed*$\}$ and $\Delta t^u = \{$*refrigerator*: *15*, *cabinet*: *120*, *bedroom*: *740*$\}$. There are three information sources: $(\{$*refrigerator*: *closed*$\}, 15), (\{$*cabinet*: *open*$\}, 120)$ and $(\{$*bedroomdoor*: *closed*$\}, 740)$. A belief distribution is then calculated for each source. This means that each source attributes some masses to context changes in Θ_{c^u}, which reflect how probable each change is according to the source. For instance, the source $(\{$*refrigerator*: *closed*$\}, 15)$ would probably attribute large masses to *refrigerator* = *open* and *cabinet* = *closed*, but small masses to *bedroomdoor*=*open*, since the former two are much more probable in this situation.

Uncertainties can be expressed by attributing some mass to combinations of elements or to Θ_{c^u} itself. For example, if it is quite likely that some context change will happen in the kitchen, but there is high uncertainty exactly which change will happen, one could attribute a large mass to the combination of all kitchen-related context changes, but a small mass to each of the individual changes. Masses attributed by one source must always sum up to 1.

Source weight

A source with a high number of observations should be trusted more than one with only a few observations. Sources for which no temporal information is available, i.e. $\delta t_x > \delta t_{max}$, should have a very small weight. Such sources should only be relevant for finding the combined masses if no source with $\delta t_x \leq \delta t_{max}$ is available, i.e. temporal knowledge always trumps general knowledge.

Equation 5.2 calculates the weight of a source $(x, \delta t_x)$. If the source has temporal knowledge, its weight is based on the number of observations in the current interval, scaled by the maximum number of observations of any source in any interval. Otherwise, the weight is based on the total number of observations of this source, scaled by the number of observations of the overall best source. In this case, the source is additionally discounted by a small ε, e.g. $\varepsilon = 10^{-4}$.

$$\text{weight}_{(x,\delta t_x)} = \begin{cases} \dfrac{\text{csum}(x, b(\delta t_x))}{\text{maxtemp}} & \text{if } b(\delta t_x) \neq -1 \\ \dfrac{\text{csum}(x)}{\text{maxtotal}} * \varepsilon & \text{otherwise} \end{cases} \tag{5.2}$$

Attributing the masses

The masses attributed by a source are based on the counts extracted from the data during the training phase. A context change with a high number of observations should be attributed a

larger mass than one with a lower number of observations. Equation 5.3 calculates the mass attributed to a context change y by a source $(x, \delta t_x)$:

$$m_{(x,\delta t_x)}(y) = \frac{\text{count}(y|x, b(\delta t_x))}{\text{csum}(x, b(\delta t_x))} * \text{weight}_{(x,\delta t_x)} \tag{5.3}$$

The mass attributed to a change $y \in \Theta_{c^u}$ is the ratio of the number of observations of y in interval $b(\delta t_x)$ to the total number of observations in this interval. The calculated mass is then discounted by the weight of the source. In a slight abuse of notation, we write $m(y)$ instead of $m(\{y\})$.

Finally, the mass put on Θ_{c^u} can be calculated as the remaining masses for reaching a sum of 1:

$$m_{(x,\delta t_x)}(\Theta_{c^u}) = 1 - \sum_{y \in \Theta_{c^u}} m_{(x,\delta t_x)}(y) \tag{5.4}$$

It holds that $m_{(x,\delta t_x)}(\Theta_{c^u}) = 1 - weight_{(x,\delta t_x)}$. For a large weight, high mass is assigned to the possible context changes and only a small mass is assigned to Θ_{c^u}, i.e. the uncertainty of the source is low. For a small weight, a small mass is put on the context changes, more mass is put on Θ_{c^u}, i.e. the uncertainty of the source is high.

Source combination

The different pieces of evidence are combined using Dempster's rule of combination [98]. Dempster's rule requires information sources to be independent. This assumption holds true for our test datasets. In case of dependent information sources, the cautious rule of combination can be used instead [28].

The general form of Dempster's combination rule for two information sources is:

$$m_{1\oplus 2}(A) = \sum_{B \cap C = A, A \neq \emptyset} m_1(B) * m_2(C) \tag{5.5a}$$

Equation 5.5b formulates Dempster's combination rule for our problem domain, to combine two sources $(x_1, \delta t_{x_1})$ and $(x_2, \delta t_{x_2})$. In the equation we make use of the fact that we only assign masses to singleton elements and to Θ_{c^u}, which simplifies the calculations.

$$\begin{aligned} m_{(x_1,\delta t_{x_1})\oplus(x_2,\delta t_{x_2})}(y) = {} & m_{(x_1,\delta t_{x_1})}(y) * m_{(x_2,\delta t_{x_2})}(y) \\ & + m_{(x_1,\delta t_{x_1})}(y) * m_{(x_2,\delta t_{x_2})}(\Theta_{c^u}) \\ & + m_{(x_1,\delta t_{x_1})}(\Theta_{c^u}) * m_{(x_2,\delta t_{x_2})}(y) \end{aligned} \tag{5.5b}$$

The combined uncertainty of sources $(x_1, \delta t_{x_1})$ and $(x_2, \delta t_{x_2})$ is:

$$m_{(x_1,\delta t_{x_1})\oplus(x_2,\delta t_{x_2})}(\Theta_{c^u}) = m_{(x_1,\delta t_{x_1})}(\Theta_{c^u}) * m_{(x_2,\delta t_{x_2})}(\Theta_{c^u}) \tag{5.5c}$$

The conflict between the sources, i.e. the masses that can not be attributed to any elements or subsets of Θ_{c^u}, can be calculated according to Equation 5.6a [98]. An example in our problem domain is given in Equation 5.6b, for two sources $(x_1, \delta t_{x_1})$ and $(x_2, \delta t_{x_2})$ and $\Theta_{c^u} = \{y_1, y_2\}$.

$$K_{1,2} = \sum_{B \cap C = \emptyset} m_1(B) * m_2(C) \tag{5.6a}$$

$$\begin{aligned} K_{(x_1,\delta t_{x_1}),(x_2,\delta t_{x_2})} &= m_{(x_1,\delta t_{x_1})}(y_1) * m_{(x_2,\delta t_{x_2})}(y_2) \\ &+ m_{(x_1,\delta t_{x_1})}(y_2) * m_{(x_2,\delta t_{x_2})}(y_1) \end{aligned} \tag{5.6b}$$

More than two sources can be combined iteratively, i.e. combine the first two sources, then combine the result with the third source, etc. After combining all sources, $m_{comb}(y)$ is the combined mass for context change y, K_{comb} the overall of conflict between the sources and $m_{comb}(\Theta_{c^u})$ is an overall measure of the uncertainty. Then the plausibility that a context change y will occur next can be calculated as $pls_{comb}(y) = m_{comb}(y) + m_{comb}(\Theta_{c^u})$.

In our implementation we chose to transform masses into commonalities and use Dempster's rule of commonalities to combine sources [98]. This step converts the combination equation into a different form, which can be calculated much faster. The final outcome of the calculation is the same.

5.4.2 Service matching

Each of the possible context changes can be representative of some user action, i.e. the context change is a result of performing this action. The higher the plausibility for a context change is, the more common is also the respective user action. The next step is now to match the candidate services S_{cand} to the predicted context changes.

Unrelated changes and side effects

Not every predicted context change must actually be representative of an action the user typically performs in the current situation. Observed context changes can also be completely unrelated (e.g. it started raining shortly after the user closed the refrigerator) or be side effects of a previous user action (e.g. it got colder after opening the front door). In the former case, unrelated changes will be observed more rarely than actual user actions and thus the plausibility attributed to the context change will be low. In the latter case, the side effect context changes might be observed very often, i.e. could be given a large plausibility. For above example, the side effect of opening the front door might be matched with a service *turn on AC* and this irrelevant service would be shown to the user as a recommendation.

We can not yet propose a solution for this problem. Further research is necessary to *(i)* find out how often the issue occurs in real smart homes and *(ii)* find an appropriate strategy to resolve the problem. If the issue occurs rarely, it would be easiest to ignore it at the risk of occasionally and temporarily showing irrelevant recommendations. Otherwise, the system needs the ability to distinguish between context changes that actually represent a user action

and other context changes. It may be necessary to include user input to achieve this, e.g. the user interface could include an option that allows to mark a recommendation as irrelevant.

Calculating the ranking

Let y be a predicted context change, i.e. the system predicts that y will occur next with plausibility $pls(y)$. A service $s \in S_{cand}$ can be recommended if executing the service will result in context change y. First calculate the combined effect that the service has in the current c^u using Algorithm 1 from Section 3.6.4, i.e. $s^{comb\_eff} = \texttt{CombineEffects}(c^u, s)$. Then a service can be recommended based on the predicted change if $y \in s^{comb\_eff}$, i.e. if s matches y.

Table 5.3 shows an example of service matching for five services $s_1, \ldots, s_5$ and four predicted context changes $y_1, \ldots, y_4$. If a service s_i matches a context change y_j, then the corresponding intersection is marked with a ✓. Services are ranked according to the plausibility of the context changes, e.g. a service that matches the context change with highest plausibility will be ranked highest. If there is more than one ✓in one row, then several services match the context change, i.e. the services are *alternatives* and can be marked as such in the user interface. If there is more than one ✓in one column, then the service matches several context changes. The ranking algorithm can rank the service according to the combined plausibility or choose the context change with the highest plausibility. Neither of these cases occurred in the smart home test datasets that the method was evaluated on.

	s_1	s_2	s_3	s_4	s_5
$y_1, pls(y_1) = 0.3$	✓				
$y_2, pls(y_2) = 0.11$					✓
$y_3, pls(y_3) = 0.03$				✓	
$y_4, pls(y_4) = 0.01$					

Table 5.3: Matching services to predicted context changes
✓for y_i and s_j signifies that $y_i \in s_j^{comb\_eff}$

The calculated ranking is only valid for the current user context and has to be recalculated whenever c^u or Δt^u change. In reality, often changes in Δt^u do not lead to changes in the output of the change prediction algorithm. This is because even though the raw elapsed times have changed, the time intervals stayed the same. This means that the service ranking is stable for long time frames, in particular when no activity is happening in the home.

5.4.3 Interpretation of the results

The previous steps returned not only a list of service recommendations, but also some measure of the conflict and the uncertainty in the system. We discuss in the following the edge cases of low/high uncertainty and low/high conflict with respect to a GUI. These considerations will be revisited in the evaluation section.

High uncertainty, low conflict No temporal sources are available, thus there is not enough information to make reliable recommendations. The user interface should present a layout that allows to select from the full list of services. More common services could be highlighted.

High uncertainty, high conflict Does not occur.

Low uncertainty, low conflict There are one or several temporal sources that are in agreement. This is the best-case scenario. The user-interface should display only the few best recommendations. The other recommendations should still be accessible, but can be hidden behind an extra button in the interface.

Low uncertainty, high conflict There are several temporal sources that are in disagreement. The user-interface should display several of the best recommendations. Again, the other recommendations should still be accessible, but can be hidden behind an extra button in the interface.

5.4.4 Runtime analysis

The service ranking has to be recalculated whenever c^u and Δt^u change. When using a resolution of one second for the elapsed time vector, then Δt^u changes every second and the service ranking has to be performed every second. The total cost of service ranking $r_{ranking}$ is made up of the costs for context change prediction, calculating combined service effects, service matching and sorting of the recommendations:

$$r_{ranking} = r_{predict} + r_{calc\_eff} + r_{match} + r_{sort}$$

Each of these costs is discussed individually in the following. Experimental results of runtimes in large smart home installations will be presented in Section 5.5.9.

Context change prediction $r_{predict}$

Let n be the number of information sources (i.e. number of nominal dimensions) and $|\Theta_{c^u}|$ be the size of the frame of discernment. In case of binary dimensions there are n potential context changes, each of which flips the current status of the respective dimension, i.e. $|\Theta_{c^u}| = n$. If there are more than two values for some dimensions, then $|\Theta_{c^u}| > n$. For example, if each dimension has four nominal values, then $|\Theta_{c^u}| = 3 * n$. In the used datasets nearly all nominal dimensions are binary (with the exception of the window shutters in the domus dataset, which have four different states), i.e. $|\Theta_{c^u}| \approx n$.

The context change prediction algorithm looks up $n * |\Theta_{c^u}|$ counts from the learned model, calculates $n * |\Theta_{c^u}|$ masses and performs approximately $n * |\Theta_{c^u}|$ multiplications to combine the masses. The cost of context change prediction is $r_{predict} = O(n * |\Theta_{c^u}|)$.

Calculate combined effects for the services r_{calc_eff}

The combined effect has to be calculated for $|S_{cand}|$ services, thus $r_{calc\_eff} = |S_{cand}| * T_{comb}$, where T_{comb} is the cost of calculating the combined effect for one service.
Optimization: Let s^{comb_pre} be the combination of all m side effect preconditions of s, i.e. $s^{comb\_pre} = s^{spre_0} \uplus s^{spre_1} \uplus \ldots \uplus s^{spre_{m-1}}$. s^{comb_pre} has to be calculated only once for each service. Then perform basis comparison to check if the combined effect might have changed: only if $\texttt{decimal}(B(s^{comb\_pre}) \& B(\Delta c)) > 0$ the combined effect has to be recalculated. Experimental results show that $T_{basis} \ll T_{comb}$.

Check service matching r_{match}

For each combination of service and predicted context change it must be checked whether the service can elicit this context change. For this check the containment operation is used, $r_{match} = |S_{cand}| * |\Theta_{c^u}| * T_{fulfills}$, where $T_{fulfills}$ is the cost of one application of the containment operation (see Section 4.5.4).
Optimization: Perform the matching only for services where s^{comb_eff} changed, otherwise the previous matching still applies. Additionally match each new prediction in Θ_{c^u} with all services in $|S_{cand}|$.

Sort the recommendations by plausibility r_{sort}

When using a sorting algorithm such as QuickSort, $r_{sort} = O(|\Theta_{c^u}| * \log |\Theta_{c^u}|)$.

Overall runtime

With above optimizations, r_{calc_eff} and r_{match} are largest when the first set of recommendations is calculated, since all combined effects and service matchings have to be computed. For all subsequent iterations r_{calc_eff} and r_{match} are very small, since a large majority of previously calculated combined effects and service matchings can be reused.

The overall runtime is dominated by $r_{predict}$, which has quadratic complexity with respect to the number of nominal context dimensions if $|\Theta_{c^u}| \approx n$ (as is the case in the used datasets).

5.5 Evaluation

In this section the proposed algorithm is compared with a Naïve Bayes classifier using the houseA and houseB datasets. It is also evaluated how the choice of time intervals influences the recommendation results and the usefulness of the conflict and uncertainty measures are explored. The section finishes with an evaluation of the algorithm runtimes.

5.5.1 Choice of datasets

Only houseA and houseB record the daily life of one smart home inhabitant over a longer time frame. In the domus and SM4All datasets, test subjects were inside the smart home mostly for shorter experiments. These datasets does not provide enough data for the individual participants and can not be used for evaluating the algorithm.

5.5.2 Procedure

Input to the evaluation is the sequence of $k+1$ user contexts $[c_0^u, \ldots, c_k^u]$, the sequence of $k+1$ elapsed time vectors $[\Delta t_0^u, \ldots, \Delta t_k^u]$ for these user contexts and the sequence of k context changes $[\Delta c_1, \ldots, \Delta c_k]$. Since the order of the events is important, the dataset cannot be randomized. Instead 10-fold cross-validation is performed as follows: in the first fold, the first 10% of the dataset are test data, the rest are training data; in the second fold the second 10% of events are test data, the rest are training data, etc.

Following procedure is applied to each of the ten folds. Train the model on the training data using the selected method. Then for each c_i^u and Δt_i^u in the test data, *(i)* calculate possible context changes, *(ii)* calculate the plausibility (or probability for Naïve Bayes) for these context changes using the selected algorithm, *(iii)* identify from Δc_{i+1} which user action a was performed, if Δc_{i+1} can not be matched to any user action, then do not proceed further for c_i^u, *(iv)* calculate service recommendations based on the predicted context changes, *(v)* compare the highest ranked recommendation with action a. The necessary descriptions of service effects and user actions were provided manually. All algorithms were implemented in Python and experiments were performed on a PC with an Intel Core 2 Duo CPU 3 GHz and 4 GB RAM.

5.5.3 Metrics

The recommendation results are evaluated using the standard metrics of precision, recall and F1. For each service, count the number of true positives (tp), false positives (fp) and false negatives (fn). Table 5.4 illustrates true positives, false positives and false negatives using as example the service/action *Open front door*.

	recommended service	**actual action**
true positive	*Open front door*	*Open front door*
false positive	*Open front door*	any other action
false negative	any other service	*Open front door*

Table 5.4: True positives, false positives and false negatives for example *Open front door*

Recall corresponds to the true positive rate ($recall = \frac{tp}{tp+fn}$): whenever the *Open front door* user action occurs in the test data, how often is the correct service recommended? Precision measures, how relevant the recommendations are ($precision = \frac{tp}{tp+fp}$): whenever

Open front door is the best recommendation, how often does it actually occur? F1 is the harmonic mean of precision and recall ($F1 = 2 * \frac{precision*recall}{precision+recall}$).

The overall precision, recall and F1 are calculated as the average over all services, weighted by the number of their occurrences. If several service recommendations are displayed in the user interface, then often recall increases (only one of the recommendations has to be correct) and precision decreases (several of the recommendations are irrelevant). All results are listed using the 90% confidence interval over 10 folds.

5.5.4 Comparison with Naïve Bayes

The method is compared with a Naïve Bayes classifier. Since Dempster-Shafer theory is a generalization of Naïve Bayes, this method is equivalent to our method without the temporal properties and without any non-specificity. A random classifier is used as a baseline in the comparison. The experiments use $\delta t_{max} = 300$ and an interval width of 10 seconds for $\delta t \in [0,60]$ and 30 seconds for $\delta t \in [60,300]$.

Only the best recommendation is used

Table 5.5 lists evaluation results when only the one best service recommendation is used. Our method significantly outperforms Naïve Bayes for both datasets. The Naïve Bayes algorithm will always predict the most common context change without any regard for temporal relations, i.e. it will always predict "Close cupboard", regardless of how long the cupboard has been open.

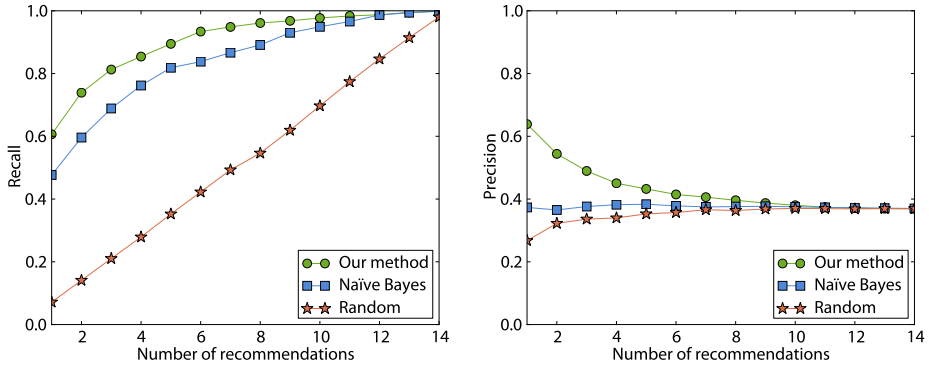
Method	**Recall**	**Precision**	**F1**
houseA dataset			
Our method	0.61 ± 0.02	0.63 ± 0.04	0.59 ± 0.02
Naïve Bayes	0.48 ± 0.03	0.38 ± 0.03	0.40 ± 0.03
Random	0.08 ± 0.01	0.30 ± 0.05	0.10 ± 0.02
houseB dataset			
Our method	0.73 ± 0.12	0.78 ± 0.10	0.72 ± 0.12
Naïve Bayes	0.55 ± 0.20	0.50 ± 0.21	0.51 ± 0.21
Random	0.11 ± 0.04	0.49 ± 0.19	0.16 ± 0.07

Table 5.5: Results for both datasets

The evaluation results show that this approach is too coarse and results in much lower recall and precision when compared to our approach. For the houseB dataset some confidence intervals are rather wide. The reason is that the dataset is partially dominated by one of the mercury switch sensors that are attached to drawers. In some of the replication folds, events for this sensor occur extremely often, which makes prediction easier.

Showing several recommendations

Figure 5.3 shows for the houseA dataset, how recall and precision are influenced by the number of service recommendations which are shown to the user. In this dataset there are 13 pairs of binary services plus one service to activate the toilet flush. This means that the user can typically be be shown at most 14 services[1].



(a) Recall (b) Precision

Figure 5.3: Number of recommendations vs recall and precision for houseA dataset

On the left-hand side, of Figure 5.3 it can be seen that the recall increases with the number of service recommendations for all methods. If the user is shown five services, the recall of our method reaches 0.9, i.e. in 90% of cases the correct service is contained in the five best recommendations. For less than 13 recommendations, our method always outperforms Naïve Bayes, for 13 or more recommendations both algorithms achieve similar results. The recommendation precision of our method decreases when the number of service recommendations increases, because more irrelevant recommendations are shown to the user. For up to 9 recommendations, our method has higher precision than Naïve Bayes; for more than 9 recommendations both algorithms have similar precision.

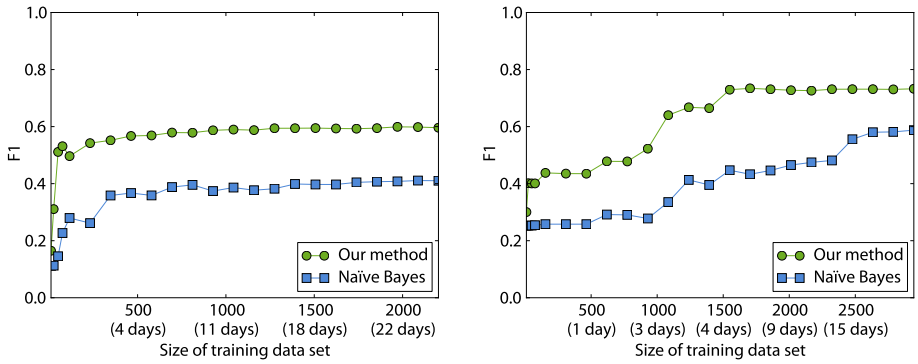
We performed this experiment also with the houseB dataset and found that our method outperforms Naïve Bayes for up to 14 recommendations, for more recommendations both algorithms achieve similar results. In 90% of cases the correct service is contained in the four best recommendations calculated by our method.

5.5.5 Size of the training dataset

Figure 5.4 shows the relationship between the size of the training dataset and the quality of the service recommendations. For this experiment only the first x events in the dataset

[1] An exception is the initial phase of the evaluation where the initial status of the devices is not known. For example, since it is not known whether a door is open or closed at the beginning of the dataset, the user is shown both services for opening and closing this door

were used for training (with gradually increasing x), the learned model was evaluated on the whole dataset.



(a) houseA

(b) houseB

Figure 5.4: Size of training dataset vs F1 for houseA and houseB datasets

In the houseA dataset (depicted in Figure 5.4a), a lot of home activity happens already in the first few days. Both our method and the Naïve Bayes classifier learn user routines quickly, however our method achieves a higher F1 at all times. The figure shows that our method can give good recommendations already after a few days of training. For both methods the learned model is very stable, i.e. new training data does not result in huge drops of recommendation quality. For the houseB dataset (depicted in Figure 5.4b), the learning of user habits happens at a slightly slower rate[2]. However, our method again outperforms the Naïve Bayes classifier and can provide useful recommendations already after a few days of learning.

5.5.6 Choice of time intervals

It is important to evaluate, how susceptible prediction accuracy is to the choice of time intervals. Table 5.6 lists recall, precision and F1 for the houseA dataset for several different interval choices. In the first half of the table, shorter and longer settings for δt_{max} are evaluated. When decreasing δt_{max}, the algorithm is quite stable and recall degrades only slowly. Only very short $\delta t_{max} <= 30$ should be avoided. Increasing the δt_{max} to 1200 has no significant effect on the algorithms performance.

[2]In Section 3.7 it was listed that the houseB dataset contains nearly 40000 sensor events. However, this dataset is dominated by pressure sensors placed on bed and chair; in particular the former fires constantly during the night. Since these events can not be mapped to a specific service, they are irrelevant for evaluating service recommendations. As can be seen in Figure 5.4b, about 3000 instances of service recommendation remain for this dataset.

Intervals	**Recall**	**Precision**	**F1**
baseline (Table 5.5)	0.61 ± 0.02	0.63 ± 0.04	0.59 ± 0.02
Influence of δt_{max}			
$\delta t_{max} = 120$	0.60 ± 0.01	0.68 ± 0.04	0.59 ± 0.01
$\delta t_{max} = 60$	0.59 ± 0.02	0.66 ± 0.03	0.58 ± 0.02
$\delta t_{max} = 30$	0.58 ± 0.02	0.66 ± 0.02	0.57 ± 0.01
$\delta t_{max} = 20$	0.55 ± 0.02	0.66 ± 0.03	0.54 ± 0.02
$\delta t_{max} = 10$	0.21 ± 0.04	0.11 ± 0.03	0.11 ± 0.03
$\delta t_{max} = 1200$	0.61 ± 0.02	0.64 ± 0.03	0.58 ± 0.02
Influence of interval widths (same width for all intervals)			
width=2s	0.60 ± 0.02	0.66 ± 0.03	0.59 ± 0.02
width=4s	0.61 ± 0.02	0.67 ± 0.04	0.59 ± 0.02
width=6s	0.61 ± 0.02	0.66 ± 0.03	0.59 ± 0.01
width=30s	0.58 ± 0.02	0.58 ± 0.02	0.55 ± 0.02
width=100s	0.54 ± 0.02	0.59 ± 0.02	0.52 ± 0.02

Table 5.6: Influence of time intervals on prediction accuracy for houseA dataset

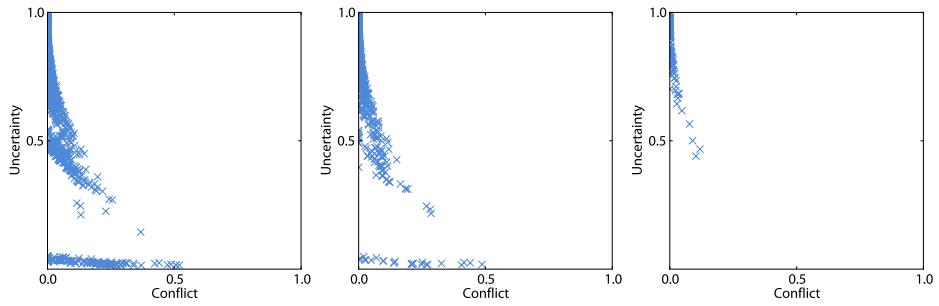
In the second half of Table 5.6, the algorithm is tested with different interval widths, while $\delta t_{max} = 300$. The results show that even very short intervals do not lead to over-fitting to the test dataset. However, for long intervals, the prediction accuracy degrades due to the over-generalization of the user behavior. The results indicate that the algorithm is quite stable with regard to the choice of time intervals, as long as very wide intervals and very small δt_{max} are avoided.

5.5.7 Exploring conflict and uncertainty

The potential use of the measures of conflict and uncertainty was discussed earlier. In the following, a first evaluation of the actual information value of these measures is presented. Please note that a partially supervised approach is used for this initial evaluation, in contrast to the rest of this chapter.

For this experiment, the algorithm was applied the whole Kasteren houseA dataset. Each cross in the scatter-plots in Figure 5.5 represents one service recommendation: the x-coordinate of a cross represents how much conflict the system had when making this recommendation and the y-coordinate represents how much uncertainty there was in the system. Scatter-plot (a) shows only the most successful service recommendations, where the service with highest probability matches the actually performed user action. It is not possible to identify any conflict-uncertainty region were the algorithm is more or less successful.

Scatter-plot (b) shows less successful recommendation results, where the correct service was not in the best two recommendations, i.e. the user would have to be shown three or more service recommendations. There is a much lower density of data points in the lower-left



(a) First rec. is correct (b) At least 3 rec. needed (c) At least 5 rec. needed

Figure 5.5: Recommendation success vs conflict and uncertainty

corner of the plot compared to plot (a), but there are still a large number of points in the upper-left corner.

The final scatter-plot (c) shows even less successful results, where the correct service was not in the best four recommendations. It can be seen that there are no longer any data points in the lower-left corner of the plot, i.e. for low conflict and low uncertainty, the correct service is found within the four best recommendations.

Using conflict and uncertainty to reduce the number of recommendations

A second experiment was performed to show how the number of recommendations that are presented to the user can be reduced based on conflict and uncertainty. Let *cutoff* be the maximum number of recommendations that are shown to the user. However, if uncertainty$<$0.4, show only *dynamic cutoff*$<$*cutoff* recommendations, i.e. make the selection easier for the user if we can be reasonably sure that the correct service will be included in the first *dynamic cutoff* recommendations. Table 5.7 compares for *cutoff=5* and *cutoff=10* the success of this strategy in terms of average number of recommendations shown, recall and precision.

For *cutoff=5*, without any dynamic cutoff, on average five recommendations are shown to the user. This average decreases to 4.86 if the list is cut at four items if uncertainty is low, while still upholding the same high recall. The average decreases further to 4.57 for an *dynamic cutoff=2*. In this case, the recall decreases as well, i.e. the user will less often find the correct service within the shown recommendations. However, one can argue the overall effect is still positive, since recall decreases only by 2%, while the number of recommendations decreases by 8.6%.

The usefulness of *dynamic cutoff* is even more pronounced for *cutoff=10*. With an *dynamic cutoff=4* the user is shown only 9.13 instead of 10 items, while recall is equal and precision increases slightly. We think that these initial results are promising and plan to further expand in this direction in the future.

Cutoff	# of Recs.	Recall	Precision
Maximum 5 recommendations (cutoff=5)			
fixed cutoff=5	5.00 ± 0.00	0.89 ± 0.01	0.43 ± 0.02
dynamic cutoff=4	4.86 ± 0.03	0.89 ± 0.01	0.43 ± 0.02
dynamic cutoff=2	4.57 ± 0.09	0.88 ± 0.01	0.44 ± 0.02
Maximum 10 recommendations (cutoff=10)			
fixed cutoff=10	10.00 ± 0.00	0.98 ± 0.01	0.38 ± 0.03
dynamic cutoff=4	9.13 ± 0.19	0.98 ± 0.01	0.38 ± 0.03
dynamic cutoff=2	8.84 ± 0.25	0.96 ± 0.01	0.39 ± 0.03

Table 5.7: Results for dynamic selection of number of recommendations

5.5.8 Runtimes on test datasets

Table 5.8 compares the training and prediction times for the methods. All algorithms process the datasets in less than one second. Our method extracts not only overall counts, but also has to sort occurrences into time intervals and perform smoothing, the training times are thus the highest.

Method	Training time [ms]	Prediction time $r_{predict}$ [ms] total	per instance
houseA dataset			
Our method	460.21 ± 11.82	152.48 ± 4.78	0.66 ± 0.02
Naïve Bayes	52.29 ± 2.63	37.35 ± 1.44	0.16 ± 0.01
Random	0.01 ± 0.00	12.46 ± 0.15	0.05 ± 0.00
houseB dataset			
Our method	810.91 ± 35.86	250.22 ± 59.69	0.81 ± 0.19
Naïve Bayes	91.33 ± 6.08	67.91 ± 8.75	0.22 ± 0.03
Random	0.03 ± 0.00	17.04 ± 0.51	0.06 ± 0.00

Table 5.8: Training and prediction times for both datasets, in milliseconds

More interesting is the time needed for predicting context changes. It can be seen that while our algorithm has the longest prediction times, one set of predictions is calculated in less than one millisecond. The results show that for these small smart home installations with only a few context dimensions and services, the performance requirements for service recommendation can easily be fulfilled.

5.5.9 Algorithm scalability

Again, it is necessary to evaluate how well the proposed algorithm can perform in larger smart home installations. In 5.4.4 we determined that the total time of calculating the service ranking is $r_{ranking} = r_{predict} + r_{calc\_eff} + r_{match} + r_{sort}$. Initial evaluations indicate that indeed the time needed to calculate context change predictions dominates the other runtimes.

For this reason, we concentrate in the following on the scalability of the context prediction algorithm. The evaluation is performed using a synthetically generated dataset that imitates the data seen in the houseA and houseB dataset. Input to the dataset generator are the number of dimensions n and the average size of the frame of discernment $|\Theta_{c^u}|$. The results of the evaluations are shown in Table 5.9.

n	$\lvert\Theta_{c^u}\rvert$	$r_{predict}[ms]$
dimensions have on average two nominal values		
100	200	5.15 ± 0.03
500	1000	30.83 ± 0.04
1000	2000	74.20 ± 0.07
dimensions have on average six nominal values		
100	500	6.09 ± 0.01
500	2500	43.35 ± 0.04
1000	5000	133.55 ± 0.15

Table 5.9: Prediction times in larger smart home installations

The first half of the table evaluates environments with binary context dimensions, as is the case in the houseA and houseB datasets. In the largest tested dataset with 1000 dimensions, one set of predictions is calculated in around 75 microseconds. When adding up the prediction runtime with the time needed for service filtering (less than 3 milliseconds) and effect calculating, service matching, and sorting, the total time needed for calculating service recommendations is well under the set limit.

Even though binary context dimensions are prevalent in the available smart home datasets, it is important to also consider context dimensions with several nominal states. For example, in the *domus* dataset, window shutters could have four states (*open*, *closed*, *lamellas open*, *lamellas closed*). In the second half of the table we have $|\Theta_{c^u}| = 5 * n$, i.e. dimensions have on average six nominal values. Since the number of potential context changes increased, also the prediction runtime increases. However, the table shows that our algorithm can still meet the time limit for service recommendation, even in large-scale installations with 1000 context dimensions.

5.6 Summary

This chapter presented a strategy for ranking services by emulating the behavior of the user. The proposed method extracts temporal relationships between context and context

changes from a user's smart home history and builds a model of the user's habits. Based on the user's current situation, the method then tries to recommend services that match probable next actions. Evaluations were performed on two publicly available test datasets and it was shown that the proposed method *(i)* can produce correct recommendations with 61% and 73% accuracy, for 90% of cases the correct service is contained in the top five recommendations, *(ii)* significantly outperforms a Naïve Bayes classifier and *(iii)* is stable with regard to choice of parameters, as long as extreme values are avoided. It was also shown that the method can fulfill the performance requirements for service recommendation even in large-scale installations.

Chapter 6

Ranking services based on user preferences

This chapter presents a second approach to service ranking that focuses on *user preferences* instead of user actions. A user preference can be any context situation that is favored by the user. For example, user preferences can:

- express comfortable environmental settings, e.g. room temperature and lighting,
- describe typical user activities, e.g. watching news on channel 8 every evening,
- set preferred home conditions, e.g. after 9pm the front door should be locked.

During service ranking, the proposed method compares the current user context with the user preferences and identifies services that can bring the user nearer to some preference. The proposed algorithm works like a classical feedback loop: *(i)* context data shows that the current situation is unsatisfactory, *(ii)* the system recommends some service that can move the user towards a preferred state, *(iii)* the user choses to execute the service or the service is executed automatically, and *(iv)* context data shows that the unsatisfactory situation was rectified. One advantage of this second approach is that nominal as well as numeric context dimensions are supported.

Limitations

The material presented in this chapter has a lower level of maturity than the material presented in Chapter 5. Problematic is in particular the lack of suitable datasets, so that only a limited evaluation of the proposed method could be performed. The lack of data also makes it hard to motivate one specific strategy for ranking services based on user preferences. For this reason, we present in Section 6.2.3 some initial ideas for service ranking and refrain from choosing one particular strategy until further evaluations have been performed.

In this chapter

This chapter starts with modeling user preferences within HAC. We then describe how to measure the current distance between users and their preferences and the impact of executing a service with respect to a preference. The resulting service score is used to perform the actual service ranking. A partial evaluation of the method is done in Section 6.3. Finally, we shortly summarize initial research efforts for automatically mining user preferences from the user's smart home history.

6.1 User preference model

We situate user preferences within HAC by modeling them as context scopes. Compared to context points, context scopes have the advantage that they allow users to formulate imprecise preferences. The first example in Table 6.1 shows one such imprecise preference, which describes a range of comfortable room temperatures in the kitchen and the bedroom. The second row states a preferred home condition: whenever it rains, the windows should be closed. Preference p_3 represents a typical user activity: every evening when the user is in the bedroom, the motorized bed is in a sitting position and the TV is set to channel 8.

Comfortable room temperature
$p_1 = \{kitchen\_temp : [19-22], bedroom\_temp : [16-18]\}$
Window closed when it is raining
$p_2 = \{rain\_status : \{raining\}, kitchen\_window : \{closed\}\}$
Watching TV before going to bed
$p_3 = \{location : \{bedroom\}, time\_of\_day : \{evening\},$
$TV : \{channel8\}, bed : \{sitting\_position\}\}$

Table 6.1: Examples for user preferences expressed as context scopes

Each preference p_i is attached with a *preference weight* w_i with $0 \leq w_i \leq 1$. The larger the weight, the more important is the respective preference. Larger weights can be given to critical preferences, e.g. a preference to have the front door locked during the night. Services that can fulfill these preferences will be ranked higher than services that fulfill low-weighted preferences.

6.2 Calculate the service ranking

Input to service ranking is the set of candidate services S_{cand}, the set of m user preferences together with their weights $P = \{(p_0, w_0), \ldots, (p_{m-1}, w_{m-1})\}$, the current user context c^u and the context change Δc that lead to c^u. The underlying $\mathbb{H}$ has n context dimensions.

Output is the list of ranked services S_{ranked}, ordered by how beneficial they are for the user.
Procedure The service ranking is performed in three steps:

1. calculate $dist(c^u, p)$ as a measure of the distance between the current user context and a preference p
2. calculate $score(s|c^u, p)$ as an approximation of the impact of executing s with respect to p
3. perform the actual ranking using some ranking strategy

6.2.1 Distance between current context and user preference

The user preference is already fulfilled

In the best case, the user preference is already fulfilled, i.e. $(c^u \times B(p)) \in p$. Then the distance $dist(c^u, p) = 0$. Typically we want to avoid recommending any services that move the user away from the preferred situation.

Some dimensions of the preference are not fulfilled

In the second case, one or several of the preference dimensions are currently not fulfilled, i.e. $(c^u \times B(p_k)) \notin p$. Let $dist_i(c^u, p)$ be the distance between c^u and p on dimension i. This distance can be zero ($p[i]$ is fulfilled) or one ($p[i]$ is not fulfilled), see Equation 6.1. [1]

$$dist_i(c^u, p) = \begin{cases} 0 & \text{if } B(p)[i] = 0 \quad \triangleright\ i \text{ is not set in } p \\ 0 & \text{if } B(p)[i] = 1 \wedge c^u[i] \in p[i] \quad \triangleright\ p[i] \text{ is fulfilled by } c^u[i] \\ 1 & \text{if } B(p)[i] = 1 \wedge c^u[i] \notin p[i] \quad \triangleright\ p[i] \text{ is not fulfilled by } c^u[i] \end{cases} \tag{6.1}$$

The overall distance is the sum of the distances on all dimensions (Equation 6.2), i.e. $dist(c^u, p)$ is equal to the number of dimensions that are not fulfilled.

$$dist(c^u, p) = \sum_{i=0}^{n-1} dist_i(c^u, p) \tag{6.2}$$

The preference can currently not be fulfilled

There are several context dimensions that are outside of the control of the smart home, i.e. can not be changed by executing a service. Time, outdoor temperature and user location are examples of such context dimensions. A user preference that contains these dimensions, may be unsatisfiable in the current user context. For example, Table 6.1 contained a user

[1] Using Equation 6.1 it cannot be expressed that 5°C is further away from a preference of 18-22°C than from a second preference of 10-12°C. One could also choose to model the distance in a more exact and non-binary fashion by calculating the numeric distance to the border of the preference. However, since we can not exactly calculate the impact of executing a service on the user preference (see following section), this additional complexity offers only limited advantages.

preference p_3 that expresses that the user likes to watch TV in bed in the evenings. If $c^u[\mathit{time\_of\_day}] = \mathit{morning}$, p_3 is unsatisfiable in c^u. Services for turning on the TV to channel 8 and for raising the bed to a sitting position might bring the user nearer towards p_3. However, these services should not be recommended, since it is impossible to reach p_3

A context dimension i is outside of the control of the smart home if there is no service that has an effect or side effect that changes i. The set of all such dimensions can be calculated once and only has to be updated when new services or context sources are added to the home. Whenever a preference p contains any of the dimensions in this set and this dimension is not already fulfilled in c^u, then p is currently unsatisfiable and $\mathit{dist}(c^u, p) = \infty$.

6.2.2 Service score

Executing service s results in a context change on some dimensions of c^u. Consequently, the distance between c^u and a preference p can increase, decrease or stay the same when s is executed. The $\mathit{score}(s|c^u, p)$ approximates the impact of executing s in c^u:

$\mathit{score}(s\|c^u, p) < 0$	Executing s moves the user further away from p
$\mathit{score}(s\|c^u, p) = 0$	Executing s does not change the distance between the user and p
$\mathit{score}(s\|c^u, p) > 0$	Executing s moves the user nearer towards p

The higher the score, the nearer s can bring the user to the desired preference p. However, service effects are context scopes, i.e. define a range of possible values for the new user context. Without executing s, one can not know exactly the new context c'^u and can not know whether c'^u fulfills the user preference. This is why any calculation of $\mathit{score}(s|c^u, p)$ can only ever approximate the impact of s.

As a first step for calculating $\mathit{score}(s|c^u, p)$, determine the combined effect $s^{\mathit{comb\_eff}}$ that the service has in the current c^u using Algorithm 1 from Chapter 3. Then a $\mathit{score}_i(s|c^u, p)$ is calculated for each dimension i, approximating the impact of the execution of s on dimension i. The same value ranges apply: if $\mathit{score}_i(s|c^u, p) < 0$, then s moves the user away from $p[i]$; if $\mathit{score}_i(s|c^u, p) = 0$, then s does not change the distance to $p[i]$; if $\mathit{score}_i(s|c^u, p) > 0$, then s moves the users nearer towards $p[i]$.

Overview of calculating $\mathit{score}_i(s|c^u, p)$

Table 6.2 gives an overview of the different cases that can arise when calculating the $\mathit{score}_i(s|c^u, p)$. Several of the cases are easy to handle: If dimension i is not part of the user preference, then executing s has no impact on $p[i]$, thus $\mathit{score}_i(s|c^u, p) = 0$. If the execution of s does not change i, then also $\mathit{score}_i(s|c^u, p) = 0$. The other two cases will be detailed in the following.

	i not set in p $(B(p)[i]=0)$	$p[i]$ currently not fulfilled $(B(p)[i]=1, c^u[i] \notin p[i])$	$p[i]$ currently fulfilled $(B(p)[i]=1, c^u[i] \in p[i])$
s does not change $c^u[i]$ $(B(s^{comb\_eff})[i]=0)$	0	0	0
s changes $c^u[i]$ $(B(s^{comb\_eff})[i]=1)$	0	Equation 6.4	Equation 6.5

Table 6.2: Different cases when calculating $score_i(s|c^u, p)$

p[i] is currently not fulfilled and s changes $c^u[i]$

If the preference is currently not fulfilled on dimension i, then $dist_i(c^u, p) = 1$. Executing the service can either decrease the distance or not have any impact on the distance. It is not possible to further increase the distance. Three sub-cases are possible when comparing the two value ranges $p[i]$ and $s^{comb\_eff}[i]$:

(a) $s^{comb\_eff}[i]$ is fully covered by $p[i]$ → every execution of s will fulfill $p[i]$
(b) $s^{comb\_eff}[i]$ is partially covered by $p[i]$ → some executions of s can fulfill $p[i]$
(c) no overlap between $s^{comb\_eff}[i]$ and $p[i]$ → no execution of s will fulfill $p[i]$

Case (a) is the best case: since every execution of s will fulfill the user preference we can set $score_i(s|c^u, p) = 1$. Case (c) is the worst case, no execution of s can fulfill the preference, therefore $score_i(s|c^u, p) = 0$. In case (b), executing s might move the user towards the preference. Such a service should only be recommended if there are no better alternatives for moving towards p. The ratio of $s^{comb\_eff}[i]$ that is covered by $p[i]$ can be used as an indicator of how likely it is that the execution of s can fulfill $p[i]$. The higher this ratio, the higher also $score_i(s|c^u, p)$. Equation 6.3 shows how to calculate this ratio for numeric and nominal dimensions.

$$dcov_i(s^{comb\_eff}, p) = \begin{cases} \dfrac{|s^{comb\_eff}[i] \cap p[i]|}{|s^{comb\_eff}[i]|} & \text{if } \mathbb{D}_i \text{ is nominal} \\ \dfrac{\text{length of range shared by } s^{comb\_eff}[i] \text{ and } p[i]}{\text{length of } s^{comb\_eff}[i]} & \text{if } \mathbb{D}_i \text{ is numeric} \end{cases} \tag{6.3}$$

Since Equation 6.3 also incorporates cases (a) and (c), one can simply set:

$$score_i(s|c^u, p) = dcov_i(s^{comb\_eff}, p) \tag{6.4}$$

p[i] is currently fulfilled and s changes $c^u[i]$

In this case $dist_i(c^u, p) = 0$ and executing the service can either increase the distance or not have any impact on the distance. It is not possible to further decrease the distance.

Again, there are three sub-cases when comparing $p[i]$ and $s^{comb\_eff}[i]$:

(a) $s^{comb\_eff}[i]$ is fully covered by $p[i]$	$\rightarrow$	after every execution of s, $p[i]$ will still be fulfilled
(b) $s^{comb\_eff}[i]$ is partially covered by $p[i]$	$\rightarrow$	after some executions of s, $p[i]$ could still be fulfilled
(c) no overlap between $s^{comb\_eff}[i]$ and $p[i]$	$\rightarrow$	after every execution of s, $p[i]$ will no longer be fulfilled

Case (a) is the best case with $score_i(s|c^u, p) = 0$: even though executing s changes c^u on i, the preference is still fulfilled. This case only arises if $dcov_i(s^{comb\_eff}, p) = 1$. Case (c) is the worst case, executing s moves the user away from p. Since this should be avoided, set $score_i(s|c^u, p) = penalty$, with $penalty \leq -1$. Case (b) might move the user away from p, to avoid this risk also set $score_i(s|c^u, p) = penalty$. Equation 6.5 summarizes the calculation of $score_i(s|c^u, p)$ for the three sub-cases.

$$score_i(s|c^u, p) = \begin{cases} 0 & \text{if } dcov_i(s^{comb\_eff}, p) = 1 \\ penalty & \text{otherwise} \end{cases} \tag{6.5}$$

Overall service score

The total service score is the sum of the $score_i(s|c^u, p)$ for all dimensions (Equation 6.6).

$$score(s|c^u, p) = \sum_{i=0}^{n-1} score_i(s|c^u, p) \tag{6.6}$$

By setting an appropriate *penalty*, it can be controlled how much a service is penalized for moving away from an already fulfilled preference dimension. If $penalty = -1$, then moving away from p on one dimension can be balanced by moving closer to p on another dimension. For $penalty < -1$, several dimensions have to move closer towards p to balance out one dimension that moves the user away. If $penalty = -\infty$, then moving away on one dimension can not be balanced out.

6.2.3 Service ranking

The final step is to determine the overall service ranking. First, calculate the distance between the context c^u and all preferences $p \in P$; calculate also the service score for all combinations of services $s \in S_{cand}$ and all preferences $p \in P$. The result will be a matrix of service scores. An example of such a matrix is given in Table 6.3 for four user preferences and five services, using $penalty = -\infty$.

Preference p_2 is unsatisfiable in c^u and no service scores are calculated. Service s_3 can fulfill preference p_1, but moves the user away from preference p_3, which has a very large weight. Service s_1 is an alternative to s_3 with respect to p_1, but it is not sure if executing s_3 will really fulfill the preference.

	s_1	s_2	s_3	s_4	s_5
p_1 $(w_1 = 0.2, dist(c^u, p_1) = 2)$	0.8	0	1	0	0
p_2 $(w_2 = 0.1, dist(c^u, p_2) = \infty)$	-	-	-	-	-
p_3 $(w_3 = 0.7, dist(c^u, p_3) = 0)$	0	0	$-\infty$	0	$-\infty$
p_4 $(w_4 = 0.3, dist(c^u, p_4) = 4)$	0	0	0	0.2	0

Table 6.3: Example matrix of service scores
each matrix entry is the $score(s_k|c^u, p_l)$ for service s_k with respect to preference p_l

Ranking criteria

A number of considerations must be made when ranking the services based on the score matrix:

Which preference should the algorithm try to fulfill first?

- preference with largest weight
- preference with smallest distance to c^u
- preference with lowest remaining distance after service execution

How to deal when service moves c^u nearer to one preference, but away from another?

- avoid moving away from fulfilled preferences
- avoid moving away from partially fulfilled preferences
- avoid moving away from preferences with larger weights

In practice we would like to combine several of these criteria to find the best ranking. This effectively transforms service ranking into a multi-criteria decision problem. Multi-criteria decision analysis (MCDA) is a research field that is concerned with structuring and solving such problems; an introduction to MCDA can be found in [92]. The output of applying an MCDA method to our problem is not necessarily an overall service ranking. Instead of obtaining the one optimal service, we will often obtain a set of several so-called *pareto-optimal* services. A pareto-optimal service dominates any non pareto-optimal service, i.e. it is better in all criteria. However, it does not dominate any other pareto-optimal services, i.e. it is better only in some, but not in all of the criteria. This means that an additional decision strategy is needed to establish a total order ranking. In the simplest case, one might choose to show all pareto-optimal services in an alphabetical or randomized order.

Unfortunately, only the criteria "preference with largest weight" is applicable in the available smart home datasets, as will be shown in Section 6.3. For this reason, we do not further investigate ranking strategies in this dissertation. We also suspect that preferred ranking criteria might vary for different smart home inhabitants. For future research it would therefore be advisable to explore the different criteria and MCDA in an actual smart home setting and gather feedback from several test users.

6.2.4 Runtime analysis

Let $|P|$ be the number of user preferences and $|S_{cand}|$ be the number of candidate services. When creating the score matrix without any optimizations, $|P|$ calculations of $dist(c^u, p)$, $|S_{cand}|$ calculations of s^{comb_eff} and $|P| * S_{cand}$ score calculations are necessary. However, the same optimizations as previously can be applied by checking which user preferences and services are affected by the context change. Then for each context change, only a few rows and columns of the matrix have to be recalculated. Therefore, the time needed for calculating the score matrix is negligible with respect to the total recommendation time. Since the ranking strategies are not yet thoroughly defined, no runtime analysis of the strategies can be performed.

6.3 Evaluation

6.3.1 Choice of datasets

Only the houseA and houseB can be used for the evaluation, since none of the other datasets record the daily life of one smart home inhabitant over a longer time frame. Unfortunately, both datasets have severe limitations that allow only a partial evaluation of the proposed method. When comparing the datasets with the three types of user preferences that were mentioned at the beginning of this chapter, the limitations become obvious:

- preferences express comfortable environmental settings, e.g. room temperature
  → *cannot be evaluated due to lack of environmental context dimensions in houseA and houseB datasets*
- preferences describe typical activities, e.g. watching news on channel 8 every evening
  → *cannot be evaluated due to lack of knowledge about typical user activities*
- preferences as preferred home conditions e.g. front door should be locked after 9pm.
  → *partially evaluated in this section*

The support for numeric context dimensions, which is one of the advantages of the method proposed in this chapter, can not be evaluated with the houseA and houseB datasets. Additionally, no information is known about the preferences of the user that is living in the prototype flat. In order to obtain realistic evaluation results, we wanted to make as few assumptions about the user's preferences as possible. For this reason, we avoided the manual creation of user preferences describing hypothetical user activities. Instead, we opted to evaluate the method based on the user's preferred home conditions.

Dataset preparation

We argue that typical context settings can be a reasonable approximation of the user's preferred conditions. For example, in the houseA dataset, the front door is closed for 99% of the time frame covered by the dataset. In comparison, the bedroom door is closed 43% of the time and open 57% of the time, i.e. closing the front door has a higher priority for

the user. Based on this insight, we extract preferences from the two datasets according to following procedure: Let *time_total* be the length of time frame covered by the dataset. Then for each context dimension and each value that this dimension can take, create a preference $p_i = \{dimension : \{value\}\}$. Calculate the time the user spent in this context setting and let weight $w_i = \frac{time\ spent\ in\ setting}{time\_total}$. For the houseA dataset, this process results in 28 user preferences.

Limitations

This approach for evaluation is somewhat different from how the method is intended to be used. In reality we would expect user preferences with more than one dimensions, which can better capture specific preferred smart home settings and activities. The evaluation can therefor only show that the proposed method generally works. We can not make any claims about the usefulness of the recommendations in a realistic smart home setting.

6.3.2 Procedure and metrics

Input to the evaluation is the sequence of $k+1$ user contexts $[c_0^u, \ldots, c_k^u]$ and the sequence of k context changes $[\Delta c_1, \ldots, \Delta c_k]$. 10-fold cross-validation is used in the manner that was described in Section 5.5.2. The user preferences are extracted from the training dataset. Then for each c_i^u in the test data, *(i)* identify from Δc_{i+1} which user action a was performed, if Δc_{i+1} can not be matched to any user action, then do not proceed further for c_i^u, *(ii)* rank services with respect to the extracted preferences, *(iii)* compare the highest ranked service recommendation with a. The results are evaluated using the standard metrics of precision, recall and F1 (see Section 5.5.3).

Selected ranking strategy

Each of the extracted preferences uses only one dimension, e.g. the $dist(c^u, p)$ is either zero or one. Also, since all services are binary, one preference matches to one service and vice versa. This means that situations were a service moves the user closer to one preference but away from another preference can not arise. Therefore, only ranking by preference weights can be evaluated and no multi-criteria decision strategy is needed.

Why compare with Naïve Bayes?

On most of the featured context dimension, the standard setting is to have the device/home infrastructure closed or switched off. The respective user preferences consequently have a large weight. This means that typically the algorithm will rank any service for closing/switching off an item higher than opening/switching on another item. This expected behavior is similar to the one observed with the Naïve Bayes algorithm in the previous chapter. If the proposed method works correctly, then ranking results should be similar to those of Naïve Bayes.

6.3.3 Results

The evaluation results are listed in Table 6.4, together with the results of the Naïve Bayes and the Random classifier that were first presented in Section 5.5.4. As expected, the method presented in this section produces similar rankings as the Naïve Bayes classifier. The recall and F1 are slightly lower for our method than with Naïve Bayes, the precision is the same for both methods. When increasing the number of recommendations that were shown to the user, the curves for Naïve Bayes and our method closely stayed together. The recall for our method is always slightly lower than Naïve Bayes. The precision of our method is slightly higher when more than three recommendations are shown.

Method	**Recall**	**Precision**	**F1**
Our method	0.45 ± 0.07	0.38 ± 0.06	0.38 ± 0.06
Naïve Bayes	0.48 ± 0.03	0.38 ± 0.03	0.40 ± 0.03
Random	0.08 ± 0.01	0.30 ± 0.05	0.10 ± 0.02

Table 6.4: Recall, precision and F1 for houseA dataset

The results show that the proposed algorithm is functional and can give results similar to the Naïve Bayes algorithm. However, a number of important aspects could not yet be evaluated. First, user preferences with more than one dimension can better capture specific preferred smart home settings and activities and can hopefully increase recommendation accuracy. Second, numeric dimensions are not supported by the method proposed in the previous chapter nor by the Naïve Bayes algorithm. It is important to evaluate, how well preference-based service recommendation can close this gap.

6.4 Automatic mining of user preferences

It has been stressed in this dissertation that one of the major disadvantages of current smart homes is that they are not learning homes. Configuration of the home has to be done *a priori* and is consequently very inflexible. The same issue arises with the method that was presented in this chapter, since information about user preferences is needed.

For this reason, Fei Li and Sanjin Sehic from the Vienna University of Technology have in collaboration with me explored the automatic learning of user preferences in [60]. The main idea is to collect context data in the smart home and automatically mine the user's preferences using subspace clustering. Subspace clustering is an unsupervised machine learning technique which can be applied to high-dimensional data; an introduction is given in [81]. Output of subspace clustering is a set of clusters, each containing objects that are similar to each other on a subset of the data dimensions. A subspace cluster can easily be transferred into a context scope on the respective context dimensions and with a data range that covers the objects in the cluster.

However, context data poses several challenges with respect to data heterogeneity: *(i)* mix of numeric and nominal dimensions, *(ii)* different value domain and data distribution

on different dimensions and *(iii)* varying importance of data dimensions (dimensions related to security are more important than those related to a comfortable environment). Existing subspace clustering algorithm only partially address these challenges.

Therefore we propose in [60] a subspace clustering algorithm that is specific to context data. After an initial pre-clustering phase that finds one-dimensional clusters on each context dimension, a modified FP-Growth algorithm [46] is used to identify clusters that should be merged to higher-dimensional clusters. In the process, a cluster quality measure is used that takes into account the number of dimensions in the merged cluster, the weight of these dimensions, and the number of objects. The effectiveness of the algorithm was evaluated on synthetically generated datasets. The results of these evaluations are very promising and further evaluations on actual smart home data are planned.

6.5 Summary

In this chapter we presented an alternative approach for service ranking. In this approach user preferences guide the ranking, a service is ranked higher if it can bring the user closer to a preferred situation. In contrast to ranking based on user habits, also numeric dimensions are supported. However, the approach is less mature and evaluations are preliminary due to the lack of suitable datasets.

Chapter 7

Smart installation assistant

The mass adoption of smart home technology is hindered by complicated installation and upgrading procedures. Today's smart home inhabitants either have a technical background or rely on expensive external contractors for installing and configuring their home. The contractors are needed not only for the initial setup of the smart home, but also whenever a new sensor or actuator is added to the system. Under these circumstances, many potential users will be discouraged from buying smart home solutions.

We propose an *installation assistant* that enables the use of the smart home recommender system with minimal configuration effort. The idea is that the user simply plugs in a new device, starts using it, and lets the installation assistant figure out what the device is doing. After a short while, the smart home system will have integrated any new sensor and status information into the context space, learned the capabilities of any new services, and starts utilizing the new knowledge for service recommendation.

In this chapter

We start this chapter by further motivating the need for a smart installation assistant. We then describe the data that is collected by the assistant and propose algorithms for learning the capabilities of a service. We also show how the assistant was integrated into the SM4All smart home prototype. Finally, the proposed algorithms are evaluated in a series of experiments with the SM4All smart home and with synthetic data.

7.1 Motivation

In the previous chapters we simply assumed that the necessary service descriptions are made available to the recommender system in some way. Manufacturers of pervasive devices could be one potential source of these descriptions. For instance, the manufacturer of a motorized bed might ship the hardware together with a small file that describes the preconditions and effects of raising and lowering the bed.

However, it turns out that many service descriptions can not be provided *a priori* by the device manufacturer since the descriptions have to be localized for each specific smart home. Listed in the following are four example situations where localization is needed:

(i) a motorized curtain is installed in front of a motorized window (situation seen in the SM4All smart home)
necessary localization: precondition for service *open window* must state that the service can only be executed if the curtain is currently open, precondition for service *close curtain* must state that service can only be executed if the window is currently closed

(ii) several cabinets with motorized drawers are installed in a small kitchen
necessary localization: preconditions must reflect that some of the drawers can only be properly opened when some of the other drawers are closed

(iii) motorized window curtains are installed in all rooms of a smart house
necessary localization: descriptions of service side effects depend on the placement of luminosity sensors and the house layout.

(iv) there is a large tree outside of one of the windows in the previous example
necessary localization: side effects for opening window curtains must be adapted to reflect that less light can come into the room when opening these curtains in the summer, when the tree has leaves

Service descriptions must therefore be manually fitted to the environment such that they reflect the interdependencies between the installed devices and sensors. The descriptions must also be updated whenever new sensors and devices are added to the system. However, the manual description of services is tedious and error-prone, hindering non-expert users to adopt the technology [82]. Instead they have to call a technician whenever a new device or sensors has to be installed, which is not acceptable for most users.

In this chapter we propose a method for automatically learning the capabilities of context-altering services without any need for manual input from users or technicians. When a new actuator is plugged into the smart home, our method starts observing the behavior of any new service with respect to its environment. Then machine learning is used to find typical patterns in the observed behavior.

7.2 Service execution history

Figure 7.1 visualizes the idea of observing the service behavior. In the initial situation the user is in the kitchen, the radio is switched off, the kitchen temperature is 22°C and a sound sensor registers minimal noise in the kitchen. When the radio is switched on, two context changes are generated: the internal status of radio is now set to channel 2 and the sound sensor registers more noise. The observed context information is stored in the execution history for this service. For each invocation of the service, the *precondition history* contains the current context prior to the invocation and the *effect history* contains the context changes that were observed after the invocation.

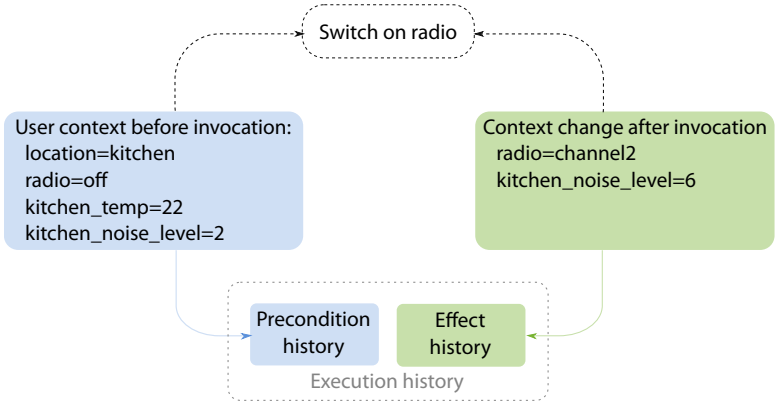


Figure 7.1: Collecting the service execution history

The effect of a service can be identified by finding the context changes that commonly occur after its execution. The preconditions of a service are those conditions under which the service is commonly invoked. Side effects are any additional precondition-effect patterns that can be found in the observations.

7.2.1 Challenges of recording the effect history

Recording the precondition history for a service is straightforward: whenever the service is executed, add the user context at the moment of the execution to the precondition history. For recording the effect history, two issues have to be considered.

Sampling time

The first challenge is to determine the optimal time for sampling context changes. The optimal *sampling time* is dependent on the context type that is being observed. To identify a suitable sampling time, we measured in the SM4All smart home the time between executing a service and the corresponding context change (called the *latency* of the context dimension). We found that the latency differs greatly between devices. Turning on one of the lamps is fast, with a reaction time of about 30 milliseconds. In contrast, when raising or lowering the motorized bed it takes 21 seconds until the action is finished and the device publishes an updated internal status.

A solution to this problem is to record all context changes after a service execution up to a very long sampling time. However, the longer the sampling time is, the higher is also the probability that other services are executed during this time and that the context changes caused by multiple services are overlapping. The optimal sampling time is thus a trade-off between missing important context changes and observing irrelevant changes. Different settings for the sampling time and the influence of this parameter on the learning results will be evaluated in Section 7.5.

Multiple context changes on one dimension

In Chapter 5 we have seen that users often perform several actions with the same device in a short time frame, e.g. opening the refrigerator and closing it after a few seconds. If the sampling time is sufficiently long, then several context changes will be recorded on one context dimension. When deciding which of these context changes should be added to the effect history we follow two different strategies based on the character of the dimension:

(i) If the dimension describes the status of a device, then add only the first context change on this dimension to the effect history. This strategy is based on the assumption that a service execution can change the internal status of a device only once, every subsequent context change on this dimension cannot be attributed to the same service execution. This assumption holds in the available smart home datasets. Device status is typically modeled as a nominal dimension, therefore this strategy is mainly used for nominal dimensions.

(ii) For all other dimensions, add only the last context change within the sampling time to the effect history. If the dimension is numeric, the mean value of the dimension within a short interval ending with the sampling time might be used instead. These dimensions typically contain environmental data, e.g. temperature and humidity. We are not interested in the first context change after the execution (e.g. temperature increases by 0.05 °C), but in the long-term development (e.g. after 5 minutes the temperature has increased by 5°C).

7.2.2 Formal definition of precondition and effect history

Definitions 13 and 14 formalize the precondition history H_s^{pre} and the effect history H_s^{eff} for a service s. H_s^{pre} and H_s^{eff} together form the execution history of the service.

Definition 13 (Precondition history) *The precondition history H_s^{pre} of a service s after q executions of s is a vector of q context points, $H_s^{pre} = [c_0^u, \ldots, c_{q-1}^u]$. Each c_j^u records the current user context at the moment of the $j-th$ execution of s.*

Definition 14 (Effect history) *The effect history H_s^{eff} of a service s after q executions of s is a vector of q context points, $H_s^{eff} = [\Delta c_0, \ldots, \Delta c_{q-1}]$. Each Δc_j combines all context changes that were observed within the sampling time after the $j-th$ execution of s. If several changes were observed on the same dimension, the conflict is resolved as described above.*

Table 7.1 gives an example of an execution history, documenting seven executions of a service s for opening window blinds. The left hand side (a) of the table shows in seven rows the context c_j^u before the $j-$th execution. There is no connection between rows, i.e. other services may have been executed and changed c^u in between two executions of s.

The context is described using five dimensions, including numeric and nominal dimensions. It stands out that the window blinds are always closed before the execution of s, whereas all other context dimensions vary. This observation indicates that $c^u[blinds] = closed$ is a precondition of opening the blinds.

	Temp	Light out	Blinds	Light in	TV
c_0^u	22	bright	closed	dark	off
c_1^u	21	dark	closed	dark	on
c_2^u	22	dark	closed	bright	off
c_3^u	20	bright	closed	dark	on
c_4^u	22	dark	closed	bright	on
c_5^u	22	bright	closed	dark	off
c_6^u	22	bright	closed	dark	off

(a) Precondition history

	Blinds	Light in	TV
Δc_0	open	bright	$\perp$
Δc_1	open	$\perp$	off
Δc_2	open	$\perp$	$\perp$
Δc_3	open	bright	$\perp$
Δc_4	open	$\perp$	$\perp$
Δc_5	open	bright	$\perp$
Δc_6	open	bright	$\perp$

(b) Effect history

Table 7.1: Sample execution history for service *Open blinds*

The right hand side (b) of Table 7.1 shows in corresponding rows the context changes Δc_j observed after the $j-$th execution of s. One clear difference between (a) and (b) is that in the former all five dimensions are used for describing c_j^u, but in the latter only a subset of dimensions is used for Δc_j and many values are missing. This is because c_j^u describes the full context of the environment, containing values for every available context dimension. On the contrary, Δc_j contains information only about those dimensions whose value changed after the execution.

In the effect history (b) it stands out again that for each execution a context change is observed on the "Blinds" dimension, indicating that $\Delta c[blinds] = open$ is an effect of opening the blinds. Context changes on the "Light in" dimension are only sometimes observed, hinting that $light\_in = bright$ could be a possible side effect of s. And indeed, when reviewing the precondition history (a), it can be seen that this effect always occurs if $light\_out = bright$ and $light\_in = dark$, i.e. a side effect of s is: if it is dark inside and bright outside, then opening the window blinds will result in it being bright inside. The single change on the "TV" dimension is an example of observing an irrelevant context change. In this case, the TV was turned off shortly after executing s, so that the context change occurred within the sampling time for s.

7.2.3 Operations on precondition and effect history

Tables 7.2 and 7.3 list common operations on the collected histories. The operations are listed only for the precondition history, but apply analogously to the effect history.

For nominal dimension i	
$\texttt{count}_i(H_s^{pre}\|v)$	Number of observations of value v on dimension i in H_s^{pre}
$\texttt{most\_common}_i(H_s^{pre})$	Value that has been observed most often on dimension i in H_s^{pre}, $\texttt{most\_common}_i(H_s^{pre}) = \underset{\forall v \in \mathbb{D}_i}{\arg\max}\ \texttt{count}(H_s^{pre}, i\|v)$

Table 7.2: Common operations on history for nominal dimensions

For numeric dimension i	
$\texttt{count}_i(H_s^{pre})$	Number of observations on dimension i in H_s^{pre}, missing values $\perp$ are not counted
$\texttt{min}_i(H_s^{pre})$	Minimum value observed on dimension i in H_s^{pre}
$\texttt{max}_i(H_s^{pre})$	Maximum value observed on dimension i in H_s^{pre}

Table 7.3: Common operations on history for numeric dimensions

7.3 Learning service capabilities

Input is the n-dimensional HAC $\mathbb{H}$ and the collected precondition and effect history for the service of interest. The algorithms also make use of several external parameters ϑ_e, ϑ_p, ϑ_s, τ, φ, β, each of which will be explained when it first occurs.

Output are the preconditions, main effects and side effects for the service.

Procedure Capability learning is performed in three steps:

1. learn the main effects
2. learn the preconditions
3. learn the side effects

7.3.1 Learning main effects

The main effects of a service are constituted by those context changes which commonly occur when the service is executed. The changes may not necessarily occur in all service executions, since due to e.g. network errors some changes may be missing from the effect history. Therefore we consider a context change to be part of the main effect of a service, if it occurs in at least $\lfloor \vartheta_e * k \rfloor$ of all q service executions[1]. For instance, a ϑ_e of 0.9 ignores occasional missing context changes, while at the same time filtering out any unrelated context changes from simultaneously executed services.

Algorithm 5 for learning the main effects of service s takes as input $\mathbb{H}$ and the collected effect history for s, with a global constant ϑ_e. Starting from an empty service effect, for all dimensions in $\mathbb{H}$ do: If the dimension is nominal, find the most commonly occurring nominal value v. If v occurs in enough service executions then set dimension i in the main effect to v (Line 4-8).

If the dimension is numeric, outlier detection [17] must first be performed to remove all values that deviate markedly from the rest of the values on i. Such outlier values can occur if the simultaneous execution of other services also resulted in context changes on i. All found outliers are replaced with the missing value $\perp$. If after outlier removal enough values remain, then set the interval from the minimum value to the maximum value on $\mathbb{D}_i$ as a main effect of s (Line 9-13).

[1] The floor function $\lfloor x \rfloor$ rounds x to largest previous integer.

Algorithm 5 Learn service main effects

1: **procedure** LEARNMAINEFFECTS($\mathbb{H} = [\mathbb{D}_0, \ldots, \mathbb{D}_{n-1}]$,$H_s^{eff} = [\Delta c_0, \ldots, \Delta c_{q-1}]$)
2: $s^{eff} = \{\}$
3: **for** $0 \leq i < n$ **do**
4: **if** $\mathbb{D}_i$ is nominal **then**
5: $v = \texttt{most\_common}_i(H_s^{eff})$
6: **if** $\texttt{count}_i(H_s^{eff}|v) \geq \lfloor \vartheta_e * q \rfloor$ **then**
7: $s^{eff}[i] = v$
8: **end if**
9: **else**
10: $H_s'^{eff} = \texttt{remove\_outliers}_i(H_s^{eff})$
11: **if** $\texttt{count}_i(H_s'^{eff}) \geq \lfloor \vartheta_e * q \rfloor$ **then**
12: $s^{eff}[i] = [\texttt{min}_i(H_s'^{eff}), \texttt{max}_i(H_s'^{eff})]$
13: **end if**
14: **end if**
15: **end for**
16: **return** s^{eff}
17: **end procedure**

7.3.2 Learning preconditions

The main preconditions of a service are those context conditions that commonly hold when the service is executed. Analogously to the main effects, we say that a context point is a precondition of a service if it holds for at least $\lfloor \vartheta_p * q \rfloor$ of all q service executions. Typically we set also $\vartheta_p = 0.9$. Algorithm 6 for learning preconditions proceeds similarly to learning main effects by looking at all dimensions in $\mathbb{H}$ (Line 3). Nominal dimensions are handled analogously to Algorithm 5 (Line 4-8).

For numeric dimensions an additional step is performed to avoid too wide dimension intervals. Take for example a service for switching on a light which has been executed under varying conditions of the outside temperature *temp_outside*, such that the found interval is $s^{pre}[temp\_outside] = [-20, 50]$. This result has little informative value as a service precondition, since it merely describes that the service can be executed under any conditions for *temp_outside*. As preconditions we instead want specific conditions described by a restricted value interval. Let $\texttt{min}_i(H^{pre})$ be the global minimal value and $\texttt{max}_i(H^{pre})$ the global maximal value on dimension i in the precondition histories of any service. Then an interval on d_i is only considered a service precondition if it considerably restricts the dimension according to a parameter τ (Line 10-12). For example if $\tau = 0.5$, then only those intervals are considered that restrict the dimension to maximal half of its global size.

Algorithm 6 Learn service preconditions

procedure LEARNPRECONDITIONS($\mathbb{H} = [\mathbb{D}_0, \ldots, \mathbb{D}_{n-1}]$,$H_s^{pre} = [c_0^u, \ldots, c_{q-1}^u]$)
  $s^{pre} = \{\}$
  for $0 \leq i < n$ **do**
    if $\mathbb{D}_i$ is nominal **then**
      $v = \texttt{most\_common}_i(H_s^{pre})$
      if $\texttt{count}_i(H_s^{pre}|v) \geq \lfloor \vartheta_p * q \rfloor$ **then**
        $s^{pre}[i] = v$
      end if
    else
      if $\frac{\texttt{max}_i(H_s^{pre}) - \texttt{min}_i(H_s^{pre})}{\texttt{max}_i(H^{pre}) - \texttt{min}_i(H^{pre})} < \tau$ **then**
        $s^{pre}[i] = [\texttt{min}_i(H_s^{pre}), \texttt{max}_i(H_s^{pre})]$
      end if
    end if
  end for
  return s^{pre}
end procedure

Postprocessing

A final post-processing step must be performed after the preconditions of all services have been learned. It may be that the context contains static dimensions which never or rarely change during the whole execution history, e.g. the internal status of an unused device. These dimensions provide little informative value as preconditions and thus should be excluded. If a dimension is static, then references to it will be contained in the preconditions of a majority of services. Therefore it needs to be checked for each dimension i in the $\mathbb{H}$, whether more than φ service preconditions contain the same value or value interval for i. If this is the case, then i is removed from the preconditions of all services.

7.3.3 Learning side effects

Side effects of a service are any additional precondition and effect pairs that can be found in the execution history. For example, for the blinds service in Table 7.1, one could see the pair "If $c^u[light\_out] = bright$ and $c^u[light\_in] = dark$, then $\Delta c[light\_in] = bright$". To be considered a side effect, a precondition and effect pair should be contained in at least $\lfloor \vartheta_s * q \rfloor$ and at most $\lfloor \vartheta_e * q \rfloor$ executions. Setting ϑ_s to a very small value increases the risk that context changes from parallel service executions are found to be side effects. In this case the recommender system might recommend the wrong service. A high ϑ_s on the other hand means that very rare side effects can not be found, i.e. the recommender system will miss some recommendations. We consider the latter case to be preferable and typically set $\vartheta_s = 0.1$, i.e. only side effects occurring in at least 10% of service executions are considered.

Preprocessing

As a preprocessing step, it is necessary to remove from the histories any dimensions which were already used in precondition or main effect or were found to be static. For preconditions, if $s^{pre}[i] = v_1$, then there can be no side effect precondition $s^{spre}[i] = v_2$, since side effects only occur if s^{pre} and s^{spre} hold at the same time, which is not possible in this case. Therefore, for all dimensions which are set in s^{pre} or were found to be static, set all observations in H_s^{pre} to the missing value $\perp$. Analogously for the effect history, for all dimensions which are set in s^{eff}, set all observations in H_s^{eff} to the missing value $\perp$.

Finding candidate effects

It is much easier to find re-occurring patterns in the effect history than in the precondition history, since the former typically contains less data dimension. A pattern can also be called a cluster a, which is described by the indexes of the objects forming this cluster. The minimum number of objects in such a cluster must be $\lfloor \vartheta_s * q \rfloor$ for a service with q executions. For finding clusters we apply the DBSCAN clustering algorithm [36], which is one of the most commonly used clustering algorithms. Output of DBSCAN is a set of candidate effect clusters A, which will be the input to the next step.

Finding matching preconditions

Candidate clusters do not necessarily describe actual service side effects. It can also happen that artificial clusters are found which contain noise objects from simultaneous service executions, such as the TV effect seen in one execution of the example blind service. We consider a candidate cluster to describe a service side effect if a precondition can be found such that: if and only if the precondition is fulfilled, the side effect happens.

This requirement is very similar to the classification rules found by classification algorithms. Classification is a supervised data mining technique which is applied to an already classified set of training data to find rules for classifying unclassified data. For a given candidate cluster $a \in A$, we annotate each c_j^u in H_s^{pre} with a class attribute that indicates whether the item j is contained in a. Then classification learning can be applied to H_s^{pre} to find any rules (i.e. preconditions) for this class (i.e. effect).

Algorithm

Algorithm 7 for side effects learning takes as input the preprocessed precondition and effect histories of a service. In Line 3 effect clustering is performed, producing a list of candidate clusters that each contain at least $\lfloor \vartheta_s * q \rfloor$ objects. For each of those candidate clusters, each item Δc_j^u in the precondition history is annotated with a class *cluster* if j is contained in the cluster and a class *nocluster+j* otherwise (Line 5-9). A different class for each non-cluster item is used to avoid finding rules for *class=nocluster*.

A classifier is applied to the annotated data in line 10. We have found that the PART classification rule learner [39] achieves good results on our context data. The result of running PART is a set of rules, where each rule r is described by its condition, class and

Algorithm 7 Learn service side effects

1: **procedure** LEARNSIDEEFFECTS($H_s^{pre} = [c_0^u, \ldots, c_{q-1}^u], H_s^{eff} = [\Delta c_0, \ldots, \Delta c_{q-1}]$)
2: $s^{side} = \{\}$
3: $A = \texttt{cluster}(H_s^{eff}, \lfloor \vartheta_s * q \rfloor)$
4: **for** $\forall a \in A$ **do**
5: **for** $0 \leq j < q$ **do**
6: **if** $j \in a$ **then** $classes[i] = cluster$
7: **else** $classes[j] = nocluster+j$
8: **end if**
9: **end for**
10: $r = \texttt{best}(\texttt{PART}(H_s^{pre}, classes) | r.class = cluster)$
11: **if** $\frac{r.correct}{r.correct+r.incorrect} > \beta$ **then**
12: $s^{side} = s^{side} \cup \{(\texttt{rule\_to\_scope}(r) \rightarrow \texttt{cluster\_to\_scope}(a))\}$
13: **end if**
14: **end for**
15: **return** s^{side}
16: **end procedure**

accuracy in terms of correct and incorrect classifications. We are only interested in the one rule r with $r.class = \texttt{cluster}$ with the highest number of correct classifications. If this rule also has a low number of incorrect classifications according to a parameter β, then the pair $(r \rightarrow a)$ can be considered a side effect of the service (Line 11). Typically we set $\beta = 0.9$.

In Line 12 two functions `rule_to_scope` and `cluster_to_scope` are called to convert rules and clusters to context scopes. For converting a cluster a, create a new effect history $H_s'^{eff}$ that retains from H_s^{eff} only those changes Δc_j with $j \in a$. Then for each dimension $i \in H_s'^{eff}$, if i is numeric, set $s^{seff}[i] = [\texttt{min}_i(H_s'^{eff}), \texttt{max}_i(H_s'^{eff})]$. If i is nominal, set $s^{seff}[i] = \texttt{most\_common}_i(H_s'^{eff})$. Rules consist of a conjunction of several conditions. Each condition describes for one context dimension the value range for which this rule applies, either as an interval between a minimum and a maximum value (for numeric dimensions) or a set of values (for nominal dimensions). To convert a rule to a context scope, start from an empty context scope and set in this scope the respective value ranges for each of the conditions.

7.3.4 Incremental learning

All three algorithms lend themselves well to incremental learning. For small execution histories, the algorithms can be re-run whenever new observations are added to a service's execution history. As the execution histories grow over time, also the time necessary for learning the capabilities increases. If there is much activity in the smart home, it might then happen that capability learning is not finished before the next execution of a service. In

this case it is advisable to run the algorithms in an offline fashion, i.e. collect several new observations and execute the algorithms regularly (e.g. once a day).

7.4 Integrating the installation assistant into a smart home

In the following it is shown how the installation assistant can be integrated into a smart home. First the requirements are listed that must be fulfilled by smart home devices. Afterwards the system architecture used in the SM4All smart home prototype is shortly described.

7.4.1 Requirements for smart home devices

The installation assistant can only be used if smart home devices fulfill two requirements:

(i) Sensed data and the internal status of actuators must be made available to the system, to later form the user context. This can happen either using push mechanisms, i.e. actuators and sensors publish updates to the environment according to the publish/-subscribe paradigm, or alternatively they must offer a method for manually pulling current status and sensed data.

(ii) Actuators must for each of their services provide icons and human-readable service names to be displayed in the user interface

When looking at common smart home protocols one finds that:

UPnP directly fulfills both requirements. A device's internal status and sensed data is published using the GENA (General Event Notification Architecture) framework [21]. Service labels and icon URIs are specified in the UPnP device description.

KNX provides only the communication infrastructure to send control signals to devices and does not fulfill any of the requirements. In the SM4ALL smart home a component called the *Pervasive controller* maintains and publishes the status of all installed KNX devices and provides the necessary user interface information.

X10 devices can publish their internal state, but do not provide the necessary user interface information. Since the user interface information is static, it can be stored on a simple web server or alternatively be provided by the *Pervasive controller*

7.4.2 System architecture

The system architecture of the SM4All smart home is pictured in Figure 7.2. Components in the box *Smart home environment* were implemented by our partners in the SM4All project. The architecture supports KNX and UPnP devices. The *Pervasive controller* provides status and user interface information for the KNX devices and allows invoking services through a web service interface. Sensor and status data from the various sources is collected and processed by the *COPAL* (COntext Provisioning for ALl) component, which produces formatted context events.

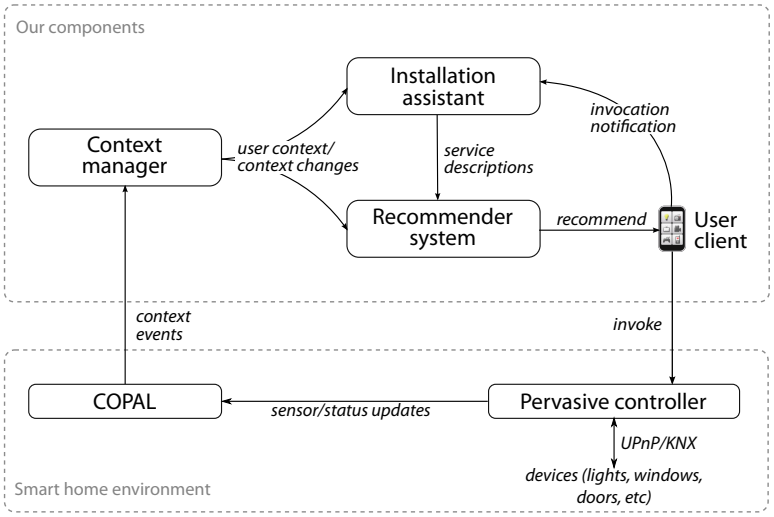


Figure 7.2: System architecture in the SM4All smart home

We added four additional components to the architecture:

- The *Context manager* creates the HAC and keeps track of the current user context and context changes. Dimensions are created automatically, based on the type of data that is being published. Context changes are broadcast by the *Context manager* as soon as new context information arrives.
- The *Recommender system* listens to the context changes and runs the service filtering and ranking as described in Chapters 4-6. In the prototype version that was installed in the smart home, only service filtering and recommendation based on user preferences was implemented. User preference learning and recommendation based on the user behavior was not implemented.
- The *Installation assistant* also listens to the context changes and runs the three capability learning algorithms that were presented in this chapter.
- The *User client* is implemented as a GUI application running on a smart phone. The GUI displays either the recommendations sent by the *Recommender system* or a list of all installed services. Users invoke a service by tapping on the respective icon.

Learning process

When the system is first installed, no information about service capabilities are available. Since service preconditions and effects are initialized as empty context scopes, no useful service filtering can be performed. In this situation, the set of recommended services equals the set of all installed services. Users need to scroll through the whole service list to select a service for invocation.

When a service is invoked through the GUI, an invocation notification is sent to the installation assistant. The installation assistant adds the current user context and any observed context changes to the service's execution history. The learning process is started when a minimum amount of observations have been collected for a service (we usually set this minimum support to ten observations). Learned service descriptions are sent to the recommender system and are directly used when calculating the recommendations. After a short while of using the system, the smart home inhabitant no longer has to select from the list of all services, but instead can rely on the generated recommendations.

7.5 Evaluation

7.5.1 Choice of datasets

A first set of experiments was performed in the SM4All smart home. The other smart home datasets can not be used, since no information about the latency of context dimensions is available. Since the SM4All prototype is built into a research facility without any access to the outside world, it was not possible to observe any side effects from outside factors such as day/night changes. Most of the services have thus rather simple capabilities referencing only one dimension. Only two service preconditions make use of more than one dimension: the window curtain in the living room can only be closed if the window is closed, vice versa the window can only be opened if the curtain is open. For this reason, additional experiments are performed using synthetically generated data.

7.5.2 Metrics

To evaluate the algorithms, first the actual service descriptions are prepared manually. Then let *actual* be a context scope that represents the actual main effect of a service. Let *learned* be the output of Algorithm 5 `LearnMainEffects` for this service, i.e. the learned main effect of the service. Since both context scopes describe a subspace in the multi-dimensional HAC, Equation 7.1 can be used to calculate the recall, i.e. how much of *actual* was correctly learned by the algorithm. Equation 7.2 calculates the precision analogously.

$$recall = \frac{\text{size of overlap between } actual \text{ and } learned}{\text{size of } \ actual} \tag{7.1}$$

$$precision = \frac{\text{size of overlap between } actual \text{ and } learned}{\text{size of } \ learned} \tag{7.2}$$

The overall precision/recall for learning service effects is the average of precision/recall over all services. F1 is the harmonic mean of precision and recall ($F1 = 2 * \frac{precision * recall}{precision + recall}$). These metrics are calculated analogously for Algorithm 6 `LearnPreconditions` and Algorithm 7 `LearnSideEffects`. A final metric is the number of executions that are necessary until the service capabilities have been learned correctly, i.e. F1$=1.0$ has been reached for preconditions, effects and side effects.

7.5.3 Experiments with the SM4All smart home

Initial tests in the smart home

We installed the components as described in Section 7.4.2 and started invoking services. After a short while, we could see that the system had begun learning the correct service descriptions. Using these automatically generated descriptions, service recommendation worked just as well as with manually provided service descriptions.

Due to a limited access to the prototype and the time-consuming nature of the experiments, we were only able to run this first test in the physical smart home. For further evaluating the system, we ran the following experiments in a simulation, using the same setup as the actual prototype, including all actuators, sensors and dimension latencies. The services were executed in a random fashion and each experiment was run for 10 replications.

Sequential service executions

We started by executing smart home services sequentially. Since dimension latencies range between 30 milliseconds (for lamps) to 21 seconds (for the motorized bed), we set the sampling time to 25 seconds. We kept pauses of at least 30 seconds between two service executions. Under these circumstances the system learns the correct service capabilities with on average 13.1 executions per service.

Overlapping service executions

In real-world scenarios it is quite unrealistic that all services are executed sequentially. In order to evaluate how the learning approaches can deal with overlapping executions of several services, the average time between service executions was gradually decreased. Table 7.4 lists the learning results for execution intervals ranging from 30 seconds (i.e sequential execution) to 1 second. In the latter case, 24 other services are executed during the sampling time of one service, creating heavily overlapping context changes. The number of necessary executions until the system correctly learns the capabilities are very stable for the 30, 10 and 5 seconds execution intervals. Only for the somewhat unrealistic case that the user continuously executes a service every second, the number of necessary execution increases slightly.

Average time between executions (in s)	30	10	5	1
Average necessary executions per service	13.1	13.8	14.4	20.8
(Standard deviation)	1.8	2.0	1.5	3.5
Worst case necessary executions	21	25	26.2	47.9

Table 7.4: Number of necessary executions in SM4All smart home

Table 7.4 also lists the worst case of necessary executions until the system correctly learns the capabilities. In these cases typically the effect was learned correctly, however the preconditions were temporarily too strict, e.g. the precondition for switching on the

bedroom lamp correctly referenced the status of the lamp, but also required that the TV should be off. This behavior can be avoided by increasing the minimum support necessary for starting the learning process. The higher the minimum support, the longer it takes until all service descriptions are learned. With a lower minimum support, service descriptions are learned faster but may, in rare cases, be temporarily too strict. This system parameter can be easily tweaked by the user to achieve a preferred behavior.

7.5.4 Experiments with synthetic data

The aim of the following experiments is to evaluate capability learning in an environment with a higher number of services and dimensions, plus more complicated preconditions, effects and side effects, as compared to the smaller-scale prototype system.

Experimental Setup

The simulation environment creates to this end 70 dimensions in a mix of 50% nominal and 50% numeric dimensions. The dimension latencies are distributed according to a normal distribution [2], with $\mu = \sigma^2 = 1$, i.e. most dimension values change between one or two seconds after service execution. 50 different services are created, each having main preconditions and effects, plus up to two side effects. For creating a service effect, first up to three dimensions are randomly drawn from the set of all generated context dimensions, and a random value or value interval is selected for each of them and set as a service effect. Generation of main precondition as well as side effects and their preconditions is performed analogously. The execution of services is simulated, with the time between two service executions drawn from an exponential distribution [3] with $\mu = execution\_interval$. All context changes up to the preset *sampling_time* are recorded. Capability learning was only applied to an execution history if at least ten observations had been made.

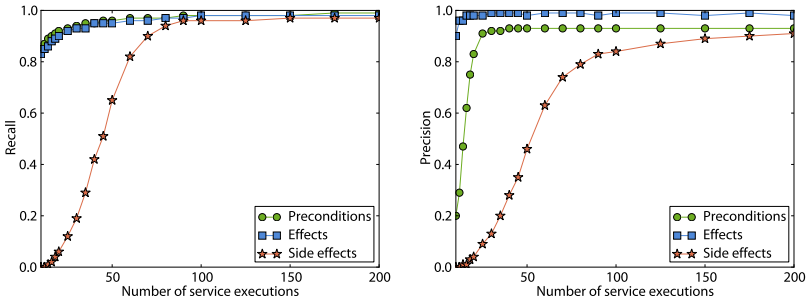
Number of service executions

The first experiment tests, how well the capability learning algorithm works depending on the number of executions per service. To test the algorithm without any simultaneous service executions, we set $sampling\_time = 10$ and $execution\_interval = 20$.

It can be seen in Figure 7.3 that main preconditions and effects of a service stabilize very quickly. After 25 executions of each service, preconditions and effects achieve precision>0.9 and recall>0.9. The learning of service effects stabilized slightly faster, which is not surprising since only a few context dimensions have to be mined, compared to all 70 dimensions for learning preconditions. For preconditions, the precision is slightly lower than the recall, i.e. the preconditions are recognized correctly, but may be too wide or include too many context dimensions. Side effects stabilize much slower, since they only

[2] The normal distribution was chosen to reflect that most dimensions have very similar latencies, but that there can also be outlier dimensions with much higher latencies (such as the motorized bed in the SM4All smart home)

[3] In modeling and simulation, the exponential distribution is a common choice for modeling the inter-arrival times of events.



(a) Recall (b) Precision

Figure 7.3: Number of service executions vs recall and precision

occur for some service executions, so more executions are necessary before side effects can be reliably mined. The results strongly improve between 20 and 100 executions per service and achieves high values (> 0.9) for precision and recall at around 175 executions.

Overlapping service executions

Next we stress tested the algorithm with overlapping service executions. We set the number of executions per service to 100, keep *sampling_time* $= 10$ and vary the *execution_interval*. For example for *execution_interval* $= 1$, service executions will happen on average every second, so for each of its executions a service will sample the context changes induced by itself and 9 other services. It can be seen in Figure 7.4 that the results for main preconditions and effects are very stable even for small execution intervals. The side effect accuracy is stable for two parallel service executions, but degrades for small execution intervals because context changes from overlapping executions are identified as side effects.

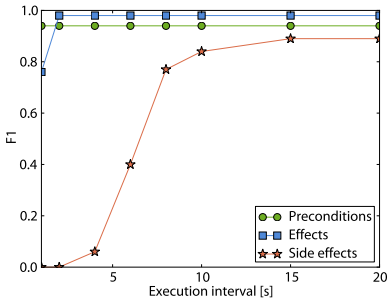


Figure 7.4: Size of execution interval vs F1

Varying dimension latencies

Finally, we evaluated how well the algorithm can deal with a wide spread of dimension latencies by using a *sampling_time* = 50 and gradually increasing the variance of the generated latencies $1 \leq \sigma^2 \leq 40$, with 100 executions per service. We found that the accuracy of the service capabilities is stable over the whole range of dimension latencies.

The experimental results show that our approach is able to reliably and quickly learn service main preconditions and effects even under very stressed conditions. Side effect learning is most reliable under normal, more relaxed conditions where individual services are executed mostly sequentially.

Runtime

Since capability learning can be performed offline, the algorithm runtime is less critical when compared to the recommender system. For this reason we did not perform an extensive runtime analysis. The experiments show that for 1000 executions, *sampling_time* = 10, and *execution_interval* = 20, capabilities can be learned in less than 2 seconds per service.

7.6 Summary

This chapter proposed an installation assistant for the smart home. It was demonstrated that descriptions of service capabilities can often not be provided by device manufacturers, which leads to complicated installation procedures that hinder customer acceptance. The chapter introduced the service execution history, which contains all data necessary for automatically learning service capabilities. We then proposed three algorithms for learning service main preconditions, effects and side effects from this history. We explained how we integrated the installation assistant into the SM4All smart home architecture and stated necessary requirements for device manufacturers. Evaluations with smart home and synthetic data show that the assistant can learn main preconditions and effects even under very stressed conditions, while side effect learning is most reliable when services are executed mostly sequentially.

Chapter 8

Related work

Chapter 2 introduced supporting technologies that are necessary for building context-aware smart home environments. This chapter complements this overview with a detailed inspection of research that is closely related to this dissertation. In particular, context modeling, service recommendation/service discovery, user behavior prediction and automatic service description are covered in this chapter.

8.1 Formal context model

Early context models

A variety of context models have been proposed since the term context-awareness was introduced in 1994 [95], a comprehensive survey can be found in [99]. Initial approaches to context-awareness defined context by example: in [94] context is the set of location, time and co-location, while in [96] context is divided into two categories "Human factors" (e.g. knowledge of user habits and emotional state, current tasks) and "Physical environment" (e.g. environmental conditions, location). In general the underlying model or data structures are either not described in these approaches or context is modeled as key-value pairs [95]. While the key-value structure is extensible, it does not provide the necessary operations for evaluating and comparing context descriptions.

A second group of approaches uses markup languages for describing user context. Context conditions in [12] for example are described using custom SGML (Standard Generalized Markup Language) tags for the individual context attributes. Again markup approaches are extensible but do not lend themselves for evaluating context descriptions. This limitation also applies to graphical context modeling using tools such as Object-Role Modeling [50].

Ontological models

In recent years, ontologies have become popular for modeling context. Strang et al. propose the CoOL ontology that describes context with three concepts: "aspect", "scale", and

"context information" [100]. Aspect is the context attribute itself, e.g. temperature. Each aspect can be expressed in different scales, for example the Fahrenheit and Celsius scales for the temperature. The context information is the actual value of the context attribute in a given scale. This model is very flexible and it is easy to define new context attributes. However, it has the same limitations as the original key-value structures.

Wang et al. choose a very different ontological model of context, explicitly defining context attributes such as activity, location, and device status [108]. For creating new context attributes, Wang's ontology has to be extended with appropriate concepts and their relationships. This is not practical when automatically generating new context attributes, which is necessary for realizing a smart installation assistant. Furthermore, concepts such as context change can not be expressed with existing context ontologies.

Context as a multi-dimensional space

Meolucci defines context for use in information retrieval [73] with the aim of improving and personalizing the retrieval of documents. For example, if a user has been repeatedly issuing queries containing the term "computer", the retrieval system automatically takes this term into consideration for the next queries. Meolucci models context as a high-dimensional vector-space and uses linear algebra for processing context information and describing context changes. Since this model is situated in a different domain, concepts particular to pervasive environments such as service capabilities are not considered.

In our domain, Padowitz and Locke et al. propose a usage of space theory for situation reasoning [80]. In this work concepts are generally treated as non-numeric enumerates so that no comparison is applicable, thereby excluding a large spectrum of context information.

HAC was inspired by Hyperspace Analogue to Language (HAL) [66] and the context space of assumptions [58], with both approaches originating from Artificial Intelligence. HAL is used for understanding natural language and measuring the difference between statements, while the context space of assumptions aims to analyze interpretations of assumptions within different communication contexts, something completely different to the meaning of context in pervasive environments.

Comparison with our work

Other works in context-awareness fail to identify the inherent connection between user context and services: service preconditions describe the context conditions in which a service can be executed and the effect determines the context changes when executing the service. HAC recognizes this connection by expressing service preconditions, effect, and side effects in terms of a multi-dimensional context space. Within this context space, a number of operations are defined that can be used to check if service preconditions are fulfilled, to calculate the overall effect of a service, and to determine how useful a service would be in the given situation. These operations are not supported by existing context models, but are crucial for the approaches presented in this dissertation.

8.2 Smart home recommender system

Mobile recommender systems

The proposed recommender system is somewhat related to mobile recommender systems. Ricci gives an overview of the field and identifies a number of common applications [88]: provide mobile users with recommendations of nearby restaurants, hotels and tourist attractions, give route recommendations, and recommend news and multimedia content. Basis for the recommendations are typically user preferences and ratings from other users.

Any useful mobile recommender systems must be context-aware, and in particular awareness for user location is a key feature. However, the generated recommendations are very different in nature from our smart home recommender system and the systems can not be easily adapted to recommend context-altering services.

Service discovery in ubiquitous computing

Within pervasive and ubiquitous computing, related work is published under the term service discovery. The aim of service discovery is to find from a list of services the one that fulfills some specific functionality. A large majority of works in the field concentrate on explicit service discovery, i.e. the user has to specify the desired properties of the service and send a discovery request to the system. Early approaches to explicit service discovery concentrated on high-level search terms such as service name and category; an overview of these earlier approaches can be found in [120].

Later approaches also took user context and Quality of Service (QoS) into account. One of the most prominent methods in this category is the EASY (Efficient semAntic Service discoverY) framework [76]. With EASY, services can be semantically annotated with additional information on service inputs and outputs, relevant context information and QoS descriptions. Semantic matching is performed between a discovery request and the available services to find a suitable service.

Proactive service discovery

However, users still must manually specify desired functionalities and send a discovery request to the system. In the smart home, such an approach would seriously decrease usability, in fact user interfaces might end up being even more complicated than they are today. Very few works have considered implicit or proactive service discovery, where relevant services are automatically pushed to the user interface. Within mobile recommender systems, Woerndl et al. recently proposed a system that automatically informs a driver when a gas station is nearby, if it detects that the car's fuel tank is nearly empty [110]. Several other proactive mobile recommender systems are referenced in [88]. Within ubiquitous computing, Hesselman et al. propose persistent discovery requests: users only have to specify such a request once and receive updates whenever new services become available that match the request [52].

At the extreme opposite of explicit service discovery we can find a few approaches for an autonomously acting home ([78, 32, 104], see also Section 2.3.3). No manual discovery requests are necessary, since the smart home automatically identifies relevant services. However, since the home executes services without being prompted by the user, there is a high risk that these methods elicit feelings of loss of control and end up being rejected by users.

Comparison with our work

The proposed smart home recommender system is in a unique position on the spectrum between explicit service discovery and the autonomously acting home. Users do not have to manually perform explicit service discovery, since our system automatically identifies services that match the user context, preferences, and habits. However, instead of autonomously executing the highest rated services, the discovery results are made available as service recommendations. This allows users to select a utilization strategies for the recommendations that matches their requirements for convenience and control.

8.3 Behavior prediction

Location prediction

Mozer [78] built a neural network that predicts which zone in the smart home will become occupied within the next two seconds. The neural network bases its decision on information coming from motion detectors, door status sensors, sound sensors, information about recent locations, and the time of day. The predictions are used to switch on the lights in the to-be-occupied zones before the user enters them. The author reports promising initial results. However, the approach is tailored to one specific home installation and supports only location prediction. It remains uncertain how well the approach is transferable to other smart homes and whether the neural network can scale for predicting further attributes of user context.

Prediction of user commands and device interactions

Das et al. propose the Smart Home Inhabitant Prediction (SHIP) algorithm for predicting the next action of a user [26]. Given a user's most recent commands, the algorithm identifies matching sequences from the collected history and uses them to predict the next command. Temporal relations between user actions are not considered. To deal with small variations in user routines, the user can set a parameter called inexact threshold that represents the maximum percentage of allowed mismatches; however, no evaluation of the effectiveness of this parameter was performed. In contrast, our algorithm does not require users to perform their actions in regular sequences. Additionally, the SHIP algorithm learns only from the commands that are issued to the home devices, i.e. it learns a history of service invocations. Our method works at the context level and can therefore learn user behavior from manually performed actions as well as from service invocations.

In the same article, Das et al. also propose the LeZiUpdate algorithm for user location prediction [26]. Gopalratnam and Cook propose an improved version called ActiveLeZi and apply it for predicting arbitrary inhabitant–home interactions, i.e. they predict which device the user will interact with next [44].

Both LeZiUpdate and ActiveLeZi build a tree of observed sequences of these interactions; ActiveLeZi converges faster when building this tree, which was an issue in LeZiUpdate. To predict the next action, the algorithm tries to match the most recent inhabitant–home interactions with previously observed sequences. The algorithm was tested in a dataset collected in a test smart home and achieved a prediction accuracy of 47%[1].

Gopalratnam and Cook also address the issue of predicting when the next device interaction will happen [44]. They model the time interval between a sequence and the next event as a normal distribution and learn the mean and standard deviation of this distribution. However, this method is not described in detail and was evaluated only on a synthetic dataset.

Prediction of sensor events

Aipperspach et al. [2] propose an approach for predicting arbitrary sensor events [2]. Their approach is also related to sequence matching: sensor events are modeled as words and tools from Natural Language Processing are used to build a language model of the sensor data. This model can calculate how probable some word is, given the most recent words. In case the sequence of most recent words has not been seen during training, the prediction algorithm falls back to use a shorter subsequence or to using the overall probability of seeing the word. Longer pauses in the sequence of sensor events are modeled as a special PAUSE event, while other temporal relations are not considered. The authors report 51% prediction accuracy on a smart home dataset[2]. They briefly discuss how the predictions can be used, e.g. to predict where the user is heading and turn on the lights in preparation. However, no generalized approach for making use of the predictions is proposed.

Usage of temporal information in activity recognition

Some recent works in activity recognition also make use of temporal information to improve recognition accuracy. Ye et al propose the usage of absolute temporal information as well as relative temporal information for activity recognition [117] and show that temporal awareness can significantly increase the activity recognition accuracy. McKeever et al. propose the usage of activity durations in activity recognition [72] and implement a learning algorithm using Dempster-Shafer theory. Evaluation results show a 70% improvement of the recognition f-measure, compared to a non-temporal Naïve Bayes classifier. Both works perform supervised learning, while our method performs unsupervised learning.

[1] For a different dataset than the ones used in this dissertation.

[2] Again, a different dataset was used.

Comparison with our work

Our proposed algorithm has several advantages compared to existing approaches. First, it does not rely on users executing their activities in orderly sequences. Instead we predict the next event based on the current user context. Second, our approach also takes temporal relations between events into account. Third, instead of learning from user commands, our method works at the context level and can learn user behavior from manually performed actions as well as from service invocations. Finally, we also propose a method for transforming the predictions into personalized service recommendations.

8.4 Installation assistant

Tools providing graphical assistance

Compared to the number of proposed service description languages, research on automatic or semi-automatic service description is sparse. Patil et al. were among the first to point out the problems of manual annotation [82]. They present the METEOR-S web service annotation framework, which graphically assists the user in annotating web service descriptions. METEOR-S uses linguistic and structural matching between a service's functional description and candidate ontologies to identify the most fitting concepts for describing a service semantically and suggests them to the user during the annotation process. ASSAM is a similar tool by Heß et al., with the difference that ASSAM employs machine-learning techniques to identify suitable concepts by learning from already annotated services [51].

Inference of service descriptions from workflows

Bowers et al. exploit the additional knowledge contained in scientific workflows for annotation purposes [11]. Scientific workflow systems aim to integrate pre-processing of data, the statistical, and data mining processes on the data, and the post-processing and visualization of the results. An actor in such a workflow is any component (e.g. shell script, web service) that performs work on the data. Actors are annotated with information about the process they perform for facilitating automatic composition of such workflows. Bowers et al. propose to automatically infer missing annotations based on the connections between actors, e.g. if it is known that an actor produces descriptions of genes, then it can be inferred that any other actor that takes the descriptions as input performs work on such descriptions. A similar approach is taken in [6] for finding annotations for inputs and outputs of web services in a service workflow.

Execute and observe

Some works propose to execute services to learn about their functionalities. Lerman at al. use machine learning techniques to identify potential input data types for a service and validate them by executing the service with sample data and observing whether the output is satisfying or erroneous [59]. Carman at al. aim to find out how the functionality of a service can be described in terms of other, already annotated services [15]. The target service and

varying combinations of existing services are executed using the same input data and the similarity of the output data is checked until a satisfying combination is found.

Comparison with our work

Our work distinguishes itself substantially from the described approaches. Previous works concentrate on finding annotations for service category and input and output data types. As far as we know, no work has been published concerning the automatic annotation of context-altering services, which are common in smart home environments. Additionally, all other approaches rely on specific information which may itself be hard to provide: typically either high-quality functional service descriptions and ontologies or partially annotated services and workflows are needed. In contrast, our approach relies solely on observing the environment. Only the data published by actuators and sensors is needed, which is a functionality already built into popular technologies such as UPnP and X10.

Chapter 9

Conclusions and future work

9.1 Summary

This dissertation focused on a number of usability concerns that currently prevent smart homes from reaching a broad audience:

- Smart home inhabitants frequently complain about the inflexibilities of manually programmed rules and scenes.
- Inhabitants struggle with complicated user interfaces that are often unusable for guests. Often the interface introduces so much overhead that it is easier to just execute the desired action manually.
- Installation of smart home devices is difficult; today's smart homes are either built by an inhabitant with DIY experience or by a technician.
- Many potential customers reject the idea of living in a smarter home because they are not comfortable with the idea that the system autonomously performs some action on their behalf.

The dissertation reduces these obstacles by proposing two smart assistants – a recommender system and an installation assistant for smart homes:

- Both assistant are built on a formal context model called HAC that defines user context, context changes, and context-altering services in the same multi-dimensional space and provides several operations for reasoning about context situations.
- The recommender system **reduces the inflexibility and complexity of smart home control**. It continuously interprets the user context and recommends beneficial services based on user habits and preferences. With the generated service recommendations it becomes possible to build adaptive smart home user interfaces that can offer varying levels of convenience and control. The recommender system can fulfill important performance requirements even in large smart home installation.

- The recommender system offers two approaches for service ranking. The first approach aims to **emulate typical user behavior** by building a model of the user habits. The learning is unsupervised, i.e. no manual input from the inhabitant is required. During service recommendation, the algorithm identifies the most probable next user actions (in form of the most probable context changes) and tries to find services that could aid the user with performing these actions. Extensive evaluations show that for two smart home datasets the correct service is included in the top five recommendations in 90% of all cases and that the algorithm is stable with regard to the choice of parameters, as long as extreme values are avoided.
- The second ranking approach focuses on **fulfilling the user's preferences**. The proposed algorithm calculates the distance of the user to a preferred situation and estimates the impact of executing a service with respect to this distance. Several criteria for using these measures when ranking services were outlined. Preliminary evaluations of this second ranking strategy show promising results.
- The installation assistant **simplifies installation and configuration** of the smart home recommender system. The service descriptions generated by the assistant can also be interesting for other tasks, such as service composition. The installation assistant makes use of a history of service executions to automatically learn the service preconditions, effects, and side effects. It can be easily integrated into existing smart home architectures and it has minimal requirements for device manufacturers, which are already fulfilled or partially fulfilled by some popular smart home protocols. The installation assistant is one of the very few works that addresses two important challenges for the smart home (the accidental smart home, no system administrator [33]) that must be solved before smart home technology can be ready for the mass market.

9.2 Limitations

We have identified some limitations of the proposed smart assistants:

- Both assistants work best if a variety of sensors and actuators are deployed in the smart home. There is not much use for the recommender system if only one or two actuators are installed. The installation assistant can not learn about service preconditions, effects, and side effects if necessary sensor data is not available. For example, if a window is furnished with a child-proof lock that is unknown to the system, then learned preconditions will not reflect the necessity to open the lock to open the window.
- Connected to this concern is the issue of the reliability of the smart home. If the recommender system bases decisions on incomplete or faulty sensor data, it might recommend services that are not actually available (the window is locked) or are not useful (it is not actually too dark in the house, the luminosity sensor reported wrong values). When using active-context awareness, this misinformation could lead to critical situations, e.g. the home autonomously decides to open a locked window, damaging the window in the process. These issues are much less severe with passive

context-awareness, since ultimately it is the user who decides which service is to be executed. We want to stress again our belief that active context-awareness is not yet a viable option for the smart home.

- The recommender system can only be used in single-person households because recommendations are personalized to one inhabitant's context, habits, and preferences. Some ideas for addressing this limitation are discussed in the next section.
- The installation assistant does not address any low-level integration issues between different smart home technologies. However, it poses only minimal requirements on device manufacturers and some standard smart home protocols already fulfill these requirements.
- The context model is not suitable for expressing relationships such as: if it is colder outside than inside, then opening the windows will make it colder inside. Currently such relationships need to be modeled by setting appropriate temperature ranges for inside and outside temperature, which is much less expressive.

9.3 Future work

There are a number of potential directions for future work:

- We would like to perform further work with the different strategies for utilizing service recommendations. In particular the mixed strategy of passive plus active context-awareness seems very promising, since it offers a very good compromise between convenience and control. Utilization of conflict and uncertainty when building the user interface should also be further explored, given the encouraging initial evaluation results. It will be necessary to build prototypes of different interfaces and perform user acceptance tests. In particular with GUIs we see the risk that users might become irritated when the list of recommendations changes too often. Further work is necessary to figure out how alleviate this risk.
- The recommender system should be adapted so that it can also be used in multi-person households. For this to be feasible, a number of challenges must resolved: *(i)* The algorithm for learning user habits must collect individual histories for each inhabitant. This means it must be able to distinguish which user is performing which action. Either inhabitants are only allowed to interact with the home through the smart home interface (instead of also being able to perform the actions manually, severely decreases usability) or must be tracked using an indoor positioning system (which do not yet offer the necessary resolutions in larger environments). *(ii)* Related to the previous issue, the recommender system should not send recommendations to inhabitant1 that only interest inhabitant2 (e.g. inhabitant1 is in the bedroom, inhabitant2 is in the kitchen, both receive kitchen recommendations, since the smart home detected recent activity in the kitchen). *(iii)* The smart home must be able to identify conflicts between the recommendations for several inhabitants, in particular when using active context-awareness.

- Further work is necessary to increase the maturity level of ranking by user preferences. In particular, an extended evaluation of this ranking approach using suitable smart home data is necessary. Automatic learning of user preferences should be explored further and must also be evaluated using actual smart home data.
- At the moment we have two strategies for service ranking that address slightly different needs. Ranking based on user habits is well suited to capture dynamic behavior in the home but can not detect issues such as a too low room temperature. Ranking based on user preferences is more appropriate for such tasks, but has only limited use in dynamic situations. At present, we have identified two possibilities for integrating these two strategies to produce one global ranking: *(i)* Each strategy calculates a normalized score (e.g. between 0.0 and 1.0) that is computed into an overall score using some combination function (average, sum, etc). *(ii)* Define only those situations as user preferences that cannot be addressed by ranking based on user habits (e.g. preferred environmental conditions). When these preferences are not met, the results from preference ranking are prioritized.

Bibliography

[1] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: a mobile context-aware tour guide. *Wirel. Netw.*, 3:421–433, 1997.

[2] R. Aipperspach, E. Cohen, and J. F. Canny. Modeling human behavior from simple sensors in the home. In *Proceedings of the Fourth International Conference on Pervasive Computing*, pages 337–348. Springer Berlin/Heidelberg, 2006.

[3] M. R. Alam, M. B. I. Reaz, and M. A. M. Ali. A review of smart homes – past, present, and future. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(6):1190 –1203, 2012.

[4] P. Bahl and V. Padmanabhan. RADAR: an in-building RF-based user location and tracking system. In *Proceedings of INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 2, pages 775–784. IEEE Computer Society, Washington, DC, USA, 2000.

[5] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2:263–277, 2007.

[6] K. Belhajjame, S. M. Embury, N. W. Paton, R. Stevens, and C. A. Goble. Automatic annotation of web services based on workflow definitions. *ACM Trans. Web*, 2:11:1–11:34, 2008.

[7] V. Bellotti and A. Sellen. Design for privacy in ubiquitous computing environments. In *Proceedings of the Third European Conference on Computer-Supported Cooperative Work*, ECSCW'93, pages 77–92. Kluwer Academic Publishers Norwell, MA, USA, 1993.

[8] S. Ben Allouch, J. Dijk, and O. Peters. The acceptance of domestic ambient intelligence appliances by prospective users. In *Proceedings of the Seventh International Conference on Pervasive Computing*, pages 77–94. Springer Berlin/Heidelberg, 2009.

[9] L. Bonanni. Living with hyper-reality. In Y. Cai and J. Abascal, editors, *Ambient Intelligence in Everyday Life*, pages 130–141. Springer Berlin/Heidelberg, 2006.

[10] D. Bonino and F. Corno. What would you ask to your home if it were intelligent? Exploring user expectations about next-generation homes. *JAISE*, 3(2):111–126, 2011.

[11] S. Bowers and B. Ludäscher. Towards automatic generation of semantic types in scientific workflows. In *Proceedings of Web Information Systems Engineering Workshops*, pages 207–216. Springer Berlin/Heidelberg, 2005.

[12] P. J. Brown. The stick-e document: a framework for creating context-aware applications. In *Proceedings of the Electronic Publishing Conference*, pages 259–272, 1996.

[13] A. B. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu, and C. Dixon. Home automation in the wild: challenges and opportunities. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2115–2124. ACM New York, USA, 2011.

[14] H. Brynjarsdottir, M. Håkansson, J. Pierce, E. Baumer, C. DiSalvo, and P. Sengers. Sustainably unpersuaded: how persuasion narrows our vision of sustainability. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, CHI '12, pages 947–956. ACM New York, USA, 2012.

[15] M. J. Carman and C. A. Knoblock. Learning semantic descriptions of web information sources. In *Proceedings of the 20th International Joint Conference on Artifical intelligence*, pages 2695–2700. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[16] M. Chan, D. Estève, C. Escriba, and E. Campo. A review of smart homes – present state and future challenges. *Comput. Methods Prog. Biomed.*, 91(1):55–81, 2008.

[17] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41:1–58, 2009.

[18] G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical report, Dartmouth College, Hanover, NH, USA, 2000.

[19] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou. Developing a context-aware electronic tourist guide: some issues and experiences. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, CHI '00, pages 17–24. ACM New York, USA, 2000.

[20] S.-L. Chua. *Behaviour Recognition in Smart Homes*. PhD thesis, Massey University, Manawatu, New Zealand, 2012.

[21] J. Cohen, S. Aggarwal, and Y. Goland. General event notification architecture base draft. Technical report, 1999.

[22] D. J. Cook, J. C. Augusto, and V. R. Jakkula. Review: Ambient intelligence: Technologies, applications, and opportunities. *Pervasive Mob. Comput.*, 5(4):277–298, 2009.

[23] K. Curran, E. Furey, T. Lunney, J. Santos, D. Woods, and A. McCaughey. An evaluation of indoor location determination technologies. *J. Locat. Based Serv.*, 5(2):61–78, 2011.

[24] J. R. Dabrowski and E. V. Munson. Is 100 milliseconds too fast? In *Extended Abstracts on Human Factors in Computing Systems*, CHI EA '01, pages 317–318. ACM New York, USA, 2001.

[25] S. Darby. The effectiveness of feedback on energy consumption. Technical report, Environmental Change Institute, University of Oxford, 2006.

[26] S. Das, D. Cook, A. Battacharya, E. Heierman III, and T. Lin. The role of prediction algorithms in the MavHome smart home architecture. *Wireless Communications, IEEE*, 9(6):77–84, 2002.

[27] S. Davidoff, M. Lee, C. Yiu, J. Zimmerman, and A. K. Dey. Principles of smart home control. In *Proceedings of the Sixth International Conference on Ubiquitous Computing*, pages 19–34. Springer Berlin/Heidelberg, 2006.

[28] T. Denœux. The cautious rule of combination for belief functions and some extensions. In *Proceedings of Ninth International Conference on Information Fusion*, pages 1 –8, 2006.

[29] G. Dewsbury. The social and psychological aspects of smart home technology within the care sector. *New Technology in the Human Services*, 14(1 &2):9 – 17.

[30] G. A. Dewsbury and H. M. Edge. Designing the home to meet the needs of tomorrow ... today: Smart technology, health and well-being. In *Annual Conference of the College of Occupational Therapists (Specialist Housing Section)*, pages 256–272. Springer, 2001.

[31] A. K. Dey, R. Hamid, C. Beckmann, I. Li, and D. Hsu. aCAPpella: programming by demonstration of context-aware applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 33–40. ACM New York, USA, 2004.

[32] F. Doctor, H. Hagras, and V. Callaghan. A fuzzy embedded agent-based approach for realizing ambient intelligence in intelligent inhabited environments. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 35(1):55–65, 2005.

[33] W. K. Edwards and R. E. Grinter. At home with ubiquitous computing: Seven challenges. In *Proceedings of the Third International Conference on Ubiquitous Computing*, pages 256–272. Springer London, UK, 2001.

[34] B. Eggen, G. Hollemans, and R. van de Sluis. Exploring and enhancing the home experience. *Cognition, Technology & Work*, 5(1):44–54, 2003.

[35] U. Esnaola and T. Smithers. Whistling to machines. In Y. Cai and J. Abascal, editors, *Ambient Intelligence in Everyday Life*, pages 198–226. Springer, 2006.

[36] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996.

[37] A. Ferscha, C. Holzmann, M. Hechinger, B. Emsenhuber, S. Resmerita, S. Vogl, and B. Wally. Pervasive computing. In *Hagenberg Research*, volume 1, pages 379–431. Springer Berlin/Heidelberg, 2009.

[38] D. Foster, M. Blythe, P. Cairns, and S. Lawson. Competitive carbon counting: can social networking sites make saving energy more enjoyable? In *Extended Abstracts on Human Factors in Computing Systems*, CHI EA '10, pages 4039–4044. ACM New York, USA, 2010.

[39] E. Frank and I. H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the 15th International Conference on Machine Learning*, pages 144–151. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[40] Z. Gabriel and A. Bowling. Quality of life from the perspectives of older people. *Ageing and Society*, 24:675–691, 8 2004.

[41] K. Gajos, H. Fox, and H. Shrobe. End user empowerment in human centered pervasive computing. In *Proceedings of the International Conference on Pervasive Computing*, pages 1–7. Springer London, UK, 2002.

[42] M. Gallissot, J. Caelen, N. Bonnefond, B. Meillon, and S. Pons. Using the Multicom Domus Dataset. Research Report RR-LIG-020, LIG, Grenoble, France, 2011. `http://domus.imag.fr/datasets/`, accessed 1. July 2013.

[43] D. Gann, J. Barlow, and T. Venables. Digital futures: making homes smarter. Technical report, Published for the Joseph Rowntree Foundation by the Chartered Institute of Housing, 1999.

[44] K. Gopalratnam and D. J. Cook. Online sequential prediction via incremental parsing: The Active LeZi algorithm. *IEEE Intelligent Systems*, 22(1):52–58, 2007.

[45] Y. Gu, A. Lo, and I. Niemegeers. A survey of indoor positioning systems for wireless personal networks. *Communications Surveys Tutorials, IEEE*, 11(1):13–32, 2009.

[46] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, 2000.

[47] R. Harper, M. Lamming, and W. Newman. Locating systems at work: implications for the development of active badge applications. *Interacting with Computers*, 4(3):343 – 363, 1992.

[48] A. Hein and T. Kirste. Towards recognizing abstract activities: An unsupervised approach. In *Proceedings of the Second Workshop on Behaviour Monitoring and Interpretation BMI'08*, pages 102–114, 2008.

[49] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen. The Gator Tech Smart House: a programmable pervasive space. *Computer*, 38(3):50–60, 2005.

[50] K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling context information in pervasive computing systems. In *Proceedings of the First International Conference on Pervasive Computing*, pages 167–180. Springer London, UK, 2002.

[51] A. Heß, E. Johnston, and N. Kushmerick. ASSAM: A tool for semi-automatically annotating semantic web services. In *Proceedings of the International Semantic Web Conference*, pages 320–334. Springer Berlin/Heidelberg, 2004.

[52] C. Hesselman, A. Tokmakoff, P. Pawar, and S. Iacob. Discovery and composition of services for context-aware systems. In *Proceedings of the First European Conference on Smart Sensing and Context*, pages 67–81. Springer Berlin/Heidelberg, 2006.

[53] J. Humble, A. Crabtree, T. Hemmings, K.-P. Åkesson, B. Koleva, T. Rodden, and P. Hansson. "Playing with the Bits" user-configuration of ubiquitous domestic environments. In *Proceedings of the Fifth International Conference on Ubiquitous Computing*, pages 256–263. Springer Berlin/Heidelberg, 2003.

[54] S. S. Intille, K. Larson, E. M. Tapia, J. S. Beaudin, P. Kaushik, J. Nawyn, and R. Rockinson. Using a live-in laboratory for ubiquitous computing research. In *Proceedings of the Fourth International Conference on Pervasive Computing*, pages 349–365. Springer Berlin/Heidelberg, 2006.

[55] ISO 29341-1:2008. Part 1: Upnp device architecture version 1.0. Technical report, 2008. `http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0-20081015.pdf`, accessed 15. June 2013.

[56] E. Kaldeli, E. U. Warriach, J. Bresser, A. Lazovik, and M. Aiello. Interoperation, composition and simulation of services at home. In *Eighth International Conference on Service Oriented Computing*, pages 167–181. Springer Berlin/Heidelberg, 2010.

[57] K. Kappel and T. Grechenig. "show-me": water consumption at a glance to promote water conservation in the shower. In *Proceedings of the Fourth International Conference on Persuasive Technology*, pages 26:1–26:6. ACM New York, USA, 2009.

[58] D. Lenat. The Dimensions of Context-Space. Technical report, CYCORP, 1998.

[59] K. Lerman, A. Plangprasopchok, and C. A. Knoblock. Automatically labeling the inputs and outputs of web services. In *Proceedings of the 21st National Conference on Artificial Intelligence*, AAAI'06, pages 1363–1368. AAAI Press, 2006.

[60] F. Li, K. Rasch, S. Sehic, R. Ayani, and S. Dustdar. Unsupervised context-aware user preference mining. In *Workshop on Activity Context-aware Systems at Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI-13)*, 2013, to appear.

[61] F. Li, K. Rasch, H.-L. Truong, R. Ayani, and S. Dustdar. Proactive service discovery in pervasive environments. In *Proceedings of the Seventh International Conference on Pervasive Services*, pages 126–133, 2010.

[62] Y. Li and L. E. P. Parker. Classification with missing data in a wireless sensor network. In *Proceedings of Southeastcon 2008*, pages 533–538. IEEE Computer Society, Washington, DC, USA, 2008.

[63] B. Logan, J. Healey, M. Philipose, E. M. Tapia, and S. Intille. A long-term evaluation of sensing modalities for activity recognition. In *Proceedings of the Ninth International Conference on Ubiquitous computing*, pages 483–500. Springer Berlin/Heidelberg, 2007.

[64] G. Low and A. E. Molzahn. Predictors of quality of life in old age: A cross-validation study. *Research in Nursing & Health*, 30(2):141–150, 2007.

[65] J. Lu, T. Sookoor, V. Srinivasan, G. Gao, B. Holben, J. Stankovic, E. Field, and K. Whitehouse. The smart thermostat: using occupancy sensors to save energy in homes. In *Proceedings of the Eighth ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 211–224. ACM New York, USA, 2010.

[66] K. Lund and C. Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods Instruments and Computers*, 28(2):203–208, 1996.

[67] R. Luo, C.-C. Yih, and K.-L. Su. Multisensor fusion and integration: approaches, applications, and future research directions. *Sensors Journal, IEEE*, 2(2):107–119, 2002.

[68] D. Lupton and W. Seymour. Technology, selfhood and physical disability. *Social Science & Medicine*, 50(12):1851 – 1862, 2000.

[69] G. Marx. Murky conceptual waters: The public and the private. *Ethics and Information Technology*, 3(3):157–169, 2001.

[70] U. Maurer, A. Rowe, A. Smailagic, and D. P. Siewiorek. eWatch: A wearable sensor and notification platform. In *Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks*, pages 142–145, 2006.

[71] R. M. Mayrhofer. *An Architecture for Context Prediction*. PhD thesis, Johannes Kepler Universität Linz, Austria, 2004.

[72] S. McKeever, J. Ye, L. Coyle, C. Bleakley, and S. Dobson. Activity recognition using temporal evidence theory. *J. Ambient Intell. Smart Environ.*, 2(3):253–269, 2010.

[73] M. Melucci. Context modeling and discovery using vector space bases. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, pages 808–815. ACM New York, USA, 2005.

[74] S. Mennicken and E. Huang. Hacking the natural habitat: An in-the-wild study of smart homes, their development, and the people who live in them. In *Proceedings of the Tenth International Conference on Pervasive Computing*, pages 143–160. Springer Berlin/Heidelberg, 2012.

[75] S. Meyer and A. Rakotonirainy. A survey of research on context-aware homes. In *Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003 - Volume 21*, pages 159–168. Australian Computer Society, Inc., 2003.

[76] S. B. Mokhtar, D. Preuveneers, N. Georgantas, V. Issarny, and Y. Berbers. EASY: Efficient semantic service discovery in pervasive computing environments with QoS and context support. *The Journal of Systems & Software*, 81(5):785–808, 2008.

[77] M. C. Mozer. The neural network house: An environment that adapts to its inhabitants. In *Proceedings of the American Association for Artificial Intelligence Spring Symposium on Intelligent Environments*, pages 110–114. AAAI Press, 1998.

[78] M. C. Mozer. Lessons from an adaptive home. *Smart Environments: Technologies, Protocols, and Applications*, pages 271–294, 2005.

[79] E. D. Mynatt, I. Essa, and W. Rogers. Increasing the opportunities for aging in place. In *Proceedings on the 2000 Conference on Universal Usability*, pages 65–71. ACM New York, USA, 2000.

[80] A. Padovitz, S. Loke, and A. Zaslavsky. Towards a theory of context spaces. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, pages 38–42. IEEE Computer Society, Washington, DC, USA, 2004.

[81] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: a review. *SIGKDD Explor. Newsl.*, 6(1):90–105, 2004.

[82] A. A. Patil, S. A. Oundhakar, A. P. Sheth, and K. Verma. METEOR-S web service annotation framework. In *Proceedings of the 13th International Conference on the World Wide Web*, pages 553–562. ACM New York, USA, 2004.

[83] A. Pentland. Perceptual environments. In *Smart Environments*, pages 345–359. John Wiley & Sons, Inc., 2005.

[84] D. Petersen, J. Steele, and J. Wilkerson. Wattbot: a residential electricity monitoring and feedback system. In *Extended Abstracts on Human Factors in Computing Systems*, CHI EA '09, pages 2847–2852. ACM New York, USA, 2009.

[85] D. Randall. Living inside a smart home: A case study. In R. Harper, editor, *Inside the Smart Home*, pages 227–246. Springer London, 2003.

[86] K. Rasch, F. Li, S. Sehic, R. Ayani, and S. Dustdar. Context-driven personalized service discovery in pervasive environments. *World Wide Web*, 14:295–319, 2011.

[87] K. Rasch, F. Li, S. Sehic, R. Ayani, and S. Dustdar. Automatic description of context-altering services through observational learning. In *Proceedings of the Tenth International Conference on Pervasive Computing*, pages 461–477. Springer Berlin/Heidelberg, 2012.

[88] F. Ricci. Mobile recommender systems. *International Journal of Information Technology and Tourism*, 12(3):205–231, 2011.

[89] C. Röcker, M. Janse, N. Portolan, and N. Streitz. User requirements for intelligent home environments: a scenario-driven approach and empirical cross-cultural study. In *Proceedings of the 2005 Joint Conference on Smart objects and Ambient Intelligence: Innovative context-aware services*, pages 111–116. ACM New York, USA, 2005.

[90] J. A. Rode, E. F. Toye, and A. F. Blackwell. The fuzzy felt ethnography – understanding the programming patterns of domestic appliances. *Personal Ubiquitous Comput.*, 8(3-4):161–176, 2004.

[91] D. Ronzan. The battle of concepts: Ubiquitous computing, pervasive computing and ambient intelligence in mass media. *UbiCC*, 4:9–19, 2009.

[92] B. Roy. Paradigms and challenges. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, volume 78 of *International Series in Operations Research & Management Science*, pages 3–24. 2005.

[93] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 8:10–17, 2001.

[94] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85 –90. IEEE Computer Society, Washington, DC, USA, 1994.

[95] B. Schilit and M. Theimer. Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22–32, 1994.

[96] A. Schmidt, M. Beigl, and H.-W. Gellersen. There is more to context than location. *Computers and Graphics*, 23:893–901, 1998.

[97] J. Scott, A. Bernheim Brush, J. Krumm, B. Meyers, M. Hazas, S. Hodges, and N. Villar. PreHeat: controlling home heating using occupancy prediction. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, pages 281–290. ACM New York, USA, 2011.

[98] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, 1976.

[99] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *Workshop on Advanced Context Modelling, Reasoning and Management at The Sixth International Conference on Ubiquitous Computing*. Springer Berlin/Heidelberg, 2004.

[100] T. Strang, C. Linnhoff-Popien, and K. Frank. CoOL: A context ontology language to enable contextual interoperability. In *Proceedings of Fourth IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003)*, pages 236–247. Springer Berlin/Heidelberg, 2003.

[101] E. Tapia, S. Intille, and K. Larson. Activity recognition in the home using simple and ubiquitous sensors. In *Proceedings of the Second International Conference on Pervasive Computing*, pages 158–175. Springer Berlin/Heidelberg, 2004.

[102] P. Tolmie, J. Pycock, T. Diggins, A. MacLean, and A. Karsenty. Unremarkable computing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI' 02, pages 399–406. ACM New York, USA, 2002.

[103] K. N. Truong, E. M. Huang, and G. D. Abowd. Camp: A magnetic poetry interface for end-user programming of capture applications for the home. In *Proceedings of the Sixth International Conference on Ubiquitous Computing*, pages 143–160. Springer Berlin/Heidelberg, 2004.

[104] A.-M. Vainio, M. Valtonen, and J. Vanhala. Proactive fuzzy control and adaptation methods for smart homes. *Intelligent Systems, IEEE*, 23(2):42–49, 2008.

[105] T. van Kasteren, G. Englebienne, and B. Kröse. Transferring knowledge of activity recognition across sensor networks. In *Proceedings of the Eighth International Conference on Pervasive Computing*, pages 283–300. Springer Berlin/Heidelberg, 2010.

[106] T. van Kasteren, A. K. Noulas, G. Englebienne, and B. Kröse. Accurate activity recognition in a home setting. In *Proceedings of the Tenth International Conference on Ubiquitous Computing*, pages 1–9. ACM New York, USA, 2008.

[107] A. Venkatesh, N. Stolzoff, E. Shih, and S. Mazumdar. The home of the future: an ethnographic study of new information technologies in the home. *Advances in Consumer Research XXVIII*, pages 88–96, 2001.

[108] X. Wang, D. Zhang, T. Gu, and H. Pung. Ontology based context modeling and reasoning using OWL. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, pages 18–22. IEEE Computer Society, Washington, DC, USA, 2004.

[109] M. Weiser. The Computer for the Twenty-First Century. *Scientific American*, 265(3):94–104, 1991.

[110] W. Woerndl, J. Huebner, R. Bader, and D. Gallego-Vico. A model for proactivity in mobile, context-aware recommender systems. In *Proceedings of the Fifth ACM Conference on Recommender systems*, RecSys '11, pages 273–276. ACM New York, USA, 2011.

[111] A. Woodruff, S. Augustin, and B. Foucault. Sabbath day home automation: "it's like mixing technology and religion". In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 527–536. ACM New York, USA, 2007.

[112] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S web services composition using SHOP2. In *Proceedings of the International Semantic Web Conference*, pages 195–210. Springer Berlin/Heidelberg, 2003.

[113] D. Wyatt, M. Philipose, and T. Choudhury. Unsupervised activity recognition using automatically mined common sense. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 1*, AAAI'05, pages 21–27. AAAI Press, 2005.

[114] X-10 communications protocol and power line interface. Technical report, X10 Inc. `http://www.x10pro.com/pro/pdf/technote.pdf`, accessed 14. June 2013.

[115] M. Xie, S. Han, B. Tian, and S. Parvin. Anomaly detection in wireless sensor networks: A survey. *Journal of Network and Computer Applications*, 34(4):1302 – 1325, 2011.

[116] R. Xu, G. Mei, Z. Ren, C. Kwan, J. Aube, C. Rochet, and V. Stanford. Speaker identification and speech recognition using phased arrays. In Y. Cai and J. Abascal, editors, *Ambient Intelligence in Everyday Life*, pages 227–238. Springer-Verlag, 2006.

[117] J. Ye, A. K. Clear, L. Coyle, and S. Dobson. On using temporal features to create more accurate human-activity classifiers. In *Proceedings of the 20th Irish Conference on Artificial Intelligence and Cognitive Science*, pages 273–282. Springer Berlin/Heidelberg, 2010.

[118] G. M. Youngblood and D. J. Cook. Data mining for hierarchical model creation. *Trans. Sys. Man Cyber Part C*, 37(4):561–572, 2007.

[119] V. W. Zheng, D. H. Hu, and Q. Yang. Cross-domain activity recognition. In *Proceedings of the Eleventh International Conference on Ubiquitous computing*, pages 61–70. ACM New York, USA, 2009.

[120] F. Zhu, M. Mutka, and L. Ni. Service discovery in pervasive computing environments. *Pervasive Computing, IEEE*, 4(4):81–90, 2005.