Resource Management for Smart Services in the Home

Marios Sioutis

Japan Advanced Institute of Science and Technology

Doctoral Dissertation

Resource Management for Smart Services in the Home

Marios Sioutis

Supervisor: Yasuo Tan

School of Information Science
Japan Advanced Institute of Science and Technology

June 2016

Abstract

Advances in network technology and consumer electronics have brought the dream of an intelligent home a step closer to reality. Currently, the most common approach to introduce intelligence in a home is by using commercially available home automation systems. However, these automation systems have many disadvantages. A non-exhaustive list of these disadvantages includes the following: home automation systems are platforms that are closed or tightly controlled by a single vendor, have severely limited interoperability and extensibility in terms of supported devices and services, offer limited or no third party services and finally the degree of sophistication of the services that these automation systems bring to the table is relatively low. These disadvantages thus lower the value proposition of home automation.

To address the above limitations of home automation and realize the dream of a truly intelligent home, the idea of middleware platforms, collectively known as "home service platforms", was pursued by the research community. These middleware platforms address the problems of traditional home automation systems by following a layered design approach. In a home service platform, the intelligent decision making process is decoupled from the actual hardware and devices, and it is delegated to software programs called "smart services". In turn, these smart services can access any device that is present in the home through a well defined Application Programming Interface (API) that is offered by the home service platform, leading to services of far greater sophistication. This clean separation of devices and application logic with the use of a middleware platform also addresses interoperability issues, and new services may be introduced at will, making such middleware platforms very extensible.

The design of home service platforms enables uninhibited device access to smart services. In contrast to home automation systems that are closed by nature and tightly controlled, this design now raises the possibility for *conflicts among services* over the use and operation of devices. If such conflicts are not addressed, they will negatively impact the user's experience, and the illusion of the smart home will dissolve.

The topic of this research is to address conflicts among smart services by introducing *resource management* as part of the home service platform. Through resource management, the author argues that conflicts among services can be resolved adequately with minimum impact on the user's overall experience. Other contemporary home service platforms tend to have limited capabilities for addressing conflicts or outright ignore this problem.

In regards to conflicts among services over resources, two types of conflicts have been identified:

- conflicts over devices,
- conflicts over physical properties of space.

The first type of conflict occurs when two or more services try to operate the same device in contradictory fashion over an overlapping period of time. A typical example would be the operation of an illumination device by two services, with one service trying to turn a light on and the second trying to turn it off; if the middleware platform does not intervene, the outcome of the above operations is time dependent. Furthermore, if device status feedback is available, the above scenario can lead to rapid blinking of the light.

The proposed home service platform addresses conflicts over devices by treating devices as *resources*, the access to which is managed by the platform itself. The problem of resource management in the home is similar to the resource management in computer operating systems, a source of inspiration for this research. To manage access to devices, the proposed platform introduces four design primitives that are readily usable by smart services:

- device access rights,
- service and user priorities,
- event notification mechanism,
- condition sets.

Using these primitives, the design of resilient services which exhibit well-defined behaviour in case of device conflicts is simplified. Exclusionary access to a device becomes explicit and services have a chance to gracefully recover in cases of conflicts. The effectiveness of these primitives is demonstrated through a set of several scenarios.

The second type of conflict (conflicts over physical properties of space) occurs when two or more smart services operate devices that have conflicting effects on the home environment. For example, the operation of a heater and an air condition unit at the same time is not only self-defeating but also a waste of electric power.

To address conflicts over physical properties, the proposed system manages physical properties of space as environmental resources and provides a high-level API to interact with them. Using this powerful API, a smart service may make requests such as “set the temperature of the room to 25 °C”

or " set the illumination around the user to 300 lx, thus further simplifying the development of smart services.

The proposed system is now able to not only detect such conflicts (using the notion of "Area of Effect") but also attempt conflict resolution, using either a space-based or intensity-based resolution scheme. To find a solution to such conflicts, the platform must decide on a set of device settings that can at least partially fulfill the conflicting environmental requests. As the number of devices and possible device settings increases, the search space becomes increasingly vast and an exhaustive search is prohibitive. The proposed system thus utilizes local and global search algorithms as well as simulation of physical properties in each iteration step to evaluate possible solutions. The effectiveness of this approach is demonstrated over several experiments regarding illumination in which conflicting requests were made. The experiments were performed in a real smart home environment and the difference between estimated and actually measured illumination intensity formed the basis for evaluating these solutions. Furthermore, the system was evaluated in terms of performance. As conclusion, the proposed home service platform is able to produce high quality solutions within a tight execution time frame of 1 second.

The final key piece of information necessary to perform conflict resolution is user location information. Currently, in contrast to outdoor location systems where GPS is the defacto standard, there is no such standard for indoor location. A multitude of indoor location systems exist with different advantages and disadvantages. For the purpose of this research, a user location information system based on passive infrared sensors and space subdivision was developed. This location system is non-intrusive, relatively low cost and can be replicated with off-the-self parts, while providing location information that is accurate to within 50cm.

Overall, this research led to the development of a home service platform that advances the state of the art of smart homes. The proposed platform offers compelling features and advanced functionality, while at the same time making smart service development simpler and more robust. By addressing conflicts among services adequately, the last technical obstacle towards mass adoption of home service platforms by consumers has been cleared. Any future home service platform that fails to offer functionality that is at least equivalent to that of the proposed platform will face obsolescence.

Keywords. Smart Home, Home Service Platform, Pervasive Computing, Resource Management, Conflict Resolution

This page is intentionally left blank.

Acknowledgments and Dedications

To my friends and family,
to my colleagues and advisers over the years,
you made this work possible.

Thank you.

To Penelope,

I've reached Ithaca, but you were gone.

Contents

Chapter 1

Introduction

1.1 Quality of Life in the House

Since the dawn of time, mankind has fought a hard battle for survival. The weapons of choice for this battle include (but are not limited to) intuition, perception, persistence, and above all, tenacity; countless tries and failures with the occasional successes that push the species one tiny step forward each time. Each hard fought success has made an impact to the way of life of our species by first being stored in our collective memories and knowledge through oral, written and later on digital means, and then replicated countless times by the vast swath of the population, in order to aid survival.

The struggles of mankind over the past couple of millennia, with its incremental successes such as the combustion engine and electricity, has lead us to the current age, often hailed as the great "digital age", where in any reasonably civilized society there is a guaranteed standard of living. Although exceptions to the rule still exist, survival of the individual is mostly thought as a solved problem in such societies. However, mankind has not rested on its laurels; with survival mostly a solved problem, the focus now swifts to further improving quality of life in terms of safety, comfort and individual fulfillment and growth.

One such area that is expected to be at the forefront in our collective struggle for a better quality of life in the coming years involves *the house*, an indoor environment tightly coupled with the quality of life of an individual. Many activities that directly pertain to survival, safety and general well-being of an individual are associated with the house. It is our respite from the elements of nature, a safe haven from ill-meaning individuals, a place for rest and sleep, a place where hunger can be satiated, hygiene maintained, and in the recent years, a space for entertainment and personal growth. It

should be of no surprise that such a multi-purpose space is, to this day and age, still riddled with inefficiencies.

The main focus of this research is the house and more specifically the "smart home", a concept that tries to address the numerous inefficiencies present in the house. By extension, it is to this fight, the fight for a better quality of life, that my research hopes to contribute, even if this contribution is infinitesimal.

1.2 The Smart Home

As a result of progress in automation technology and electronics, the notion of "smart houses" reached the general public in the late 1970's with the release of products supporting the X10 protocol[11]. Furthermore, according to [29], the term "smart home" appears for the first time in an official capacity in 1984. In this term, the word "smart" is interpreted as "intelligent", indicating that the house itself exhibits some behaviour that can be perceived as intelligent.

Indeed, in recent years, the term "smart house" has been closely associated with the regulation of power consumption and its reduction, achieved through automation and other means. It can be easily argued that regulating power consumption is a "smart" thing to do, in the sense that regulating power consumption leads to less power consumed, something that directly benefits an individual in monetary terms.

However, this definition of a smart house is very narrow in scope, and for the purpose of this research the term "smart home" is preferred. The interpretation of this term by the author is the following: a smart home is any indoor household environment which exhibits behaviour that can be classified as intelligent, with the ultimate motive of improving the quality of life of the house occupants. This definition encompasses power consumption but also includes (but not limited to) notions such as personal safety, hygiene regulation and comfort.

To an extent, the notion of a house exhibiting "behaviour" may seem intuitive to a technologically savvy individual. However, this notion was nothing sort of revolutionary as recently as fifty years ago, thus it is prudent to expand on it and explain the finer details of house behaviour.

Behaviour implies *action, reaction to stimulus and decision making*, concepts that must be explained in the context of a home environment. By its inherent nature, a house is inert and static. There are characteristics of the house such as weather insulation and natural light penetration that, although designed intelligently, are passive traits of the house. In other words,

no dynamic action occurs as a result of these traits. There is no reaction to stimulus and there is certainly no decision making in a newly built, unoccupied house.

It is through electric and electronic devices installed in the house, where it can be argued that the house exhibits some sort of behaviour. Indeed, household appliances and electronic devices exhibit behaviour (and in many cases quite complex behaviour) that may be considered intelligent. As such an example, consider an air condition unit: it exhibits

1. action, as it regulates the temperature and humidity of a room, it demonstrates

2. reaction, as through its sensors it varies its operation and power consumption, and finally

3. decision making, as demonstrated through its "auto" operation setting, where the device relies on some algorithm to decide its operation.

The above example shows a typical device that exhibits a complex behaviour that can be considered smart. However, introducing smart devices in the house is not enough to consider a house as being "smart". A way to assess if a house is smart or not is to evaluate how ubiquitous and pervasive this intelligence is throughout the house. It can be argued that introducing smart devices in the house will improve the quality of life and introduce some intelligent behaviour in the house. However, this intelligence will not be pervasive; it will be, at best, *fragmented*. Each smart device will be smart, on its own, and the user will be able to tell so.

To realize a truly smart home, the boundaries among individual devices must be removed so that the illusion of an intelligent indoor environment, as a whole, can be maintained. This means that devices should be able to interoperate in harmony, achieving tasks that are complex and the contribution of each device although distinct, is just a part of bigger effort. In this model, the contributions of each device are blended together to achieve the greater task at hand.

1.3 Ubiquitous Intelligence and Automation

The first attempts at a smart home involve automation. To this day, many vendors such as Insteon[6] offer automation solutions for the home. The main areas where such solutions focus (but are not limited to) are home security and surveillance, energy management and automated control systems.

Home security automation solutions involve a vast array of sensors such as open/close sensors, CO^2 sensors, broken glass sensors, cameras and a central controlling device that processes their feedback. In case of an emergency, this central controlling device may be configured to contact a security company or the local authorities directly, notifying them regarding the nature of the emergency. Furthermore, surveillance solutions usually offer the option of streaming video, providing an individual the chance to check up on the status of a loved one or a pet, and in case of emergency contact a family member or a friend for assistance.

Home energy management systems (collectively known as HEMS) is a field of applied research with growing popularity, especially in this day and age where cheaper energy from fossil fuels is in the process of being heavily regulated and alternative power sources such as nuclear and renewable energy sources still struggle to become viable alternatives of producing power. HEMS provide immediate monetary benefits to the occupants of a house by reducing the amount of money spent on electric energy throughout the day. These systems again rely on a series of sensors to measure power consumption, a central controlling device that provides the user with feedback and in recent years integrate with demand-response systems deployed by various electric power companies. Such systems have the capability to schedule the operation of devices during times when the electricity cost is lower and also sell electric power back to the grid, if a solar panel or wind turbine is present. With the advent of electric cars, such systems will gain in popularity even more.

Finally, automation control systems allow the occupants of a house to automate tedious tasks such as controlling the illumination of a room, gardening solutions and operating specific devices under certain conditions. These automation solutions do exhibit action and reactions to stimuli (they can determine whether the conditions for invoking an automation procedure have been met) but it can be argued that there is limited to no automated decision making in the process. Nevertheless, such systems are deemed useful by their users, assisting them in their everyday lives.

A common theme in automation systems is that they instrument multiple devices and combine them with a central control point to achieve their task. Such systems are more sophisticated that single smart devices, albeit still have a tight focus on a single task. These systems are pervasive and ubiquitous in spatial terms as their deployment may cover the entirety of the house. However, these systems are limited in terms of the functionality offered, thus of limited intelligence.

Traditional automation systems have several other disadvantages that severely limit their usefulness. First, automation systems are often incom-

patible. Devices from other vendors can rarely be integrated with the rest of the automation system, leading to vendor lock-in. Second, the functionality provided by such solutions is tightly controlled by the automation system vendor. Upgrades to the functionality are usually distributed in the form of closed-source firmware upgrades for the central controlling point, with the users unable to officially extend and enhance said functionality. Thirdly, each automation system comes with its own central control point, leading to a clutter of devices that are both difficult to operate, maintain, manage and secure. Many automation systems also prove to be a fertile attack vector, by adopting weak encryption protocols or no security mechanisms at all.

To conclude, automation systems have many disadvantages and although they may have spatial pervasiveness in the confines of the home, due to their limited functionality fail to be truly intelligent.

1.4 Ubiquitous Intelligence and the Home: a Home Service Platform

To overcome the limitations of automation systems, a different system design approach has been pursued by the smart home research community. In this approach, a software platform is designed in such a way that can communicate with a vast array of heterogeneous devices, ranging from sensors to cameras, air conditions to multimedia equipment and others. The intelligent decision making is delegated to software programs that integrate or communicate over the network with the software platform in order to interact with the devices in the house. Throughout this research, such platforms will be referred to as "home service platforms" and the intelligent decision making programs as "smart services" (or plainly services).

A home service platform following this design paradigm can fulfill the criteria of ubiquitous intelligence. First, spatial ubiquity can be achieved in the same fashion as automation systems; as long as devices are installed throughout the house, the system can exhibit its intelligence everywhere inside the house. Secondly, in stark contrast to the automation systems, by utilizing the home service platform services have now access to a multitude of devices of diverse types. Using these diverse devices, it is now possible to achieve far more sophisticated tasks that were simply not possible in the confines of a traditional automation system.

This design paradigm also addresses the shortcomings of traditional automation solutions by offering

1. network communication protocol independence,

2. smart service implementation flexibility,

3. open or at least "hacker-friendly" service development tools and libraries.

Expanding on the above advantages, a home service platform avoids vendor lock-in and has better device integration from multiple device vendors by implementing various, often at times competing network communication protocols. This fact combined with the diverse types of devices often supported leads to a more flexible ecosystem on which smart services are free to utilize whichever devices are deemed necessary to carry out their tasks. Furthermore, in their effort to support functionally diverse devices, such home service platforms often some protocol independent API focused on the functionality of the device, abstracting any communication protocol minutia and in turn making the development of smart services easier. Lastly, a very important feature of home service platforms is their improved value proposition by virtue of being extensible systems; new functionality can be added by deploying smart services developed by a third party. In their effort to gain traction, the tools for developing smart services are often offered usually free of charge to third party developers, enterprise companies or hobbyist hackers alike. It is expected that when a home service platform achieves widespread adoption, such third parties that develop smart services are going to play a crucial role for enhancing the functionality of the platform by offering new and innovative smart services.

In their current form, home service platforms still have some limitations that further prevent them for reaching widespread adoption. This research hopes to address some of these limitations and thus make home service platforms an even better value proposition over traditional automation systems. An overview of the most important limitations of home service platforms is given in the next section

1.5 Limitations of Current Home Service Platforms - Conflicts Among Smart Services

The introduction of home service platforms lifts the limitations of traditional automation systems. However, these platforms also remove the tightly controlled and usually well-tested environment associated with automation systems which is responsible for their robustness. Automation systems operate inside well-defined boundaries, with very specific tasks to complete.

In contrast, home service platforms offer an open-access device model; given sufficient credentials, a smart service may access any device it needs, at

any given time. In this open-access device model the possibility for conflicts among smart services emerges. It is the view of the author that in order to create a functional and well-behaving home service platform, management of *conflicts* among smart services is of utmost importance.

Before the overall scope of this research is introduced, it is necessary to consider the nature of the conflicts which may occur among smart services as well as their cause. In general, the cause of a conflict lies with the limited availability of a given *resource* that is necessary to two or more entities at the same time. In the context of smart home services, two major types of conflicts have been identified:

- conflicts over the use of a device and,
- conflicts over the environmental properties of the home environment.

The first type of conflict is easy to grasp; two or more smart services try to utilize the same device or home appliance in a contradictory fashion. For example, two smart services may try to use the tv set at the same time, one for entertainment purposes and another for streaming the latest news. Another example would involve the use of a surveillance camera; a smart security service uses the camera to detect break-ins or suspicious movement whereas an indoor location system may use the camera to track the location of the occupants of the house. Finally, an even simpler example is that of two services trying to control directly an air condition unit, using different temperature settings and/or operation modes.

The above examples are typical cases where it is necessary for a smart service to operate a device in an *exclusionary* manner. This problem may also be considered a parallel to the resource management that typical computer operating systems perform, a field of research from which inspiration was drawn.

The second type of conflict has a more complex nature. In this type of conflict, two or more services operate devices that have conflicting effects on the environment. A simple example is the following: a smart service is operating an air condition unit in cooling mode and at the same time another smart service operates a heater in the same room; although the smart services do not conflict directly over the use of devices (as they operate different devices) they do produce conflicting effects on the environment.

Unless there is specific support from the home service platform, these two types of conflicts cannot be predicted ahead of time and user input is necessary to resolve them. In turn, if user intervention is necessary to resolve conflicts, the illusion of a ubiquitous intelligent home environment dissolves.

1.6 Targets and Scope of this Research

This research aims to advance the state of the art of home service platforms by addressing conflicts of smart services through comprehensive resource management. A home service platform is designed from the ground up with conflict detection and resolution as top priority.

The proposed home service platform addresses the two types of conflicts introduced earlier with distinct approaches. For conflicts over environmental properties, the proposed platform uses a novel approach where the properties of the environment are treated as resources themselves. The environmental properties targeted by the platform are

1. illumination,
2. temperature,
3. humidity and finally,
4. sound and noise levels.

By undertaking the responsibility to manage the environmental resources itself, the proposed platform now has the ability to detect and even attempt to resolve these conflicts. The management of environmental properties is discussed in detail in chapter 2, where its effectiveness is demonstrated through various experiments.

Regarding device related conflicts, the platform draws inspiration from computer operating systems and introduces a set of primitives to facilitate exclusionary use of devices: device access rights, service and user priority, an event notification mechanism and finally condition sets. The rationale and effectiveness of these primitives are demonstrated in chapter 3.

Finally, the last piece that completes a home service platform that aspires to be characterized as ubiquitous and intelligent is the availability of context information. In the author's view, information regarding the *location* of the users inside the house is necessary to offer reactive smart services. Although a variety of indoor location systems exist today, most of them are the result of research efforts and there are hardly any commercially available solutions that do not have an exorbitant price tag. For this reason, an experimental indoor location system based on passive infrared sensors was developed. The details regarding this indoor location system as well as its overall evaluation can be found in chapter 4. Although this location system was proven to be of limited usage, its design and availability affected the design of several APIs regarding location information offered by the proposed platform.

To finish this introduction, the advantages of using the proposed platform are introduced. Firstly, due to the comprehensive API for discovering and controlling devices, development of smart services is simplified and services can be executed on any instance of the proposed service platform, regardless of the variations of deployed devices in a house.

Secondly, the proposed platform provides a predictable execution environment for the smart services; any kind of change in the execution context (such as losing access rights to a device) can be intercepted and services can then react and take corrective measures, or even voluntarily suspend their execution. This predictable behaviour is a key factor to designing resilient services that behave consistently and in a predictable fashion.

Finally, by delegating the management of environmental resources to the home service platform itself, it is now possible to detect and resolve conflicts regarding environmental properties. Furthermore, the difficult task of controlling environmental properties from a smart service by directly controlling devices such as lights, curtains and others is simplified; the service only need to specify the intensity of the property as well as the area over which the desired intensity is to be enforced. The proposed platform then proceeds to find the most appropriate combination of devices and device settings to fulfill such request by services.

Chapter 2

Management of Environmental Resources in the Home

2.1 Introduction

The use of home service platforms opens the way for new and highly sophisticated services to be deployed in the home. As the sophistication level of these smart services increase, the need to control aspects of the user's surrounding physical environment arises. Aspects of the environment such as temperature, humidity, illumination as well as sound and noise levels can be considered to be innate properties of the physical environment. These properties or characteristics are readily perceptible to the occupants of the house[1], affecting them in many direct and sometimes more subtle ways.

The physical environment as perceived in terms of the four properties identified earlier is for example tightly interlinked with a sense of comfort; a room temperature that is too high or too low usually leads to discomfort. Furthermore, different illumination conditions may be more fitting for specific tasks or time of the day, while also having an impact on the mood of the occupant. Sound such as music may affect the mood of the listener and stimulate his or her creativity, while on the other hand noise may be irritating, especially when the occupant tries to relax or sleep.

With such a huge potential to affect an occupant's every day life, it only stands to reason that a smart service of high sophistication should be able to affect these physical properties. Currently, smart services do have the ability to manipulate these physical properties *indirectly*, by operating devices that have an effect on these properties. For example, a smart service may turn on and off a light to control illumination or operate an air conditioning unit

[1]Or any person in the house for that matter

in hopes of adjusting the temperature.

This model of indirect interaction with the physical properties of space turns out to be less than ideal. First, it is very difficult for a smart service to estimate the effect of the operation of a device on the surrounding environment. Thus, trying to control a physical property this way is rudimentary at best, complicated in terms of programming and grossly unsatisfying for user experience at worst. Secondly, the possibility of conflicts in regards to physical properties arises; two or more services may try to affect physical properties in a contradictory way. For example, two services may try to achieve both a bright and a dark illumination environment in the same room. Such scenario is a common case of conflict over environmental resources. Finally, depending on the availability of devices in the home environment and their variability, it becomes hard to develop smart services that are "write once, run everywhere" without including specific logic for individual devices.

To address these issues, a different approach where *the physical properties of space are treated as environmental resources* that are managed directly by the service platform was pursued. In this approach, smart services can make abstract requests for environmental resources, and then having the home service platform decide the most appropriate combination of devices and device settings that should be used to fulfill these requests. Furthermore, by delegating the responsibility of management of environmental resources to the home service platform, it is now possible to detect and resolve conflicts regarding physical properties at least partially. Finally, by introducing a more abstract API to facilitate requests regarding physical properties, smart service development is simplified, reducing development time and making services more easy to deploy in any home.

The work presented in this chapter was published in [58].

2.2 Related Works

To the best of the author's knowledge, so far there has not been a home service platform that integrates physical properties of the environment as resources. Thus, the related works usually only focus on a single physical property such as illumination. It is for this reason that related works in this section will be presented in groups, depending on the physical property that is to be controlled.

Illumination The work presented in [52] follows a similar approach with the proposed system. In it, a set of constraints regarding illumination are first formulated and then with the use of genetic algorithms an acceptable

solution is found. However, no discussion regarding the performance of the system in terms of execution time is presented.

In [50], the problem of finding appropriate illumination settings for devices is formulated as a binary satisfaction model and also as a continuous satisfaction model. At its core, a binary search algorithm is used to derive the appropriate settings for devices. The results show a good match between the measured and estimated illumination intensity.

In [49], a closed-loop illumination control system is presented that in its core uses stochastic hill climbing. Its convergence speed was reported to be in the order of 5 minutes, thus making its use in real-time scenarios hard to recommend. Nevertheless, power savings of up to 33% were reported in comparison to an office environment that does not utilize this system.

Temperature and Humidity The management of temperature and humidity in the proposed platform is currently not implemented. However, related research in the field of Home Ventilation and Air Conditioning systems (HVAC) is plentiful and can serve as a source of inspiration.

A review of contemporary HVAC systems is presented in [44]. In it, an HVAC system is described as a "complex, non-linear, multi-input multi-output system". A mathematical model that describes the operation process of such systems is difficult to construct, therefore different intelligent control methods utilizing for example neural networks, evolutionary algorithms and fuzzy logic are proposed as the preferred modeling tools. Examples of HVAC systems deploying fuzzy logic include [17], [18]. Furthermore, a method for training such a fuzzy logic controller using genetic algorithms is presented in [32]. A system that further utilizes past data to make accurate predictions about the temperature can be found in [43].

To attempt to effectively control temperature and humidity, a simulation model that is fast enough and produces reasonable accuracy results is needed. In [56], a method for fast simulation of temperature based on proper orthogonal decomposition (POD) is discussed. Such an approach can be appropriate for the proposed system and would compare favourably to time-consuming simulation methods based on computational fluid dynamics. A second work that utilizes POD and finite volume method can be found in [41].

Sound In a similar fashion, a fast simulation method for sound sources would be necessary for the correct estimation of sound and noise levels. A very interesting work based on ray tracing is presented in [64]. The authors proposed a guided ray tracing system which is able to compute the effects of a sound source from a receiver's position from 8 to 30 times per second.

As it is explained in section 2.3.6 a ray tracer for the proposed system has already been developed. With slight modifications to account for diffraction, it could potentially be used for the modeling of sound and noise in the home.

2.3 Design and Rationale

2.3.1 Design Principles and Advantages

The design of the proposed system comes as a direct answer as to how to simplify the management of environmental resources in the home. Two simple ideas that answer this question lead our design:

1. offer a high-level API to control physical properties to the smart services and

2. delegate the management of the physical properties to the home service platform.

The presence of a high-level API for controlling physical properties of space is a unique feature of this platform. Using this API a smart service may make a request regarding the intensity of a physical property over an area of space. As an example, a service may request that the illumination around the user be set to a specific amount of lux, or that the temperature of a room be set at 25°C.

The advantages of using such a high-level API is that service development is *simplified*. Any complex and error-prone logic for identifying individual devices and estimating their effects on the environment becomes unnecessary. Eliminating this portion of software logic leads to decreased development time and allows the service to be deployed in any instance of the home service platform, regardless of the different devices that may be in use. Furthermore, even if such a complex logic was developed successfully for a smart service, this effort is an effort that would have to be duplicated for any other service that would want to achieve similar results. This is a process that is bound to introduce errors and produce corner cases that were not taken into consideration during development, thus having a negative impact on the user's experience.

The advantages of delegating the management of physical properties to the home service platform are also clear. This feature is complementary to the high-level API for controlling physical resources and is the mechanism that makes the most appropriate decisions regarding how to better fulfill requests regarding physical properties. On top of the advantages described

above, the home service platform is also able to perform conflict detection and resolution regarding physical properties, something that was previously impossible. Conflicts over environmental resources can now be detected based on the overall "Area of Effect" that a request has. Conflict resolution can also be achieved by making a compromise among the conflicting requests. The conflicting request can be compromised either in terms of the area that they cover or the intensity of the physical property that they address.

2.3.2 Area of Effect and Compromising Techniques

The proposed platform supports conflict detection and conflict resolution for environmental resources such as temperature, humidity, illumination, as well as sound and noise levels. To achieve this, the concept of "Area of Effect" was introduced.

The area of effect can be classified in two categories: the area of effect of a device and the area of effect of a smart service. The area of effect of a device is the physical space over which the device is able to affect one or more properties of said space. The effect that the device has on the intensity of a property of the physical space must exceed a certain threshold, or else its effects can be deemed insignificant and thus ignored. As this threshold value increases, it is usually the case that the area of effect decreases. For example, the area that a light may illuminate at an intensity of 1000 lux must be many times smaller than the area which could be illuminated at only 100 lux.

The area of effect of a service is the physical space over which the service enforces the intensity of a physical property over a certain threshold. The area of effect of a service is the result of the combination of the area of effect of the devices that are used to fulfill the needs of the service.

Depending on the physical property, some properties may be contained successfully in smaller areas than other properties. Taking illumination as an example, barring any windows or transparent surfaces, it is usually limited in the confines of a single room. Conversely, sound and noise may penetrate walls and be perceptible in other adjacent rooms or different floors of the same building. Temperature and humidity can be reasonably contained and controlled inside a room, but thermal loss to adjacent room is a fact that should be accounted for.

The area of effect can now be used to check for conflicts. Should two or more smart services try to achieve a different effect on the environment, their respective areas of effect can now be examined for conflicts. A simple example are two smart services that want to achieve different temperature settings in the same room. The area of effect for these services becomes the room itself.

Assuming that these services want to achieve an exact intensity setting, these services conflict. It is worth noting that, if these two services specified an intensity of *more than* 25°C and 28°C respectively, without an upper bound, these services do not conflict for they are pushing the intensity of the physical property (in this case temperature) towards the same direction.

After a conflict has been detected, the next step is to attempt conflict resolution. In stark contrast with device conflicts (which are the main topic of chapter 3), physical properties may in essence be treated as shared environmental resources, i.e. these resources can be used at the same time by multiple services. However, the requests made by the smart services may be only partially satisfied.

Two main compromising approaches for resolving conflicts over environmental resources are proposed:

- space-based conflict resolution,
- intensity-based conflict resolution.

In the space-based conflict resolution approach, the system tries to minimize the overlapping area of effect of the services. This approach can be used when the physical property that is to be controlled can be reasonably contained over a given area. Illumination is the most representative case of such a property. For example, in a reasonably sized room it is possible to illuminate only part of it, keeping the rest of room substantially darkened.

In contrast, the intensity-based conflict resolution approach is used when the physical property that is to be controlled has a tendency to become homogeneous over an area. The most representative physical property of this kind is temperature, which tends over time to spread evenly inside the confines of a room. In the intensity-based conflict resolution approach, an intermediate intensity that will partially satisfy the conflicting requests of the smart services is applied. For example, given two services that want to achieve 22°C and 25°C of temperature in a room, the system may choose to apply any temperature in the $[22°C, 25°C]$ range. The details for choosing the most appropriate temperature in this range may vary depending on the sophistication of the algorithm employed and may take into consideration many parameters that affect the perceived temperature by the user.

2.3.3 Achieving an Optimal Solution: a Combinatorial Problem

Having established that the home service platform is responsible for the management of environmental resources, it must now fulfill request regarding

these resources as optimally as possible. To satisfy these requests, the platform must operate one or more devices with appropriate settings.

As the number of settings for each device and the number of devices increases, the number of possible combinations explodes exponentially. For n number of devices that have m possible settings, the number of possible configurations C becomes:

$$C = m^n \tag{2.1}$$

In the experiments section, a room with 25 lights that have dim-switch functionality ($0\% - 100\%$ illumination intensity) is used. This particular setup creates an immense search space (100^{25} possible configurations) that has to be searched to find an appropriate solution. Although typical room configurations and device deployments currently may seem unusual, this does not detract from the fact that the problem of operating appropriate devices with the most appropriate settings is in essence a *combinatorial problem*.

The reason that such configurations are unusual is due to the simple fact that up until now they were difficult to operate by regular users. With the presence of a home service platform, such rich configurations with many devices are not only possible, but even desirable, as they enable more granular control of the physical properties of the home.

There are several options to explore the search space of this problem. One of the first approaches pursued was a heuristic method as described in [57]. The problem with heuristic methods lies in the fact that such methods may have implicit assumptions regarding the environment and may fail to perform adequately well in other scenarios that differ significantly.

In light of such previous experience, local search algorithms that make no hidden assumptions were pursued. A local search approach is flexible enough to produce acceptable results even in the most unusual circumstances due to their agnostic nature.

2.3.4 Local Search and Search Iterations

For each iteration during a local search, one solution candidate is selected and is evaluated for its overall fitness. After the evaluation of the candidate solution is complete, the local search meta-heuristic is used to guide the search towards even better solutions.

Some well-known meta-heuristic strategies are the following: Hill Climbing, Simulated Annealing, Tabu Search and Ant Colony Optimization. Furthermore, Genetic Algorithms can also be used to tackle such combinatorial problems. Each of these meta-heuristic (or classes of) strategies vary in operation and in their handling of candidate solutions. A simple hill climbing

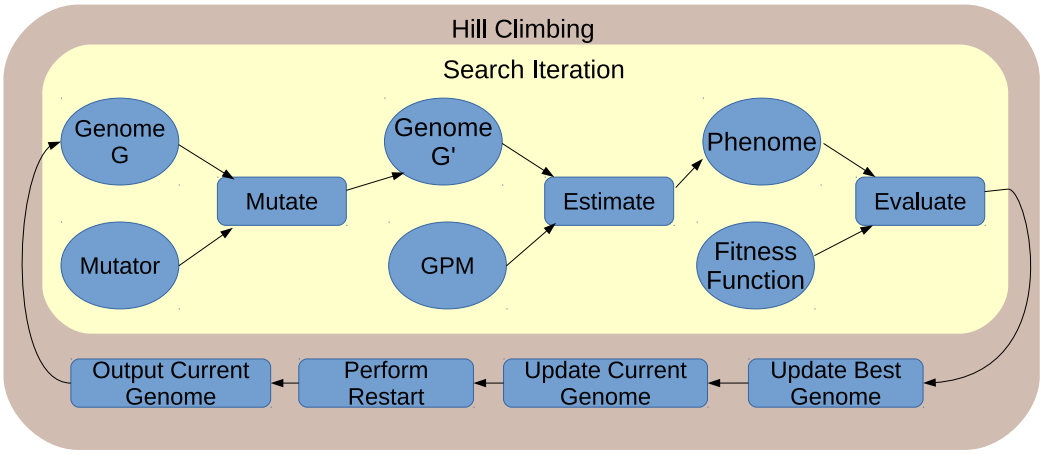

Figure 2.1: Local Search Engine Configuration

algorithm will for example, reject any solution that is strictly worse than the currently best known solution. However, this kind of greedy approach may lead to a local optimum point, failing to find better solutions that otherwise exist in the search space. Random Restarts is a common strategy supplementary to Hill Climbing which resets the search and starts off again from a new random point in the search space. Tabu Search will not consider solutions that have been visited before and so on. An exhaustive list of global optimization algorithms and their details can be found in [68].

The configuration of the local search utilized in this platform can be seen in Fig. 2.1. The proposed system utilizes a hill climbing meta-heuristic strategy and, depending on the configuration, restarts may occur.

Before describing the steps performed in each search iteration, several key concepts that pertain to global search algorithm must be briefly introduced. These concepts are also prevalent in genetic algorithms, so the nomenclature borrows heavily from the science field of biology.

The first two concepts are those of the *Genome* and *Phenome*. According to [68], "the search space $\mathbb{G}$ (genome) of an optimization problem is the set of all elements g which can be processed by the search operations". In the context of this research, given a set of devices D and a set of settings S_i for the i-th device d_i, a *genotype* g takes the form of:

$$g = (s_1, s_2, ..., s_n) \text{ where } s_i \in S_i, n = \text{ number of devices} \tag{2.2}$$

The search space thus becomes

$$\mathbb{G} = S_1 \times S_2 \times ... \times S_n \tag{2.3}$$

As it is, a genotype cannot be directly evaluated for its quality; it is just a set of device settings. The next step is to evaluate what would be the effect of these device settings on the environment. Here the notion of the *Phenome* appears.

The phenome $\mathbb{P}$ (also known as the problem space) of an optimization problem is the set containing all elements p which could be its solution. A candidate solution $p \in \mathbb{P}$ is also known as a phenotype.

For a given problem there may be many representations of $\mathbb{P}$. In the context of this research, the effects that devices have on the environment have to be quantified. Thus, given a set of n points, and I_i being the intensity of the physical property at point i, a candidate solution $p \in \mathbb{P}$ takes the form:

$$p = (I_1, I_2, ..., I_n) \tag{2.4}$$

Furthermore, since the intensity of a physical property can be measured as non-negative real number, P is an n-dimensional $\mathbb{R}$ space.

$$\mathbb{P} = {\mathbb{R}_{\geq 0}}^n \tag{2.5}$$

It is clear now that there is a need for a specific function that can convert from a given genotype to a phenotype in the problem space. Such functions are known as *Genome-to-Phenome mapping*s. This *gpm* function has the following property:

$$\forall g \in \mathbb{G} \exists p \in \mathbb{P} : gpm(g) = p \tag{2.6}$$

In other words, each possible genotype g can be mapped to a candidate solution. An inverse function may exist in other problems, but in this research the *gpm* maps from a discrete space of device settings into a continuous space of physical property intensity. Thus, strictly speaking, such an inverse function is not possible[2].

Depending on the problem, the actual implementation of a *gpm* function may vary significantly, from a simple function that completes in an instant to multi-day complex simulations.

After a candidate solution has been found, it is time to be evaluated in terms of its overall fitness. For this purpose, special *objective function*s that map from a phenotype to $\mathbb{R}$ are used.

$$Obj(p) \rightarrow \mathbb{R} \tag{2.7}$$

These objective functions are specific to the problem domain, and usually contain expert knowledge directly pertaining to the problem at hand. In

[2]Reduced accuracy inverse *gpm* functions may be possible by making the problem space discrete, but actually creating such functions may still be impossible or extremely difficult.

there are multiple objectives that need to be optimized, the problem becomes a *multi-objective optimization problem*. This is the case for the proposed system when a conflict over physical resources occurs; two or more smart services request conflicting properties on the environment and the system performs a multi-objective optimization for these conflicting requests.

In multi-objective optimization, a *fitness function* $Fit(p)$ can be used to determine the overall fitness of a candidate solution. A fitness function combines the results of all the objective functions currently present into a single index, usually in $\mathbb{R}$, that can be used for comparisons among candidate solutions.

$$Fit(p) = f(Obj_1(p), Obj_2(p), ..., Obj_n(p)) \rightarrow \mathbb{R} \tag{2.8}$$

As is also the case with objective functions, it cannot be stressed enough that the quality of this fitness function is going to have a huge impact on the results of the local search. There are many approaches to multi-objective optimization such as Weighted Sums, Pareto Optimization, Constraint Handling and others, each with its own advantages and disadvantages. Depending on the nature of the problem, some approaches may be more fitting for certain classes of problems than others.

In the following sections the details for each discrete phase of a search iteration step (exploration, simulation and evaluation) will be explained.

2.3.5 Exploration

The exploration step of each search iteration is responsible for generating a new genome that will later be examined for its fitness. This generation of a new genotype is achieved with the use of *mutation operations* (also known as mutators). These mutation operations operate on a given number of parent genotypes and produce a new offspring genotype by altering and/or combining the characteristics of the parents.

In local search, the *null operation* and a *unary operation* are prevalent. A null operation can generate a genotype without any external input. This new genotype is usually created at random or, depending on the problem, be a copy from a collection of initial genotypes that are deemed appropriate.

The unary mutation operation receives as an input a parent genotype. This genotype will then be modified and a single offspring genotype will be produced. The way that the parent genotype is modified is again problem-specific.

In evolutionary algorithms, binary and even n-nary mutation operations are more common. As their names imply, these mutation operations combine 2 or more genotypes to produce a new offspring genotype.

Regardless of the context of use, it must be stated that the quality of the mutation operation will effect the overall results of the global search. This is because the mutator operation dictates which neighbors of the current accepted solutions will be visited. This exploration needs to be versatile enough to consider neighbors which have a good chance to escape local minima.

Mutation Operations Used For the needs of this system, two unary mutation operations were developed: a random mutator and a binary-search mutator. The null operation used in the system is again a random genotype generator that picks initial values for the genotype.

The behaviour of the random unary mutator is as follows: provided with an initial genotype g, the mutator will first select one "gene" (in this case, the setting of one device) s_i and assign to it a random value from S_i. This random unary mutator is fair in terms of the probability of each device being considered for mutation in each iteration. However, if a device i has a vast number of possible settings compared to another device j (i.e. the cardinality relation of the set for these two devices is $|S_i| >> |S_j|$), the settings for the device i will statistically have less chances to be examined. If the number of all devices involved in the search have the same number of settings, then this random unary mutator is completely fair. Furthermore, even in the case where the number of settings for each device differs significantly, it is still trivial to create a completely fair random mutator.

The second unary mutator that was developed for this system is a "binary search" mutator. This mutator is stateful, i.e. it has a state that is retained through consecutive calls to this mutator. This binary search mutator operates as seen in Algorithm 1. In essence, one device is picked at random, and the range of possible settings are evaluated using binary search. When there is no further improvement possible, the mutator will reset its state (associated with the gene index that is under investigation) and proceed to select another gene in its next iteration.

It must be stated that this binary mutator operation has a hidden assumption: if the last mutation operation is not present in the genome g that is supplied as an initial argument, it means that the mutation produced strictly worse results and thus was rejected by the meta-heuristic strategy. Although this assumption is correct when the meta-heuristic strategy is a simple hill climbing, this assumption does not hold true for all possible meta-heuristics. For example, if a simulated annealing meta-heuristic search strategy is used, it is possible that, although the mutation produced an overall less fit phenotype, it was adopted as the base for the new search iteration due to high

Algorithm 1 "Binary Search" Unary Mutator

```
1: geneIndex = −1, geneIndex, geneValue
2: function BINARY MUTATOR(g)
3:     if geneIndex = −1 then
4:         return initBinarySearch(g)
5:     else
6:         return performBinarySearch(g)
7:     end if
8: end function
9:
10: function INITBINARYSEARCH(g)
11:     geneIndex ← Random(geneRange)
12:     geneValue ← Random(min, max)
13:     decide search direction
14:     g′ = g
15:     g′[geneIndex] = geneValue
16:     return g′
17: end function
18:
19: function PERFORMBINARYSEARCH(g)
20:     if g[geneIndex]! = geneValue then
21:                                ▷ Previous gene modification did NOT survive
22:         change search direction if necessary
23:     end if
24:     Update min, max search range
25:     geneValue ← Random(min, max)
26:     g′ = g
27:     g′[geneIndex] = geneValue
28:     if min == max then
29:         geneIndex = −1                          ▷ Binary Search Finished
30:     end if
31:     return g′
32: end function
```

"temperature"[3].

The two mutators will be compared in terms of result quality and speed performance in the experiments section of this chapter.

2.3.6 Simulation

To estimate the effects that a given genotype has on the indoor environment, a simulation step becomes necessary. This simulation acts as the *gpm* function of the local search, i.e it maps a set of device settings (the genotype) to the estimated effects that these results would have on the environment if they were to be applied (the phenotype).

The phenotype for the classes of conflicts that the proposed system handles is the intensity of a physical property at a set of points in space. This set of points is generated through information that was passed to the platform as part of the physical resource requests that the smart services make. For example, a service may request a specific amount of illumination at the user's current location; the user's location will be used as such one point of the phenotype. It is also possible that depending on the request, one request may require multiple points in space to be evaluated.

Illumination Simulation In its current iteration, the system supports only the management of illumination as a physical property. To estimate the effects that devices have in regards to illumination, two simulation approaches were pursued:

- an approach based on ray-tracing,
- an approach based on interpolation.

Simulation Using a Ray-Tracer For the approach based on ray-tracing, a custom ray-tracer was developed that utilizes attenuation fall-off information from the light sources. This ray-tracer was based on the octree implementation that was used for the indoor location system described in chapter 4.

Performing ray-tracing in real time is a computationally intensive task. Although it may be possible to perform real time ray-tracing with commodity and specialized hardware that is available as of early 2016, it is still a challenge. Furthermore, relying on real time ray-tracing would in turn mean

[3]A temperature setting is present in simulated annealing. Depending on this temperature setting, worse solutions may be adopted as the base for a new search.

that the cost of deployment for the proposed home service platform would rise, as specialized or expensive hardware would become necessary.

To tackle the above difficult points, a non real time ray-tracing approach that needs an extra pre-computation step was pursued. A simple physical fact makes this possible: illuminance (the intensity of light) from a point source falls off according to the inverse square law.

$$I = \frac{P}{4\pi r^2} \tag{2.9}$$

For a given scene that involves illumination devices, assuming that the emitted amount of power of the light sources is known, the amount of light that reaches a specific point in the scene from each light source can be computed.

However, this model of a perfect point light source does not represent reality very well. First, reflection from surfaces is not taken into consideration. Secondly, the remaining geometry of the scene is also not taken into consideration. This means that this simple model does not account for blocked illumination, as is the case when a scene object blocks the direct line-of-sight path from a light source to a point in space.

The ray-tracer used in the system models light sources as a set of rays. The number of rays used for each light source is customizable and using typical settings a light source is represented with as much as a few hundred rays. The energy of the illumination source is split evenly among these rays. These rays bounce freely in the scene and collide with objects. The rays reflect on surfaces that have an associated absorption coefficient. Furthermore, each ray is intersected with the leaf nodes of the octree, and attenuation information in the form of $\frac{1}{4\pi r^2}$ is recorded. By multiplying this attenuation information with the energy that is carried by a ray and summing the contributions of all the rays of a light source, it is possible to estimate the effects that a light source has on the given octree node.

Another point that must be addressed is the quantization of space. Although space as perceived by humans seems continuous and not quantized[4], the system must perform quantization of space, since otherwise the number of points for which light attenuation information will have to be computed would be infinite. The quantization of space is performed through the use of an octree. For each leaf node of the octree, light attenuation information from all the sources present in the scene is stored. Depending on the size and the initial creation parameters of the octree, the granularity of space quantization can be controlled. For example, during the experiment session 15cm^3 leaf nodes were used.

[4]The physics regarding quantization of space has yet to reach a definitive conclusion.

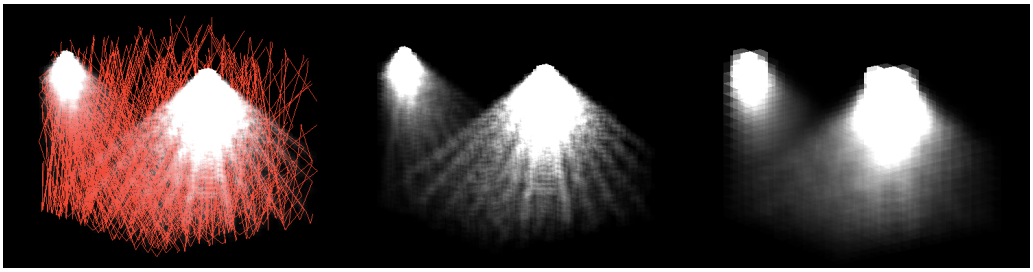

Figure 2.2: Ray-Tracer Visualization for 2 Light sources. Left: Rays Visualized. Center: Without Rays. Right: With Smoothing Pass

With this infrastructure in place, given a genotype g it is now possible to estimate the illumination at a given point z as follows:

- find the bounding leaf node for the point z in the octree,
- retrieve the attenuation information for the light sources that illuminate that node,
- depending on the settings of each light source in g, calculate the power output of each light source that reaches that node,
- sum the amount of illumination reaching that leaf node.

A visualization of the results of the ray-tracer can be seen in Fig. 2.2.

Due to the relatively low number of rays used to model light sources, discrepancies in the light reaching adjacent leaf nodes are apparent. In an effort to mitigate these discrepancies, a special smoothing pass is performed after the light attenuation information has been computed. In this smoothing pass, leaf nodes exchange a small amount of energy with neighbouring nodes. The results of the ray-tracing before and after the smoothing pass can be seen in the middle and right pictures of Fig. 2.2. Clearly, the results after a smoothing pass represent reality closer.

Interpolation Based Simulation Due to concerns regarding the accuracy of the ray-tracer, an alternative method for illumination simulation based on interpolation was also developed. This interpolation based simulation interpolates between three variables of the illumination device:

- brightness setting,
- distance from source and
- angle from source.

Figure 2.3: A Philips Hue Light with its Accompanying Fixture.

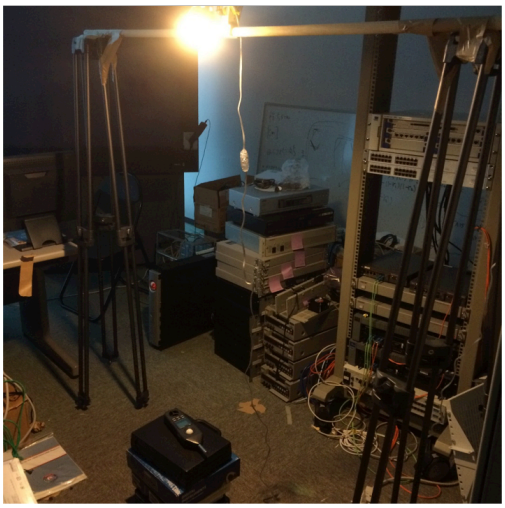

Figure 2.4: Environment Used for the Modeling of Philips Hue Lights.

These three variables are what ultimately dictate the amount of illumination reaching a point in space from a light source. For experimental purposes, the popular Philips Hue lamps were modeled. These lights were used in combination with a light fixture as seen in Fig. 2.3. Illumination measurements were taken at distances of 0.25m, 0.5m, 0.75m, 1m, 1.25m and 1.5m, at an angle of 0°, 30°, 45°, 60° and 90° and with brightness settings of 100%, 75%, 50% and 25%. The environment in which these measurements where taken in a dark environment without any other illumination source. This environment can be seen in Fig. 2.4.

During simulation, the brightness setting of the device is known (this information is part of the genotype g) and the angle as well as the distance from the point of interest is calculated. Using these three parameters, three successive linear interpolations are performed sequentially; first in terms of angle, then in terms of distance and finally in terms of brightness. ”

Through this interpolation procedure certain assumptions and limits are in place. The first limit that is enforced is that if a point lies at an angle that is greater than 90°, it is assumed that no significant amount of illumination reaches that point, thus the illumination for that point is set to zero. Secondly, if a point lies at a distance that is greater than 1.5m away from the light source, this distance is successively halved until the distance falls into the $\left[0\text{m}, 1.5\text{m}\right]$ range. The interpolation procedure is carried out with this new shorter distance and an illumination estimation is computed. The root of this estimation is then successively computed as many times as the

original distance had to be halved. For example, for a point that lies 4m away from a light source, the interpolation estimation will use a distance of 1m (the original distance was split in half two times). Supposing that the result of this interpolation is 625lx, the illumination that reaches the original point which is 4m away is only 5lx. This approach is justifiable through the inverse square law as explained in the previous section.

In the experiments section of this chapter, data obtained through the use of both of these simulation methods will be compared and contrasted with actual measured values.

2.3.7 Evaluation

The final step of a search iteration after the exploration and simulation steps is the evaluation of the resulting phenotype p in terms of fitness. For this step, a set of objective functions and an overall fitness function are used. Their characteristics and the rationale behind their design is explained in this section.

Objective Functions The requests regarding environmental resources received from a smart service are translated into one or more objective functions used to evaluate the fitness of a candidate solution. These requests are interpreted as a set of upper and/or lower bound targets that have to be achieved in order to fulfill them. The equation used for a target that has a lower bound intensity that has to be fulfilled can be seen in Eq. 2.10. The objective function $Over$ computes the degree of satisfaction achieved by the intensity of a physical property val for a given target $target$ which is subject to maximization.

$$Over(val, target) = \begin{cases} -\frac{(val-target)^2}{target} & \text{if } val < target, \\ \log_{target}(\frac{val}{target}) & \text{if } val >= target. \end{cases} \tag{2.10}$$

For this equation, assuming that $val < target$ the result is going to be negative, and it is the square difference of the current intensity and the target, scaled by the actual target. The use of square difference ensures that the more a target is being violated its satisfaction degree will further rapidly decline. In contrast, if $val >= target$ it means that the target is fulfilled. For a fulfilled target the use of a logarithmic function gives diminishing returns.

For a target that has an upper bound, i.e. the intensity of the physical property must be less than the provided target, a similar objective function is used (Eq. 2.11).

$$Under(val, target) = \begin{cases} \log_{target}(-\frac{val-2*target}{target}) & \text{if } val <= target, \\ -\frac{(val-target)^2}{target} & \text{if } val > target. \end{cases} \quad (2.11)$$

Finally, the objective functions scale their reported result appropriately based on target. This allows for the direct comparison among different objective functions and gauge their degree of satisfaction in absolute terms.

Fitness Functions The final piece of the puzzle for the fitness evaluation of a candidate solution is the fitness function used to aggregate the results of the objective functions. These fitness functions are again subject to maximization.

Two fitness functions were implemented, with the first one being just a simple summation of the objective functions:

$$Fit(Obj_1, Obj_2, ..., Obj_n) = \sum_{i=1}^{n} Obj_i \quad (2.12)$$

Due to the way in which objective functions are designed, unsatisfied targets will impact the overall fitness score in a very pronounced negative fashion. This fact forces the search towards solutions that avoid heavily violating any upper or lower bounds, thus emphasizing compromise among conflicting objective functions.

The second fitness function improves upon the simple summation fitness function by also taking into consideration the number of devices that are used as part of the candidate solution. If the calculated sum of all objectives is positive it is a very good indication that all objectives are either met or if unmet objectives do exist they are violated only slightly. For such cases the overall fitness receives a bonus score depending on the number of devices it does not use and a configurable factor a. For cases where the sum of the objective scores is negative, the fitness is penalized depending on the number of devices that are being used and the same factor a. The equation for this new fitness function can be seen below (Eq. 2.13).

$$\text{Setting } S = \sum_{i=1}^{n} Obj_i$$

the new fitness function becomes:

$$Fit(Obj_1, Obj_2, ..., Obj_n) = \begin{cases} a * S * (used + 1) & \text{if } S < 0 \\ a * S * (unused + 1) & \text{if } S >= 0 \end{cases} \quad (2.13)$$

This fitness function allows for potentially worse solutions in absolute terms to be evaluated more favourably to better quality solutions which use more devices. For example assuming a currently optimal solution with objective sum S and fitness F that utilizes n devices and a new candidate solution with objective sum S', fitness F' that uses $n-1$ devices, for the new candidate solution to be registered as the new best solution then $F' > F$ must hold true, which leads to $s * a\frac{(n+1)}{n} < S'$ (with $S, S' < 0$). This means that the objective scores of the new candidate solution may be at most $a * 1/n^{th}$ worse than those of the current best solution. Similar logic applies where $S, S' > 0$. This margin where a worse solution that uses fewer devices will be preferred over the current best solution can be adjusted with the scaling factor a.

The current fitness function assumes that all devices involved in the simulation consume the same amount of power. Thus, a reduction of the number of used devices will lead to a direct reduction in power consumption. However, scenarios where all devices have the same power consumption are quite uncommon. Should power consumption information be available, a fitness function that combines illumination estimation and power consumption can be constructed in a similar fashion. Such a fitness function can act as the building element for a system-wide energy saving policy.

2.3.8 Meta-Heuristic Search Strategies

Having explained the procedures taking place during each search iteration (exploration, simulation, evaluation), the search metaheuristics used in the current iteration of the platform are explained.

Currently two variations of multi-objective hill climbing have been implemented: one with restarts and one without restarts. The algorithm for these two variations can be seen in Alg. 2 with the only distinction that the *shouldRestart*() check always returns false for the hill climbing variant without restarts.

The hill climbing algorithm as seen in Alg. 2 holds the current genome in variable g and performs a mutation on it which results into a new genome g'. The phenome p is produced through the use of a *gpm* function and its fitness is stored in variable fit'. If this new fitness is the best fitness encountered so far, the genome is stored in variable $bestg$. Also if this new fitness is better than the current fitness fit, the current genome g is replaced by g', ready to be used as the base of new mutations in the next iterations. For the hill climbing algorithm without restarts $fit = bestfit$ is always true.

With a variety of mutation operations, simulation methods (*gpm* functions) as well as a set of objective and fitness functions, even a simple meta-

Algorithm 2 Hill Climbing with Restarts

```
1: function HillClimbing
2:     g, p, g', fit, fit', bestfit, bestg ← init()
3:     while terminate() == false do
4:         if shouldRestart() then
5:             g' ← Randomize(g)
6:             fit ← −∞
7:         else
8:             g' ← Mutate(g)
9:         end if
10:        p ← gpm(g'), fit' ← evaluate(p)
11:        if fit' > bestfit then
12:            Bestg ← g', bestfit ← fit'
13:        end if
14:        if fit' > fit then
15:            g ← g', fit ← fit'
16:        end if
17:    end while
18:    return bestG
19: end function
```

heuristic search strategy such as hill climbing can produce results of high quality if configured properly. This will be demonstrated in the experiments section of this chapter.

2.4 Environmental Resource Request API

In a similar fashion to the API used for the control of devices in chapter 3, smart services have access to a higher level API with which they can make requests regarding environmental resources. The use of this API simplifies service development and frees the developer from the error prone task of indirectly trying to control physical properties of space by haphazardly operating devices in the home.

In order to fully specify the desired features for a physical property, the following pieces of information are necessary: the physical property type (illumination etc.), the area over which the request is to be enforced upon, and finally the intensity of the physical property. With this information available, the home services can make requests such as "set the illumination to 300 lx at the position of the user", "set the temperature of the room at

```
@Override
public void run() {
  Illumination illumination = (Illumination) platform.getPhysicalResource(
      ↪ ResourceType.ILLUMINATION);
  illumination.set(this, new AreaDynamic(theUser, 3), IntensityType.
      ↪ getLowerboundIntensityType(300));
  goToSleep();
}
```

Figure 2.5: Controlling Physical Properties With Two Lines of Code

25 °°C " etc.

The `getPhysicalResource(ResourceType type)` method call can be used to get a reference object for the given type of environmental resource. Each resource exposes appropriate methods for controlling the physical property or getting an estimation of its intensity at a given point in space.

The `set(...)` method for controlling a physical property requires exactly three arguments: a reference to the service that is making the request, a specification of the area where the setting is to be enforced upon, as well as the desired intensity.

To express an area, an instance of the `Area` class is necessary. To create such an instance, the coordinates of a point in space (an "anchor" point) as well as a distance specifier are necessary. In its simplest form, an area is specified as static, i.e. an area that never changes over the lifetime of the request.

The `AreaDynamic` subclass of `Area` can be used to express information for areas that can change dynamically. To create an instance of this class, a reference to an object that generates motion events as well as a distance specifier is necessary. Instances of the `User` class generate such motion events. With this combination, an "area changed" event is generated for each user motion event, which are then in turn used by the platform to update its calculations. The user thus acts as a moving anchor point.

To express the desired intensity there are several options: request a lower bound, an upper bound, a range of acceptable intensity or finally an exact intensity setting. This information is captured by an instance of the `IntensityType` class. For each bounded or exact intensity setting, the illumination subsystem internally generates two competing objectives which will take part in the simulation that will follow.

Putting this all together, it is now possible for a home service to make requests regarding environmental resources using only two lines of code, as shown in Fig. 2.5. This code generates a physical resource request that will be handled by the illumination subsystem. Note that there is no guarantee

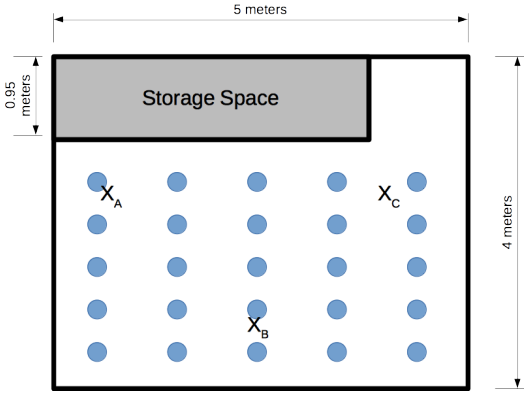

Figure 2.6: Top View of Experiment Room and Lamp Placement

Figure 2.7: A View of the Experiment Room

that the intensity specification will be honored, especially if this physical request conflicts with requests made by other services.

2.5 Experiments

2.5.1 Experiment Setup

The experiments of this section were conducted in a room with 25 Philips Hue lamps installed. A top view of the room can be seen in Fig. 2.6 and the actual room can be seen in Fig. 2.7. The room size 4m by 5m, with a closet space taking up space and the entrance located at the top right. The 25 Philips Hue lamps were affixed close to the ceiling at a height of 2.3m above ground.

Starting from the leftmost column and going down, the lights are sequentially assigned numbers. The lights of the first column are assigned numbers from 1 to 5, the lights of the second column are assigned the numbers 6 to 10, all the way to the rightmost column where the lights are assigned the numbers 21 to 25. This index assignment uniquely identifies a light in the scene.

2.5.2 Illumination Conflict Among Multiple Services

Scenario and Configuration

For the first experiment of this section, a case where 3 services make conflicting requests regarding the illumination of a room at the same time. Three services A, B, C make requests for more than 300lx, less than 20lx and more

than 300lx at points X_A, X_B and X_C respectively. The placement of these points can be seen in Fig. 2.6.

As a possible story that would justify a scenario like this, it is easy to imagine that a person at point X_A is performing some task that requires illumination, another person that moves through the room is currently at point X_C (thus again requiring illumination in order to navigate the room safely) and finally a third person at point X_B that wants to rest, thus requiring a very low amount of illumination. Although the requests from services A and C do not directly conflict with each other (since both services require a lower bound for illumination intensity), both of these requests conflict with request of service B that requires a very low upper bound for the illumination at the vicinity of its user.

For this scenario, four different configurations of the local search engine were tested:

1. Hill Climbing and Optimized Binary-Search Mutator (NO),
2. Hill Climbing and Random Mutator (NR),
3. Hill Climbing with Restarts and Optimized Binary-Search Mutator (RO).
4. Hill Climbing with Restarts and Random Mutator (RR),

The termination criteria for all of the above cases were either the completion of fifty thousand search iterations or a total execution time of one second, whichever comes first. The restart strategy used for the variants with restart was the lapse of a thousand iterations without improvement on the current solution.

As the *gpm* function, the simulation approach based on the ray-tracer was used, and the sum of the scores produced by the objective functions was used as the overall fitness function.

Comparison With Reality

The four configurations were tested using the same random seed and they produced the solutions seen in Fig. 2.8. The numbers in these images represent the brightness setting for the hue light that is located at the current position. The brightness settings have a range from 0% to 100%.

Looking at the solutions produced by the four search configurations, it becomes immediately apparent that three of the four search configurations converged. The “NR”, “RO” and “RR” search configurations converged in the sense that their solutions use the same devices (the lights numbered 1, 2,

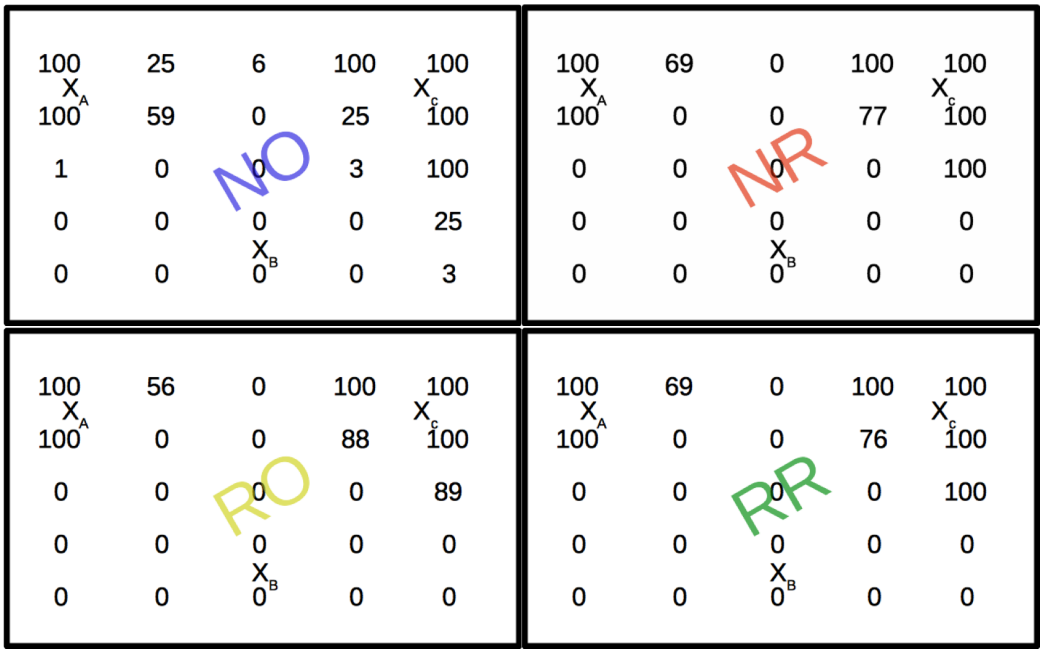

Figure 2.8: Solutions for the Four Configurations. Top Left: No restarts, Optimized mutator. Top Right: No restarts, Random mutator. Bottom Left: Restarts, Optimized mutator. Bottom Right: Restarts, Random mutator.

6, 16, 17, 21, 22 and 23). Furthermore, the "NR" and "RR" configurations reached what practically is the same solution; only the brightness setting for light no. 17 is different (77% to 76%). The lights used in these three solutions are lights that are very close to points X_A and X_C, where a lower bound of illumination is necessary to fulfill the requests of services A and C. All other lights are turned off, in order to accommodate for service B that requested a dark environment.

The solutions proposed by the four search configurations were then compared to the actual illumination measured at the scene. The results can be seen in Tab. 2.1. The estimated illumination for points X_A, X_B, X_C had a tendency to be higher than the actual measured illumination. This discrepancy can be attributed to the limitations of the ray tracer.

Regardless of the accuracy of the ray tracer, the four search configurations lead to sensible solutions that demonstrate the viability of conflict resolution for environmental resources. Using the estimated lux intensity as an indication, none of the actual requests were able to be fulfilled. However, neither of these requests were violated to an extreme degree and are in fact just barely violated; the dark setting for service B is estimated to be barely above 20lx (and in reality it was just 14lx) whereas the bright settings of services A

Table 2.1: Showcase Results per Search Configuration

Search Strategy	Measured Lux	Estimated Lux	Fitness
NO	186, 14, 218	296, 20.1, 258	-0.01974
NR	194, 14, 226	291, 20.5, 279.5	-0.0062
RO	187, 14, 222	286, 20.7, 281	-0.00685
RR	195, 14, 227	291, 20.5, 279	-0.00622

and C where estimated to reach approximately 90% of the required 300lx intensity.

Solution Quality and Performance Considerations

In order to evaluate performance and the quality of solutions produced by the four search engine configurations, the experiments of the previous section were repeated 200 times using the same starting seed for the random number generator.

Performance Considerations The results from these repeated experiments can be seen in Fig. 2.9 and Fig. 2.10. These figures show the best, worst, average and median iteration count and time (in milliseconds) of when the best solution was reported. This is not the execution time of the four configurations; the search configurations executed until one of the two termination criteria was met (one second of execution time or 50000 search iterations).

In terms of absolute speed, the "NO" search configuration (no restarts, optimized binary-search algorithm) produced the fastest results. With the best solution reported at just 79ms into a run, and with a worst case scenario of only 339ms, it surpasses the other three search configurations easily.

In stark contrast, the "NR" search configuration (no restarts, random mutator) took on average 302 seconds to report its best solution, with a worst case scenario of 714ms. This can be attributed to the unpredictable behaviour of the random mutator.

Regarding the search configuration variants that used restarts("RO", "RR") , the average time of reporting their best solutions was higher than the search configurations without restarts ("NO", "NR"). The "RO" and "RN" search configurations had an average time of 490ms and 542ms respectively. Furthermore, in their worst case scenarios, both search configurations exhausted the upper limit of execution time of 1 second.

The higher best, average and worst times of the reported solution for the search configuration variants with restarts show the impact of the restart

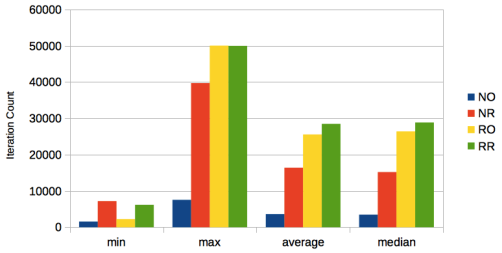

Figure 2.9: Performance Evaluation - Iteration Count

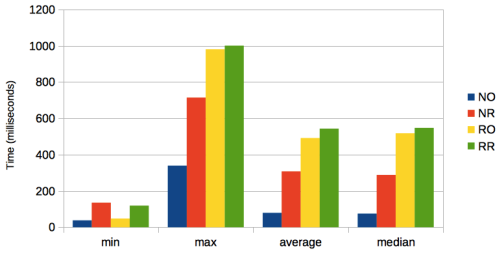

Figure 2.10: Performance Evaluation - Execution Time

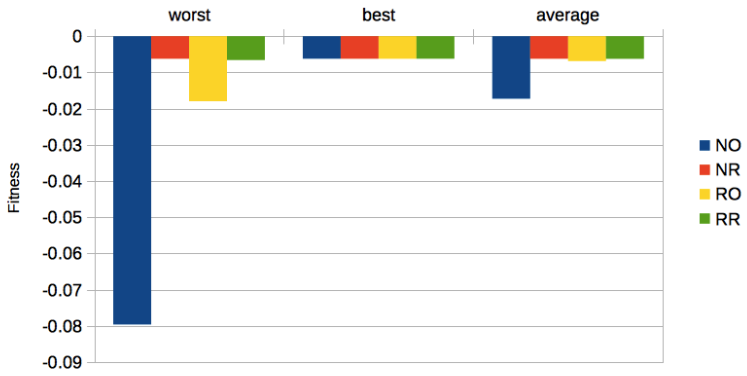

Figure 2.11: Solution Quality of the Four Search Configurations

component in terms of execution time. However, these higher times also indicate that it is a common occurrence for a better solution to be found after one or more restarts.

Solution Quality The results of these 200 runs in terms of absolute fitness can be seen in Fig. 2.11. The first perhaps initially surprising observation is that the search configuration variants with the random mutator perform significantly better than the variants that use the optimized binary-search mutator. It might be natural to assume that binary-search should outperform randomness but this result demonstrates the power of randomness in local search algorithms: the ability of the search to escape local maxima and find solutions with better fitness.

More specifically, the worst case and average fitness reported by the "NO" configuration was -0.0796 and -0.0172 respectively, values that are far worse than even the worst case fitness produced by the configurations with random

Table 2.2: Unique Solutions and Standard Deviation

Algorithm	Unique Solutions	Standard Deviation
NO	162	0.012 012
NR	3	0.000 002
RO	71	0.001 237
RR	78	0.000 054

mutators. Using restarts, the "RO" configuration improves over the "NO" results (-0.0179 worst case and -0.0068 average), but still cannot match the configurations that use the random mutator.

The absolute best performance in terms of fitness solution was observed by the "NR" (no restarts, random mutator) search configuration, whose best, average and worst performance are practically indistinguishable at -0.0062, with differences appearing in the sixth decimal digit. In fact, the "NR" configuration even outperformed the "RR" configuration.

Although the worst, average and best case scenario can give a quite accurate picture of the way these four configurations behave, the number of unique solutions reported in these 200 runs and their standard deviation in terms of fitness were also calculated. These results can be seen in Tab. 2.2.

Astonishingly enough, in 200 runs the "NR" configuration only produced 3 unique solutions, that differed by one percentage of illumination in one or two genes, thus being overall the most consistent search configuration overall. The tendency of the "NO" configuration to get stuck at local minima is readily apparent, as it produced 162 unique solutions with relatively high standard deviation among these solutions. Finally, although the "RO" configuration produced fewer unique solutions than "RR" (71 against 78), the standard deviation of the latter is far smaller, thus being more consistent.

Remarks From the results presented in the previous sections, two points worth noting arise.

The first point is the accuracy of the ray tracer used as a *gpm* function. As explained in Sect. 2.5.2, the ray tracer currently overestimates the illumination at given points. Although more accurate predictions would be preferable, its current behaviour is better than an alternative where underestimation would occur. The human vision has great adaptability, and variations in bright environments tend to go unnoticed by the user. In contrast, small variations in illumination in darker environments are far more pronounced and immediately perceived by the user.

The second point is about the proper selection of a search configuration.

Table 2.3: Illumination Request Details

Case	Intensity Type	Illumination	
		Estimated	Measured
1	$200 <$	463.8	419
2	$100 <$	408	382
3	$[200, 300]$	300	281
4	$[20, 30]$	30.7	23

For our purposes, the case for selecting the NR algorithm can be made, based on the simple fact that it produced the most consistent results in terms of solution quality, with an exceptionally low standard deviation. Nevertheless, the average execution time for this algorithm is close to 500 milliseconds. Should a faster response time be of utmost importance, the use of the NO search configuration is advised.

2.5.3 Evaluation of Optimized Fitness Function for Single Requests

To evaluate the performance of the optimized fitness function, an experiment where a single service would submit a physical resource request with various intensity bounds was set up. Four unique cases were considered, the details of which can be seen in Tab. 2.3. The center of the area requested was located at point X_C. During the local search, the optimized fitness function was used. The search configuration used was an "RR" configuration (restarts, random mutator). Finally, the estimated illuminations calculated by the `gpm` function were compared to measurements taken at the actual scene. Doubts about the accuracy of the ray tracer based *gpm* approach led to the development of the interpolation-based *gpm* function, as introduced in Sect. 2.3.6.

Cases 1 and 2 demonstrate the effectiveness of the new fitness function. Since the intensity type for these two cases only has a lower bound, to maximize the illumination objective functions it would be enough to switch all lights present in the room to maximum brightness. However, such a setting would be wasteful in terms of power consumption and would illuminate spot X_C with far more intensity than requested.

With the use of the new fitness function, the local search algorithm was guided to a high quality solution that utilizes only 8 and 6 out of the 25 available lights for case 1 and 2 respectively. Furthermore, the illumination was measured at 419 lx and 382 lx, an illumination level that handily fulfills

Table 2.4: Performance and Solution Quality Characteristics

Case	Avg. lx	Reported Time in ms (Average / Worst)	Unique Solutions
1	463	215.6 / 892	58
2	409	217 / 783	59
3	299	385 / 1000	491
4	30.6	441 / 1000	500

the requested illumination.

In cases 3 and 4, the platform still produced solutions that satisfied the bounded illumination conditions requested. For a bounded illumination condition it is enough to generate two competing objective functions that take part in the simulation. However, due to the way the objective functions are designed, the objective function representing the lower bound (20 lx in case 4) slightly overpowers the objective function of the upper bound (30 lx in case 4). Thus, the platform always generates solutions that hover slightly above the upper bound of a bounded condition. This behaviour can be exploited to achieve very specific illumination conditions.

Finally, the use of the optimized fitness function as *gpm* led to results that matched the reality better than the ray-tracer based *gpm*. For illumination intensities of a few hundred lux, the interpolation based *gpm* function produced estimations that were very close to the actual measured values. The only significant deviation appeared at lower intensities (case 4), where the illumination was estimated at 30.7lx contrary to an actual measurement of 23lx at the scene.

Overall, the subject of an effective *gpm* function for illumination simulation is still open to further improvements.

The solutions of the cases introduced above can be seen in Fig. 2.12, 2.13, 2.14, and 2.15.

Solution Quality and Execution Performance

In order to evaluate the consistency and robust performance of optimized fitness function as well as the interpolation-based *gpm* function, the experiments of the previous section were repeated 500 times. The aggregated results for each case were compared in terms of estimated illumination, average execution time as well as number of unique solutions. The aggregate results of these 500 runs can be seen in Tab. 2.4. Moreover, the average solution for each case can be seen in Fig. 2.16 with the average solution for case 1 on the top left.

Figure 2.12: Sample Solution (Case 1)

Figure 2.13: Sample Solution (Case 2)

Figure 2.14: Sample Solution (Case 3)

Figure 2.15: Sample Solution (Case 4)

First, for cases 1 and 2, the minimum illumination conditions of 200 lx and 100 lx respectively are fulfilled. Of particular interest are the number of unique solutions generated; 58 and 59 unique solutions respectively. What this table fails to convey is that, in case 1, *the top 11 solutions amounted for 446 runs out of the 500 runs*, with these 11 solutions using the exact same set of devices, with negligible variations in settings (for example, the brightness of a device might be set at 99 or 98 instead of 100). Furthermore, the top solution was reported 342 times out of 500 runs, which incidentally is the solution depicted in Fig.2.12. From these observations, it can be said that the search produced the same output roughly 90% of the time. Similarly, in case 2 the top 6 solutions where reported a total number of 413 times out of 500 runs, with the top solution reported 346 times.

The average solutions for cases 1 and 2 as seen in the top row of Fig. 2.16 show the consistency with which the illumination subsystem identified the most appropriate devices to use. In both cases, the closest devices to the point of interest C were selected consistently as part of the most promising solution, and their brightness setting was set to the highest brightness possible to maximize their illumination contribution. This "core area" is highlighted with a green dashed line.

The situation changes for cases 3 and 4, as reflected by the high number of unique solutions reported. Starting with case 3, the top solution was reported only 4 times and utilized only 4 lights. 90% of the solutions reported made use of 5 or 6 lights, with the remaining solutions using 7 lights or more. A more detailed look at the average solution revealed some patterns: the lights of the first, second (with the exception of the fourth light) and last column where switched off for the majority of the time. The lights of the third and fourth column had the most diverse range settings, actively contributing to the solutions. However, with the presence of an upper illumination bound of 300 lx not all lights were set to their maximum brightness; as demonstrated in cases 1 and 2 that would lead to illumination of approximately 400 lx. Case 3 still exhibited a "core" of commonly used devices in the solutions it reported. However, in contrast to cases 1 and 2, the average brightness setting was significantly lower than 100%.

Case 4 was by far the most idiosyncratic case, with no solution reported more than once. The solutions utilized anywhere from 4 to 8 devices to fulfill the request, but this time almost any device in the room was used, with no discernible pattern for more commonly used devices. However, the farther a light was located from the point of interest the higher its brightness settings tended to be. Specifically, almost all the lights that contributed the most in the solutions for cases 1,2 and 3 have an average illumination setting of less than 10. Conversely, lights that are farther away tend to average

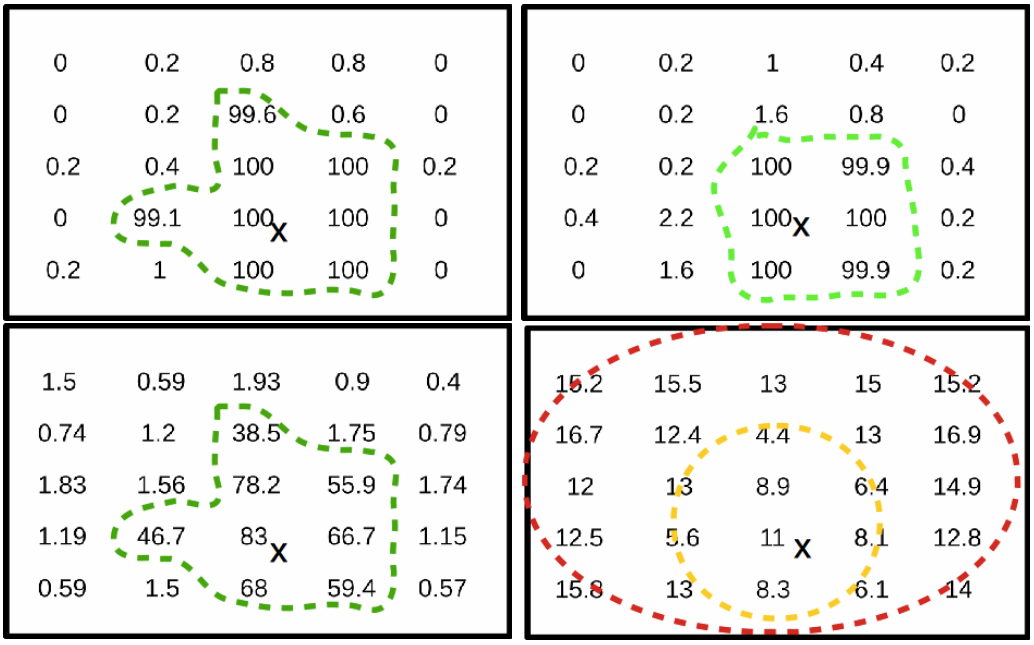

Figure 2.16: Average Solutions for Cases 1-4 (500 Runs)

to a brightness setting of 14 and upwards, as can be seen from the average solution. This is expressed in the lower right image of Fig. 2.16 with an inner yellow circle of devices that have consistently lower brightness settings than the devices marked with the outer red circle, that tend to have considerably higher brightness settings the farther away they are located from point X_C.

The variation in results suggests that, even with a fitness function that suppresses the number of devices used, this type of request *is not heavily constrained*. If the fitness function had power consumption data available for use, it would be expected that, due to their higher power consumption, lights located farther away with high brightness settings would be evaluated unfavourably compared with lights closer to the point of interest with lower brightness settings. Such a fitness function would again drive the search with better focus, reducing the variability of the results.

In terms of computational performance, the best solutions for all four cases were reported on average in less than 500 ms of execution time. These results are in line with the results of the previous section, suggesting that an upper bound of execution time set at 1 second is enough to enable the system to produce good quality solutions.

2.5.4 Illumination Conflict for Moving Target

In this last experiment section, the following scenario was considered: a user starting from point X_A in the room moves slowly towards point X_B with a speed of 0.5m/s. At the same time, another user is resting at location X_C. Due to the lack of an operational location system at the moment, the motion of the user was simulated.

For the user in motion, a request for a lower bound of 300 lx is made. For the user at rest a request for an upper bound of only 30 lx is made. Every second the position of the moving user is updated. This new location information is used to update the calculations for this room.

A frame-by-frame development of the results produced by the system for this experiment can be seen in Fig. 2.17. Each frame represents a solution that corresponded to a change of the location of the moving user (denoted by 'X' in the frames).

With the exception of a very strange first frame, the results depicted in the remaining frames are of very good quality. Only the closest lights to the user are utilized, with every other illumination source turned off. In the fifth frame, the second light of the third column of lights is only partially lit (brightness set to 65), a very interesting decision on the part of the illumination subsystem. This light is too close to the user who is resting; further increase in the intensity of that light source would have adverse effects for that user. The estimated illuminations for this frame are 264 lx and 30.7 lx for the moving and resting user respectively. For all other frames, the illumination estimation for the user at rest where hovering around the 25 lx mark, whereas for the moving user a typical illumination value was at 250 lx, with frames 6 and 7 dipping as low as 200 lx and 212 lx respectively.

This experiment was repeated, this time with a much slower speed of 0.1 m/s. The same first frame persisted, leading us to believe that some implementation detail regarding the initialization of the simulation is amiss. Nevertheless, after the first position update, the algorithm produced again very smooth transitions as the user traveled from point A to B. 45 frames were produced as a result, with solution characteristics similar to the first, faster run.

The results of this experiment are a demonstration that, as long as appropriate objective and fitness functions are used to guide the search, using a random mutator at the core of the search algorithm can produce high quality results in practically real time.

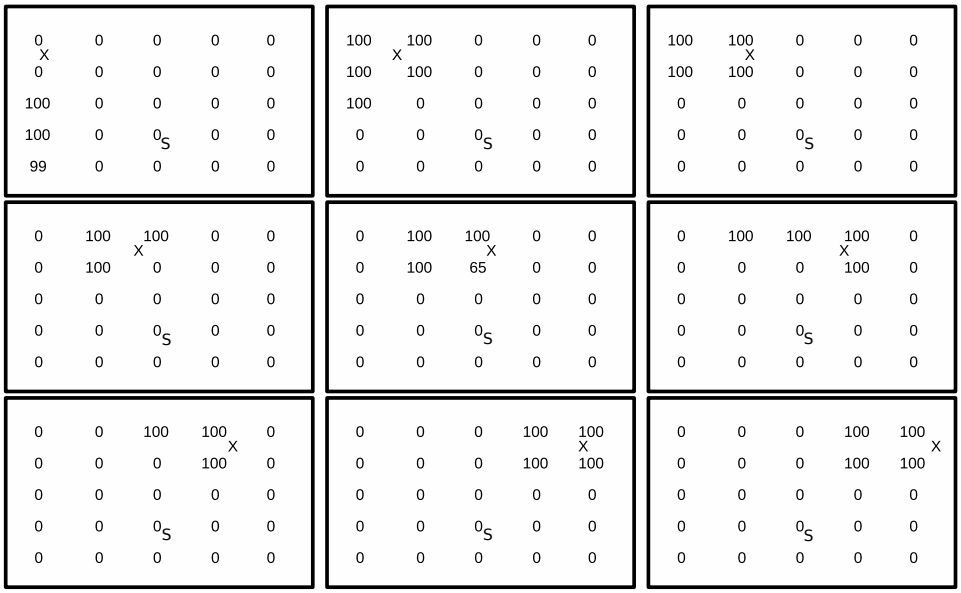

Figure 2.17: Frame-By-Frame Results for the User in Motion

2.6 Conclusions and Future Work

In this section the topic of managing environmental resources was discussed. The proposed home service platform takes the novel approach of integrating the management of such resources as part of its functionality. Through a high-level API, smart services can now make direct requests regarding physical properties of space, leaving the platform to decide the best combinations of devices and settings that are necessary to fulfill them.

The proposed approach of delegating environmental resource management to the home service platform has two advantages. First, it simplifies smart service development. Smart services no longer have to do any guesswork and try to operate devices whose effects on the environment are hard to estimate. This becomes a task for the service platform itself. Moreover, the services can now make complex requests regarding moving targets very easily, provided that data from an indoor location system is available. Secondly, conflicts over environmental resources can now be detected and resolved, using space-based or intensity-based conflict resolution.

The proposed home service platform is designed with support for illumination, temperature, humidity as well as sound and noise levels as environmental resources. However, in its current state, only management of illumination as a resource has been implemented.

As the number of devices present in the house rises, the problem of deciding appropriate settings for devices in order to fulfill environmental resource

requests becomes a combinatorial optimization problem. This research proposes the novel idea of utilizing global search algorithms in combination with simulation of devices as a solution to this problem. As demonstrated in the experiments section, the illumination subsystem is capable of producing very good quality solutions within a second of execution time, a time-span that is short enough to be considered real time for our purposes.

As future work, focusing on the rest of the physical properties of space should be a priority. For each property, a suitable device simulation method becomes necessary, but the simulation of device effects on the environment is a balancing act between performance and accuracy. Although a more accurate simulation is desirable, the simulation should be very fast, as it is expected to be executed thousands of times in a second. Furthermore, in cases where the search space is limited, more accurate simulations and an exhaustive search could yield comparatively more favourable results.

Sensor feedback is another desirable feature that should be introduced into the system. Using sensor feedback, the system should be able to further refine the quality of the solutions it produces. The difference between a projected intensity value of a physical property and the simulated expected intensity can act as a bias factor during subsequent search iterations. After a given solution has been applied, should this bias value exceed certain thresholds the system should trigger a new search instance and adjust solution evaluation based on this newly discovered bias that was the result of sensor feedback.

The objective and fitness functions could be another area of improvement. Objective functions currently do not take into consideration how users perceive illumination. An objective function that mimics human perception could help discover even more desirable solutions for users. Alternative fitness functions that combine heterogeneous information from multiple objectives can also be pursued, with the most apparent candidate a fitness function that takes into consideration device power consumption.

On the topic of objective and fitness functions, more elaborate multi-objective optimization approaches for heterogeneous objectives should be considered. In future revisions of the system, information such as power consumption, user and service priority and others may be taken into consideration during the evaluation of a solution in order to produce even more sophisticated results. However, such a task should be undertaken with the greatest of caution: there are no clear-cut answers for automatically evaluating a solution based on heterogeneous objectives, and it is possible that expert knowledge bias may be introduced in the system.

Lastly, a qualitative evaluation of the system in terms of user satisfaction may yield significant insights for future improvement or reveal shortcomings

of the current implementation.

Chapter 3

Managing Devices as Resources of the Home Environment

In this chapter, the details and inner workings of the proposed home service platform are presented. More specifically, related research, platform architecture, device modeling, primitives used during smart service development, API design and example usage, an experimental section as well as conclusions are introduced, in that order. This work was published in [60].

3.1 Related Research

In this section, related research regarding home service platforms is presented. Special attention is given to the problems of conflict detection, conflict resolution, device modeling, interconnectivity, as well as overall platform design. Before any other related works are presented, a literature review of the state of the art for embedded middleware platforms in the smart home can be found in [67].

3.1.1 Consumer Oriented Platforms

The first step regarding related research is to examine the characteristics of home service platforms that are soon to be released to the market such as Apple's Homekit[2] and Home OS[20],[19] from Microsoft's research division.

Regarding the logical representation of home appliances, both of these platforms model devices in terms of their functionality. In Microsoft's solution, there is a clean separation between device connectivity and device functionality. The device connectivity layer addresses concerns regarding the actual network communication protocols such as searching for devices

and device availability, whereas the device functionality layer is responsible for creating APIs describing the functionality of a device, suitable for use by the home services. Device functionality is aggregated into *roles* with a device possibly exposing multiple roles. Similarly, in Apple's homekit, a device is referred to as an *accessory* and each such accessory offers *services*, with each service representing a concrete functionality of a device. Despite the disparity in terminology and nomenclature among these platforms, the core idea is the same: expose device functionality through APIs and abstract the underlying network communication details. Although there are differences in the way these APIs are employed and used, the proposed system follows this same approach.

The approach of these two platforms in regards to conflict detection and resolution differ significantly. The simplicity of Apple's homekit is staggering: the user is responsible for everything. Should a conflict among services occur, it is the responsibility of the user to resolve it, usually by terminating one or more of the conflicting services. The design philosophy of homekit seems to be closer to a remote control: trust the user to do the right thing. In contrast, Microsoft's HomeOS utilizes "Datalog" rules. These datalog rules combine a wealth of context information, such as location, time, service, user and others to define a context. Then, given this context access to certain devices can be granted, denied, or even ask the user to explicitly allow or deny the access to a device. A major drawback of this approach is that these rules must be created a priori. The burden of generating these rules falls on the user. Specific tools that are relatively easy to use even by inexperienced users have been created to facilitate this task. However, the behaviour of services in case of a conflict during runtime is not sufficiently addressed. In case of conflicts that somehow were permitted to occur by the datalog rules, a service running on HomeOS may be unexpectedly terminated at any time. This is a significant drawback that can severely harm the user's overall experience. A discussion regarding the state of the art of access control in the home can be found in [65].

Finally, the problem of obtaining and deploying new services is sufficiently addressed by both platforms with the introduction of a market place[19], a model that is similar to the way smart phone applications are distributed. However, due to the potential variety of smart services and the variations in users' houses, the task of verifying the correct operation of smart services is a daunting task. Furthermore, checking the compatibility of a service with the user's home environment is another task which should be further addressed. Simply comparing the devices present in the house with the devices that a service is expected to use may lead to an underwhelming experience.

3.1.2 Conflict Detection and Conflict Resolution

A significant body of work regarding conflict detection and conflict resolution already exists, and is also known under the name of "feature interaction"[16]. In this work, the two main methods for detecting and resolving feature interactions are pointed out: off-line techniques (or design-time) and on-line techniques (or run-time). Furthermore, hybrid approaches are also possible.

The related work that is perhaps the closest to the proposed system can be found in [38] and [69], where a system for the detection of conflicts on a device layer and an environmental layer is presented. This work correctly points out the significance of environmental conflicts.

A work regarding conflict detection based on ontology and semantic web can be found in [33]. Furthermore, formal verification approaches based on Linear Temporal Logic can be found in [73] and [39]. A common theme in these works is the representation of services as a script or a scenario, with clearly defined steps in a description language. This necessary precondition to apply linear temporal logic can be limiting when trying to implement smart services. The model checker used to detect and resolve these conflicts is Spin[10].

Depsys [46] is another contemporary home service platform that attempts conflict detection and resolution. This system provides install-time detection of conflicts by explicitly specifying the dependencies of each service. Dependency types include requirement dependency, name dependency, control dependency for the sensors, control dependency for the actuators, missing dependency and app interdependency. A device control request in this system includes information about the intended effect, an "emphasis" factor and a condition. The emphasis is used to denote operations of higher or lower importance, information that is used during conflict resolution.

Physicalnet[66] introduces the concept of access control for individual states and events. These access rights can be either read or write, also associated with a unique priority.

A classification of different feature interactions possible in the home environment can be found in [40]. In the same work, various methods for conflict resolution are also proposed but not implemented. These conflict resolution approaches are:inquiry to the user, priority of service, priority of user, priority of interface, priority of timing and meta priority.

Feature interaction for health care in the domestic environment is discussed in [62], where a graph-based approach for detection and resolution of feature interactions has been proposed.

Finally, a framework for describing feature interaction in ubiquitous computing environments can be found in [45].

3.1.3 Interconnectivity and Home Automation

A vast body of work regarding connectivity and interoperation in home automation exists.

One of the pioneering platforms for the home was Jini[27]. In jini, devices could register themselves as a form of service, which could then be utilized by other client software.

Regarding wireless technology, in [54] an evaluation of wireless home automation technologies is presented. Furthermore, in [25] a survey of architectures and technologies for wireless home automation is presented.

A large number of home automation systems that were the result of research efforts focuse on a single aspect of the system. Such examples are [24] which focuses on a home automation system utilizing the Zigbee protocol, [28] where a home automation solution based on android is introduced, [36] presents a home automation based on UPnP, and [35] an energy-aware platform that introduces very specific features focused on energy efficiency.

To facilitate the connectivity among devices, the concept of overlay networks has been proposed in [47]. Such an example P2P network based on SOAP can also be found in [53].

Finally, home automation platforms that are currently available and gaining traction in the marketplace are Nest[8], Insteon[6], Philips Hue[9] and IFTTT[5] among others. Some of these platforms such as insteon utilize their own communication protocols where others are using industry standards such as ZigBee[13], Z-Wave[12], KNX[7] and Bluetooth[3]. An older protocol that exists from the late '70s mostly targeted towards remote operation of devices as switches is X10[11]. In closing, a network protocol for the control of devices that is gaining popularity in Japan is ECHONET Lite[4].

3.1.4 Home Service Platform Architecture

A number of home service platforms such as [51],[70] and [34] leverage the power and modularity of the OSGi framework. OSGi provides a number of desirable features such as modularity of software with its use of bundles, discoverability of OSGi services as well as fine control over the lifecycle of software. However, the use of OSGi as the core framework for a middleware platform usually results in a consolidated platform, with an inability to distribute services to other potential processing nodes. Another platform that utilizes R-OSGi that can overcome this problem can be found in [71]. Other platforms such as [55] and [15] utilize REST as their communication protocol of choice, which can lead to systems that are more loosely coupled.

The modeling of devices in terms of functionality is again discussed in

[37], where a functional description of devices is achieved using SOAP. The extensibility of a home service platform is further discussed in [26],

A discussion regarding middleware architecture for ambient intelligence in the home can be found in [23], addressing major considerations such as service discovery and communication interoperability, a model for syntactic and semantic service specification and matching of service capabilities.

In closing, a showcase for an ambient intelligence environment can be seen in [74].

3.2 Platform Architecture and Smart Services

The proposed system is implemented as a middleware platform on top of which smart services can be executed in the form of modules. As stated in the introduction chapter, a smart service is any program that encapsulated specific logic used to complete a target task, utilizing devices that are present in the home environment.

In its current iteration, the proposed system is implemented as a centralized system, designed to be deployed on commodity hardware. The programming language of choice is Java, with the services spawning threads for their data processing needs as necessary.

A conceptual architecture model of the proposed platform can be seen in Fig. 3.1.

In this model, smart services interact with the home service platform through a well-defined resource request API. The requests made through this API are then processed by the appropriate management layer; requests for direct use of a device are handled by the device resource management layer, whereas request regarding environmental properties are handled by the environmental resource management layer. Of particular interest in this design is that the environmental resource management layer relies on the device resource management layer; the device settings necessary to fulfill an environmental resource request are decided by the environmental resource layer but their actual enforcement is achieved through the device resource management layer.

An especially important role in the system is fulfilled by "bridges". These bridges are the necessary software that allows the platform to interact with network-enabled devices of various protocols. A bridge is responsible for mapping logical commands such as "turn on" or"set temperature to 30 °C" into actual network protocol commands, appropriate for each device. A bridge may be responsible for the operation of multiple devices at a time. In the proposed platform, a single bridge is responsible for the operation of all

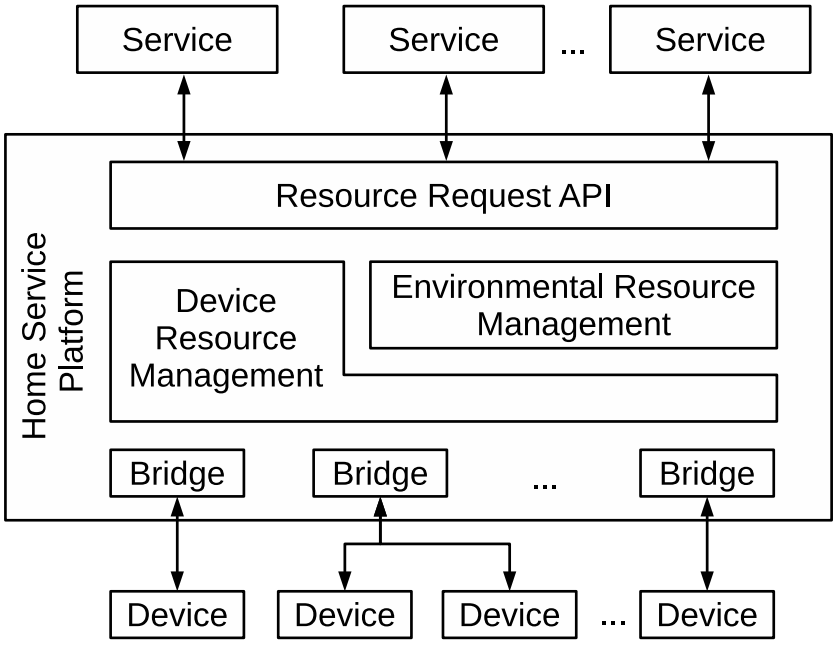

Figure 3.1: Platform Architecture

ECHONET Lite devices, whereas a different bridge is responsible for the communication with the popular Philips HUE illumination lightbulbs. By adding new bridges to the platform, support for new devices and communication protocols can be achieved. Finally, it is the role of the bridge to expose a network-enabled device as a logical device connected to the platform and furthermore map its functionality to already existing device models present in the platform. The device modeling and implementation is explained in the next section.

3.3 Device Modeling

3.3.1 Operations

In the proposed platform, device functionality is expressed in terms of unique *operations*. The `Operation` class and its subclasses are used to encompass device functionality as a logical abstraction. This class has appropriate members and getter-setter methods for passing in the necessary arguments for the operation as well as extracting the result of the operation. For example, the source code for the `SetOperationState` operation can be seen in figure 3.2. Using this operation, a smart service may turn off or on a device.

```
public class SetOperationState extends Operation{

    private OperationState requestedState = OperationState.UNKNOWNERROR;
    OperationState resultingState = OperationState.UNKNOWNERROR;

    public SetOperationState(OperationState newState) {
        this.requestedState = newState;
    }

    public OperationState getOperationState(){
        return this.resultingState;
    }

    public void setOperationState(OperationState newState){
        this.resultingState = newState;
    }

    @Override
    public SetOperationState applyToDevice(DeviceInterface aDevice) {
        aDevice.doOperation(this);
        return this;
    }

    public OperationState getRequestedState() {
        return requestedState;
    }
}
```

Figure 3.2: Operation Example

The passing of appropriate arguments for this operation can be achieved during the creation of an instance of this operation i.e. pass the desired operation state as an argument to the constructor. Then, when the device processes this operation, it can extract the desired state using the `getRequestedState()` method. After this operation has been processed, the resulting state is stored using the `setOperationState()` method and can be later retrieved by the application using the `getOperationState()` method.

For each function provided by the device an appropriate subclass of the `Operation` must be created. However, since these operations are used to describe the functionality of devices, great care must be taken in their design and usage. Each operation subclass must express a unique and well-defined functionality of a device. Device implementers have to select the appropriate set of operations to express the functionality of their devices and be careful not to misinterpret the meaning of an operation.

3.3.2 Device Implementation

A device that is part of the home service platform is associated with several pieces of information such as location, position and others. However,

```
public interface DeviceInterface {
    public void  doOperation(Operation operation);
    Access getNeededPermissions(Operation operation);
    Collection<Class<? extends Operation>> getSupportedOperations();
}
```

Figure 3.3: Device Interface and its main methods.

the main implementation of a device relies heavily on the three methods [1] presented in Fig. 3.3.

These three functions constitute the core of the implementation for a device. A smart service may interact with a device by:

1. performing an operation,
2. getting the required access rights necessary to perform an operation,
3. getting a list of all the currently supported operations for this device.

The generic `doOperation()` method is the main method through which a device performs a given `Operation`. This method may throw exceptions in case the following cases:

1. the device does not support the requested operation,
2. insufficient access rights for the requested operation,
3. network communication failure,
4. operation timed out.

Internally, each device implementation registers operation handlers for the types of operation it supports. If present, the appropriate operation handler will be invoked for the requested operation. If such a handler does not exist, an `UnsupportedDeviceOperation` exception will be thrown. With this modeling approach, to implement a new device it is enough to implement and register the appropriate operation handlers to a subclass of `DeviceImpl`, an abstract class used as the base for all device implementations.

The `getNeededPermissions()` method is a way to inquire about the necessary access rights in order to perform an operation. It is used internally by the platform in order to ensure that the smart service which invoked an operation on a device has sufficient access rights to perform it. Furthermore,

[1] The interface presented here is actually spread into several interfaces and the exceptions that can be thrown have been removed.

```
public class PowerStateAdapter extends DeviceAdapter{

    public PowerStateAdapter(){
        requiredOperations.add(GetOperationState.class);
        requiredOperations.add(SetOperationState.class);
    }

    public void turnOff(){
        this.doOperation(new SetOperationState(OperationState.OFF));
    }

    public void turnOn(){
        this.doOperation(new SetOperationState(OperationState.ON));
    }

    public OperationState getOperationState(){
        return new GetOperationState().applyToDevice(this).getOperationState();
    }
}
```

Figure 3.4: Adapter Interface Example - Power State Adapter

it can also be used by smart services to find out if there is a need to elevate their access rights before invoking that specific operation.

Finally, the `getSupportedOperations()` method is used to inquire a device about the operations it supports. This information can be used as criteria by a smart service to select appropriate devices for use or filter unwanted devices during search.

3.3.3 Adapter Interfaces

Although the API offered for interacting with devices is simple, invoking an operation on a device requires several steps: create an instance of the operation, set the appropriate arguments, perform the actual invocation, check the results and handle any exception that might have occurred. All the above steps are, although necessary, a tedious and sometimes error prone task. Furthermore, due to the fact that a large number of operation subclasses exist, it is easy to invoke an unsupported operation on a device if due diligence is not exercised.

To simplify interaction with devices, a special set of *adapter interface* classes have been introduced. The platform provides a number of typical adapter interfaces. If an appropriate adapter interface is not offered by the platform, a smart service is free to implement and use its own adapter interfaces. A simple example for a service that is interested in controlling the power state of a device can be seen in Fig. 3.4.

This power state adapter interface offers three methods intended to be

used by a service: a `turnOff()` method, a `turnOn()` method and finally a method to get the current operation state, named `getOperationState()`. Before it can be used, a device must be set as the "backing device" for the adapter interface. After this step is complete, the smart service is free to invoke any of the method provided by the adapter. Adapters are reusable and the backing device can be changed at will, at any time.

The major bulk of the implementation of adapter interfaces happens inside the `DeviceAdapter` class. This class is responsible for the error handling. The programmer has a choice regarding the handling of exceptions: either handle them through traditional **`try/catch`** blocks as it is common in Java, or perform "C style" error checking with the help of adapter interface. Even a mixed approach is possible; have only specific exceptions be "silenced" and let important exceptions propagate through the stack until handled.

Finally, another important aspect of an adapter interface is that it maintains a list of the necessary operations that a device must support in order to be usable by the adapter interface. This supported operation list can also be used as a search criteria; a smart service can query the platform for a list of devices that can potentially be used through the specified adapter interface.

Although the use of adapters is, strictly speaking, not necessary, their use is highly encouraged as they simplify smart service development substantially. More details regarding the usage of the adapter interface APIs will be introduced in section 3.5.

3.4 Primitives for Conflict Resolution

In this section the four main primitives provided by the platform for conflict resolution are introduced. The four primitives are: device access rights, service and user priorities, an event notification mechanism and finally condition sets. These primitives play an important part in the development of smart services. The rationale is explained over the following sections.

3.4.1 Device Access Rights

There are three access rights present in the proposed platform:

1. "no access",
2. "read-only" access and
3. "read-write" access.

When a smart service obtains a reference to a device object, the initial access rights for that device are set to "no access". With these access rights, the smart service cannot interact with the actual device directly. However, these access rights are enough to query the home service platform about general properties of the device that are already known to the platform. Such information includes the name, the location as well as the supported operations of the device. No communication to the actual device will be generated as a result of querying the above characteristics.

The next access right present in the platform is the "read-only" access right. This access right can be used to retrieve information about the current state and operation of a device. Designed with information gathering in mind, multiple smart services may hold read-only rights for a given device and enable concurrent access to such data. For example, multiple smart services may be interested in the sensing data of a temperature sensor. In such a case, each smart service uses read-only access rights to access the data of the sensor. Network communication with the device may happen multiple times or, depending on the bridge implementation, data can be temporarily cached and accessed multiple times. In either case, all smart services can access the necessary data, in a concurrent, non-exclusionary fashion. A smart service may hold the read-only rights for a device indefinitely, since in practice it does not affect the operation of other services in any way.

The final access right is a "read-write" access right. As the name implies, on top of the "read-only" access right a smart service can also hold a "write" access right. The "write" access right is associated with actuation. The "read-write" access right is necessary for any operation that results in an action or a change of state of a device.

It is by design that only a single smart service may hold the "read-write" access rights for a given device. The rationale behind this design decision is simple: if more than two smart services could invoke operations that result in an action or a change of state for the same device, conflicts would occur. The "read-write" access right facilitates the exclusionary use of devices by services, which ensures that, on an individual device basis, no operation conflicts can occur.

A smart service must explicitly obtain the read-write access rights of a device through an API call. In contrast to the read-only rights, the service must again voluntarily release the read-write access rights of a device before it becomes usable again from other services. Using this explicit locking mechanism, a service can enforce its exclusive use of a device for its desired time period.

However, the new problem now transforms into which smart service should be the sole owner of the "read-write" access rights of a device. To address

this problem, the service and user priority primitives that are described in the following section are used.

3.4.2 Service and User Priority

Smart services designed for this platform are associated with a numeric service priority. When comparing service priorities, the service with the higher numeric priority is the service with the higher priority overall. In its current iteration, the numeric priority range is $[1, 10]$, with 1 being the lowest possible priority and 10 being the highest possible priority.

Priorities are used mainly to decide which service will be granted the read-write access of a device. If there are multiple smart services interested in these rights, they will be granted to the service with the highest priority. Furthermore, a service with a higher priority will obtain the read-write access rights of a device even if it is currently used by a service of lower priority. The service of lower priority in turn will be notified for the change of its access rights which will be downgraded to read-only access rights through the notification mechanism described in the next section.

Each service is also associated with a user, usually an occupant of the house. This user is either the user who actually initiated and/or scheduled the service, or the user on which the service is focused on and tries to satisfy. User priority can be used as a tie-breaker during the assignment of read-write access rights to services with equal priority. Furthermore, depending on his or her priority, a user may voluntarily lower any service associated with him, or try to raise the priority of one of his service up to his user priority level.

Services come with a suggested priority, a priority level deemed appropriate by the developer of the service. As a general rule of thumb, services that interact with a multitude of devices should hold lower priorities. In contrast, services that have a well defined goal (for example, a home theater service) can be assigned higher priority.

The reasoning for the above rule of thumb is simple. Services that access many different types of devices have a higher risk of interfering with other services compared to services that use only a few devices. Furthermore, services with a clear goal that use only a few select devices either tend to be interactive services that the user is currently paying attention to, or their importance may be high, thus a higher priority level can be justified.

3.4.3 Event Notification Mechanism

An event notification mechanism has been designed and used extensively throughout the proposed platform. Many objects as exposed by the platform

have provisions for a subscriber-publisher model for notification events.

The most prominent event is that of a change in the access rights of a device, currently held by a service. It is through this notification mechanism that a smart service can react to a change in the access rights of a device. A service may at any point have its read-write access rights downgraded to read-only rights due to the sudden appearance of a service with higher priority. In similar fashion, a service of lower priority may request read-write access rights that currently cannot be satisfied. In this case, that service now has the option to wait until read-write access rights can be granted in a future point in time; it will be notified through the event notification mechanism.

Another type of event is user location events. When a user moves inside the house, the new location information is communicated to the smart services through notification events. A service may subscribe to these events and alter its operation. For example, when the user enters a new room a service that subscribes to location notification gets notified of the new position and may choose to use some of the devices present in the room and/or release devices that were used in the previous room.

This event notification mechanism is the core element of the platform that provides the services with a chance to react to context changes. The location of a user is treated as such a change of context of high importance. It is expected that non-intrusive and affordable indoor location systems will materialize within the next few years, thus the proposed platform was designed with such an indoor location subsystem already in mind. The details of an indoor location system as pursued by the author can be found in chapter 4.

3.4.4 Condition Sets

The last primitive that the home service platform provides is the condition set. The condition set captivates a very simple idea: have the smart service perform its main task only when a certain set of conditions hold true.

There are two types of conditions useful to a service:

- context conditions
- device access conditions.

Context conditions include conditions that directly pertain to the physical environment inside the house. Conditions such as date, time, user location and others are considered as context conditions, whereas the devices that are necessary for the correct operation of a service are expressed as device access conditions.

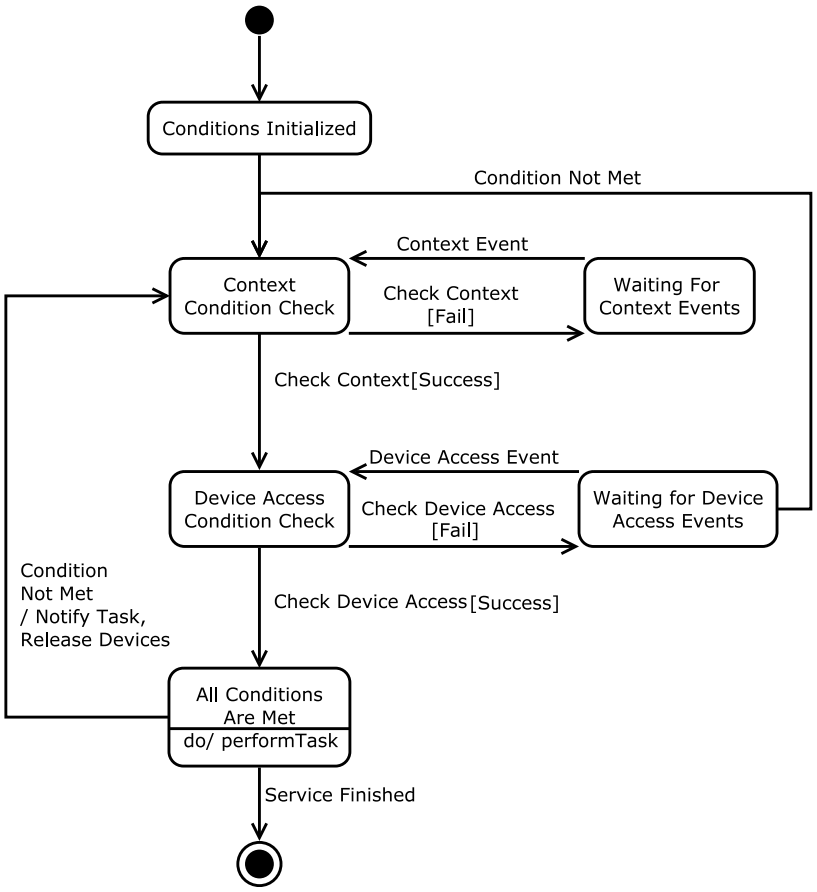

Figure 3.5: Behaviour of Condition Sets

Condition sets have two states: unsatisfied and satisfied. When a transition from unsatisfied to satisfied state occurs, all the necessary device access rights are secured automatically on behalf of the service. In a transition to the opposite direction (satisfied to unsatisfied), the device access rights that the service held are downgraded to "read-only" access rights.

The behaviour of condition sets can be seen in Fig. 3.5. First, a condition set is initialized. Then the context conditions are checked: if the context conditions do not hold, the condition set remains unsatisfied and await further notification events.

Upon the arrival of notification events, the context conditions are evaluated once again. Should all the context conditions hold, the platform proceeds to check the device access conditions.

To evaluate whether the set of device access conditions can be satisfied or not, the platform performs internal checks to determine whether the service with its current priority can acquire all the necessary access rights. Should the system be unable to assign the necessary access rights for even a single device, the condition set remains unsatisfiable and awaits for further device access notification events.

If the platform is able to grant all the necessary device access rights that the service requires, then the condition set finally becomes satisfied, the access rights are secured and the main task of the service may begin its operation.

Should at any point during the task's execution a condition becomes unsatisfied, the device access rights are released automatically and the whole condition set becomes unsatisfied, awaiting again for any number of notification events.

3.5 API and Examples

There are certain operations that are common to all smart services targeting the proposed platform. Such common operations are searching for devices, acquiring the necessary access rights, operating devices, receiving notifications and setting up condition sets if necessary. The details for these common operations are introduced in this chapter, demonstrating the use of the programming API available to smart service developers.

In many of these operations, the use of a `DeviceAdapter` instance is strongly recommended. The design of a device adapter and its relation to other classes present in the system can be seen in Fig. 3.6. The `DeviceAdapter` class is an abstract class that implements base functionality relating to error handling. Its subclasses are used to simplify interactions with devices, as

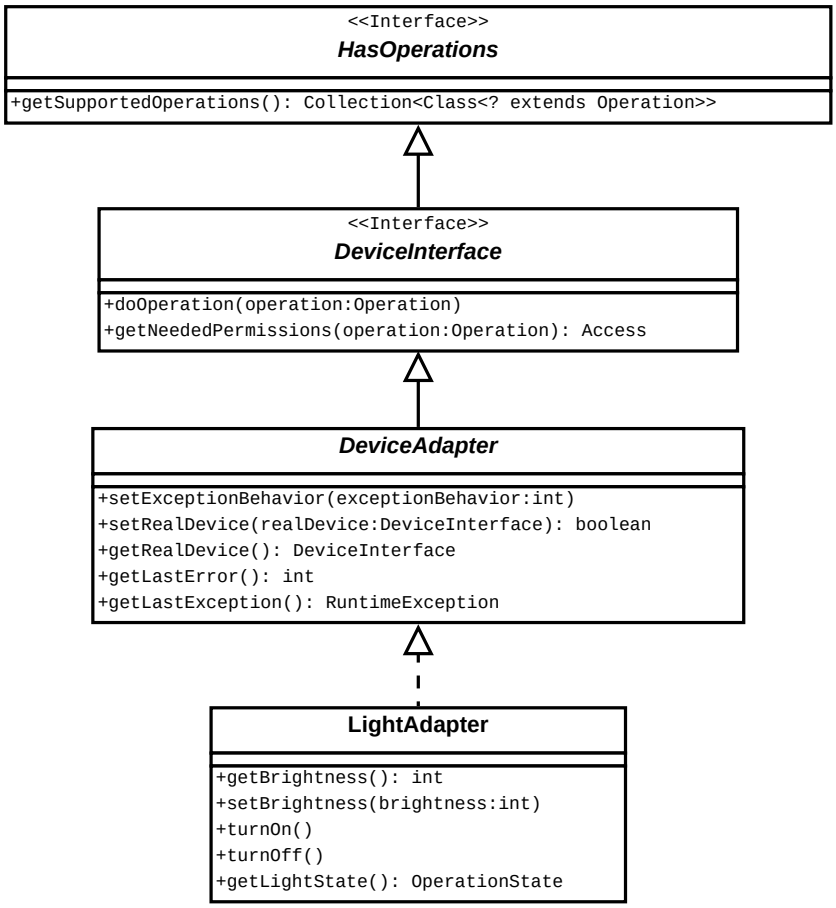

Figure 3.6: Device Adapter Class Diagram

explained in section 3.3.3.

3.5.1 Searching For Devices

The API methods used to search for a device can be seen in Fig. 3.7. The `selectDeviceWithOperations()` method returns a list with all the devices that support a given set of operations as expressed by the `HasOperations` object, which is usually an instance of a `DeviceAdapter` subclass. Using the `SelectDeviceByName()` method a service may look up a device by its name, provided that the service already knows the name of that device.

```
public class HomePlatform{
  /* ... code snip ... */
  public List<DeviceInterface> selectDeviceWithOperations(
    Service aService, HasOperations operations){...};
  public DeviceInterface selectDeviceByName(
    Service aService, String deviceName){...};
}
```

Figure 3.7: API for Searching Devices

3.5.2 Acquiring Access Rights

A service can acquire and release appropriate access rights for a device using the methods seen in Fig. 3.8.

First, using the `requestAccess()` method, a service may request read-only or read-write access rights for a specific device. In the case that read-write access rights are requested, the request will be placed at the appropriate queue for the services that are waiting for read-write access rights for that device. The return result of this method is the currently assigned access rights. The service can then check and see if the appropriate access rights were obtained by checking this result. Furthermore, if read-write access rights were requested but not granted immediately, the service may wait for notification events until the device becomes available again some time in the near future.

The `requestAccessTemporary()` method is used the same way as the `requestAccess()` method, with the sole difference that, if read-write access rights cannot be obtained instantly, this access request is not enqueued, thus the service will not receive any notifications in the future if the device becomes available.

With the `getCurrentAccess()` method, a service may check the device access rights it holds for a specific device.

Finally, using the `canGrantAccess` method a service may make an inquiry to the platform to check if the requested access rights can be granted instantly. Should that be the case, the service may proceed and acquire the access rights for that device using one of the previously mentioned methods.

When a service no longer requires read-write access rights, it has the option to downgrade these access rights to read-only or no-access rights using the `requestAccess()` method again. Common to all the above methods is that the service that makes the request must provide itself and the target device as arguments. Also, in three of these four methods, the desired access rights must be passed as a parameter to the method. Finally, all of the above methods return the access right that would be the result of the invocation of

```
public class HomePlatform{
  /* ... code snip ... */
  public RWAccess.TYPE requestAccess(
    Service aService, DeviceInterface proxy,
    RWAccess.TYPE privType) {...}

  public RWAccess.TYPE requestAccessTemporary(
    Service aService, DeviceInterface proxy,
    RWAccess.TYPE privType) {...}

    public RWAccess.TYPE getCurrentAccess(
      Service aService, DeviceInterface proxy) {...}

  public RWAccess.TYPE canGrantAccess(
    Service aService, DeviceInterface proxy,
    RWAccess.TYPE type) {...}
}
```

Figure 3.8: API for Acquiring Access Rights

that method.

3.5.3 Using Devices

Although not an absolute necessity, it is strongly advised to use any of the preexisting `DeviceAdapter` subclasses to interact with a device. If such an appropriate class is not present or its functionality is not adequate, a service developer is free to introduce and use new `DeviceAdapter` implementations as necessary.

Two preparatory steps must take place before a device can be used through a device adapter:

- instantiate the device adapter and
- set the backing device for the adapter.

The instantiation of a device adapter usually takes place during the discovery of appropriate devices; the device adapter implements the `HasOperations` interface and thus can be used to search for devices that implement the necessary functionality.

After instantiation is complete and the appropriate access rights for a device have been secured, that device must be set as the "backing device" of the adapter using the `setRealDevice()` method. When this step is complete, any of the methods provided by the adapter class may be invoked on the device. Furthermore, it is possible to change the backing device of an adapter at any time.

Figure 3.9: Device Adapter subclasses and their functionality

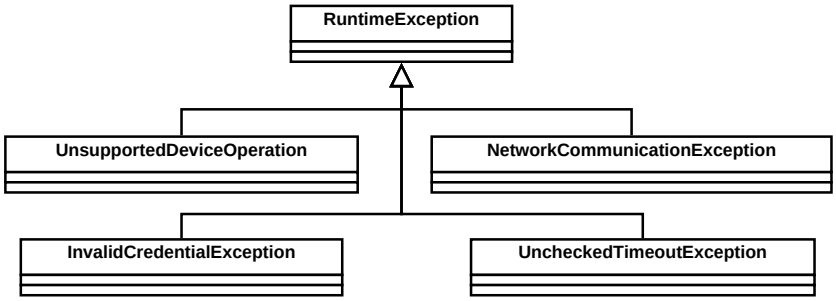

Figure 3.10: Defined Runtime Exceptions

An overview of the available adapter classes currently present in the system can be seen in Fig. 3.9. Each adapter interface has been designed with a specific device in mind and offers method calls with simple and easy to understand names such as `turnOn()`, `setBrightness()` and others.

An alternative method is to invoke operations on devices directly, using the `doOperation()` method with the desired `Operation` subclass instance. In this case, the developer must instantiate an appropriate operation object, set its parameters and pass it to the device. This method may throw four types of runtime exceptions that must be handled by the application to ensure its correct operation.

After an adapter method has been invoked, there are two ways in which error checking can be performed:

- "C style" error checking and
- exception handling.

Using the `setExceptionBehavior()` method, individual types of exception may be silenced. The "C style" error checking can be achieved using the `getLastError()` method to determine the type of error that occurred. If an error occurred, the exception that would have been thrown can be retrieved using the `getLastException()` method, which returns the exception as it was caught by the adapter. This can be used to further investigate the cause of the error.

If "C style" error checking is not preferred, the application must have an appropriate **`catch`** block for the classes of runtime exceptions defined by the platform. Runtime exceptions were preferred over checked exceptions because the alternative would riddle the developer's code with a vast amount of **`try`**/**`catch`** blocks that would reduce the readability of the code. The type of runtime exceptions currently present in the system can be seen in Fig. 3.10.

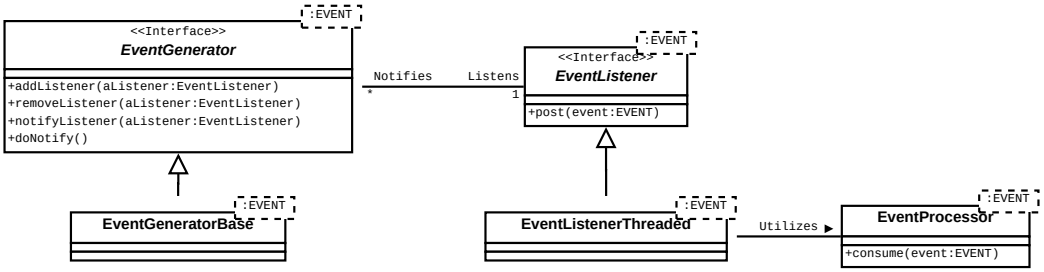

Figure 3.11: Notifications

3.5.4 Receiving Notifications

Notifications are used as the mechanism for sharing information regarding context changes and changes in the access rights of devices. The notification mechanism is based on the `Observer` design pattern as introduced in [22].

Observers in the platform implement the `EventListener` interface whereas the subjects that emit events must implement the `EventGenerator` interface. The `EventGeneratorBase` class implements the basic functionality of an observed subject, able to register, de-register and notify observers. This class can be used directly, usually as a"mix-in" class[2]. Furthermore, for the implementation of an event listener, the `EventListenerThreaded` class generates a thread that waits for the specified type of event. The actual processing of the events is then relegated to an instance of the `EventProcessor` class. The relations of the above mentioned classes can be seen in Fig. 3.11.

All the notification related classes and interfaces are parametrized in terms of the event type that they handle. In Java this templating functionality is achieved through the use of "Generics".

Currently, the two most important events present in the system are user motion events and device access rights changes. For user motion events, a service may directly register a listener with an instance of the `User` class. To listen for device access changes, a service must register appropriate listeners for each device individually. Such a listener registration can be achieved using the following method from the `HousePlatform` class.

```
public void registerForAccessChange(
  Service aService,
  DeviceInterface aProxy,
  EventListener<AccessChangedEvent> listener);
```

The events provide methods to retrieve the relevant information. In the case of a motion event, an instance of the `User` class representing the user

[2]Since Java does not support multiple inheritance directly, this class can be used in a composite object.

who moved is returned as a result. The new position of the user can be retrieved through this instance. In the case of an access rights change, the `AccessChangedEvent` carries information regarding the device for which the access rights were changed as well as the new access rights assigned.

3.5.5 Using Condition Sets

The final primitive that services utilize directly is the condition set. A service initializes a condition set using its constructor. Next, the various necessary conditions are created, initialized and finally registered as part of the condition set. Finally an instance of the `ConditionTask` calls the `waitOnConditions` method of the condition set, ready to be notified when the condition set is satisfied. An example of these steps can be seen below.

```
ConditionSet conditions = new ConditionSet(
        this, this.platform);
long start = ...
long end = ...
conditions.register(new TimeCondition(start, end));
conditions.register(new UserLocationCondition(serviceUser, theaterRoom));
conditions.register(new DeviceAccessCondition(
        this, speaker, RWAccess.TYPE.READWRITE, platform));
conditions.register(new DeviceAccessCondition(
        this, screen, RWAccess.TYPE.READWRITE, platform));
platform.acceptConditionSet(conditions);
conditions.waitOnConditions(this);
```

This code is part of a sample home theater service that utilizes condition sets. The search for devices step has already been concluded and the most appropriate devices (a speaker and a screen) have already been discovered.

In the above example, two context conditions are initialized and registered. The first condition is a time condition with a start and end time. The second condition is a user location condition, which requires that the service of the user is in the room designated as the theater room.

Moving on, two device access conditions are also registered; the first condition requires that read-write access rights are necessary for the device designated as "speaker" and the second condition requires read-write access rights for the device designated as a "screen".

As the last two steps, the condition set is registered with the main home platform instance, and the service calls the `waitOnConditions` method itself, as it implements the `ConditionTask` interface.

The condition set will call the `performTask()` method on the condition task when the set is satisfied. Also, whenever the condition set becomes unsatisfied it will call the corresponding `suspendTask()` method of that task.

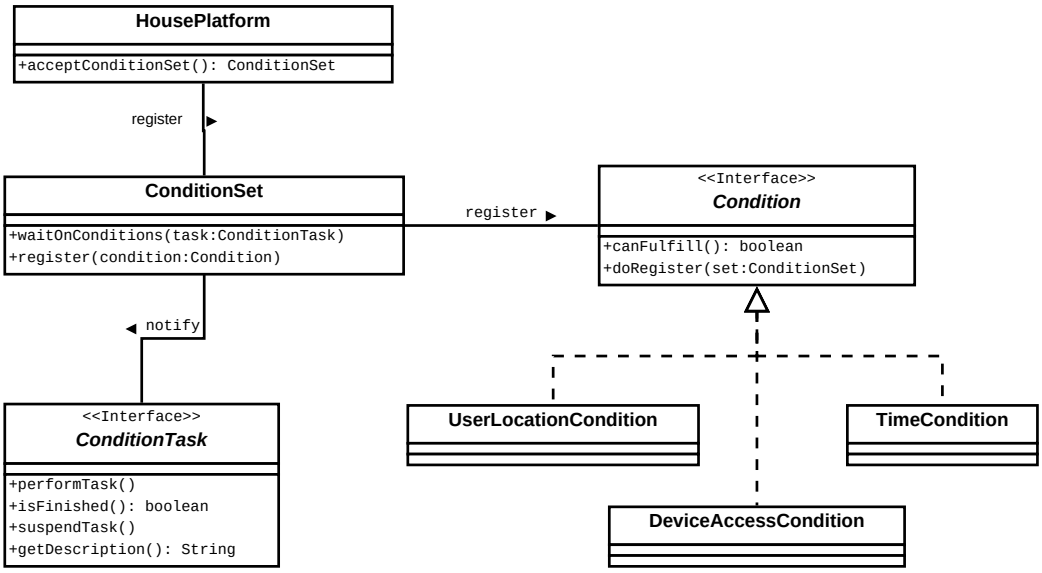

Figure 3.12: Condition Set Related Class Diagram

Error handling in the `ConditionTask` also becomes simplified; as soon as the necessary access rights are lost, if the task tries to invoke a command that requires these access rights, the exception will be caught and the task will be suspended again. The same is also true for other exceptions, if there is no exception handler present.

An overall class diagram with all the classes that are related to the condition set can be seen in Fig. 3.12.

3.6 System Demonstration

Two scenarios that demonstrate the effectiveness of the primitives present in the system will be explained in the following sections. In these two scenarios, a total of four sample smart services were developed. The details of these four services are explained in the next section.

3.6.1 Sample Smart Services

Four sample services were developed for the purposes of demonstration:

1. Energy Saving Service,
2. Smart Listening Service,
3. Home Theater Service,

4. User Scheduled Task.

For each service, the user of the service and his user priority as well as the service priority were defined. The details of the four services follow.

Energy Saving Service The energy saving service is a simple service with only a single task: turn off any device which is currently not in use. This service can potentially turn off any device in the house and has the potential to interfere with the normal operation of other services. Therefore, its priority is set to the lowest level possible.

This service registers notification handlers for all the devices present in the house. As soon as the service obtains read-write access rights for a device, that device is turned off. Due to the low priority of this service, any other service with priority higher than the energy saving service is able to obtain the read-write access rights if it so wishes to.

The service priority is set to "1" and the user for this service is the fictional user "Dad".

Smart Listening Service The smart listening service is a service that plays back an audio stream such as a podcast or a song playlist using the available speakers in the home. This service chooses the most appropriate set of speakers depending on the user's location; should the user of this service move to a different room, the smart listening service will try to use another pair of speakers located in the new room (if available) and stop using the speakers located in the previous room. Should no speakers be available in the new room, the service continues to use the speakers it already uses.

A user is associated with each instance of this service. For this target user, the service receives user location notification events and adjusts its operation as described above. For demonstration purposes, the priority for this service is set to a mid level priority of "5", with the user of this service set to the fictional user "Mom".

Home Theater Service The home theater service is responsible for playing back movies in the house. The necessary devices for this service are a screen device and a set of speakers. The movies themselves are thought to be streams, either from some local or cloud storage.

The priority for this service is set to a fairly high priority setting of "7", with the rationale behind this decision being simple; watching a movie is an interactive process that the user usually dedicates his full attention to it. Furthermore, this service uses a very specific and limited set of devices, limiting the chances of conflicts with other services. By assigning a high

priority to this process we ensure that the user of this service will not be interrupted during the playback of the movie.

This service utilizes a condition set. One of the conditions in this set is that the user is located inside the room. Should the user move outside the move at any point the service will suspend itself, awaiting the return of the user back to the room.

For demonstration purposes, the user of this service is set to the fictional character "Kid2".

User Scheduled Task The final service developed is a simple user scheduled task that plays back a short reminder audio message. Since this service acts as a simple reminder, its priority is set to the highest level of "9". The reminder message is played back using a speaker device. The intended recipient of the message is the user "Kid2", and the user who set up the service is the user "Mom".

The rationale for assigning the highest priority level for this task is as follows:

1. the reminder notification is of high importance, so it should be unobstructed by other services that happen to be using the available speakers in the house,

2. the reminder message is short, so any interruption that may occur to other services will also be short. Therefore, even by assigning a high priority, this priority is not miss-used and interruption to other services is still kept to a minimum.

3.6.2 Simple Device Conflict

In the first scenario a simple device conflict is demonstrated. The conflict occurs over the use of speakers, between the smart listening service and the energy saving service. In this scenario, the energy saving service is holding the read-write access rights for device "speaker4" and the smart listening service is holding the read-write access rights for device "speaker3".

The event that triggers the conflict is the user's movement to a new room in the house. The interaction unfolds as follows:

1. user "Mom" moves to a new room,

2. the middleware platform generates a user motion notification event, that is received by the smart listening service,

3. the smart listening service requests a list with all the available speakers in the home, and identifies "speaker4" as the most appropriate speaker to use inside the new room,

4. the smart listening service requests read-write access rights for "speaker4",

5. the access rights for device "speaker4" of the energy saving service are downgraded to read-only rights,

6. the smart listening service is granted the read-write access rights for "speaker4",

7. the smart listening service starts playback on "speaker4" and relinquishes the read-write access rights of the previously used speaker, "speaker3",

8. the middleware platform notifies the energy saving service of an upgrade in its access rights for device "speaker3",

9. finally, the energy saving service receives the notification and turns off the device "speaker3".

A timeline showing the events explained above can be seen in Fig. 3.13. The raw output produced by the services can be seen in Fig. 3.14. The output is slightly reordered due to scheduling of threads, but this is only a cosmetic issue, and operation steps described above can be clearly traced.

In this demonstration, the two services end up switching read-write access rights for two devices ("speaker3" and "speaker4"). Both services are able to continue their operation uninterrupted, also adjusting their operation to context changes. This is a common scenario that is expected to frequently occur during the operation of this middleware platform. A service may lose read-write access rights to a device and may adapt to this change or, in a worst case scenario where no appropriate substitute device is found, voluntarily suspend its operation.

3.6.3 Condition Set Demonstration

For the second demonstration of the platform, a scenario which involves the home theater service, the user scheduled task and the energy saving service was set up. In this scenario the home theater service and the user scheduled task are using condition sets and their effectiveness is demonstrated.

The scenario unfolds as follows: the user "Kid2" has activated the home theater service at the living room. During the playback of the movie, the user

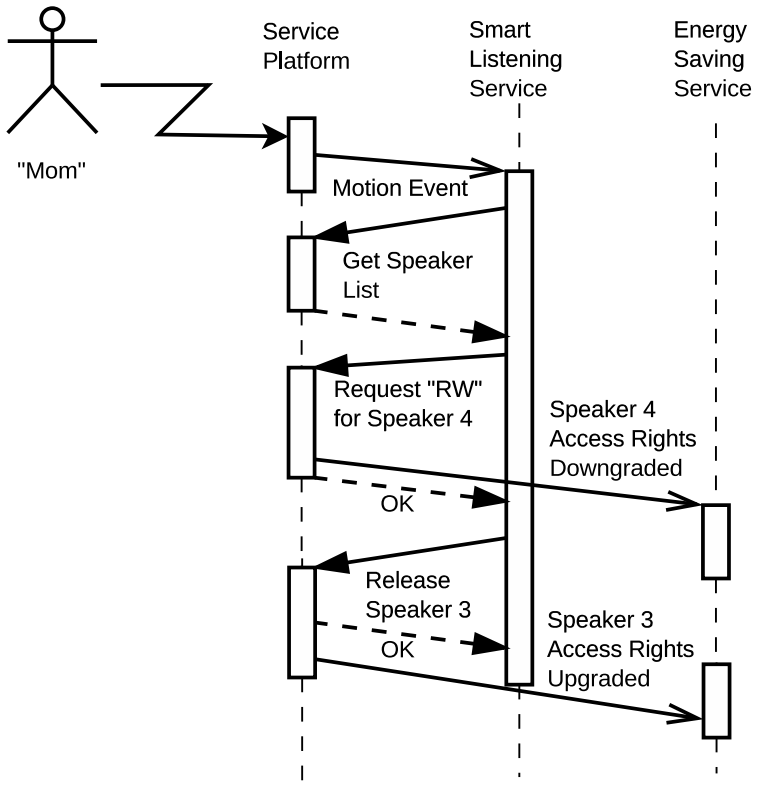

Figure 3.13: Timeline

```
Smart Listening Service: User moved into room: Sleeping1
Smart Listening Service: Releasing previous speaker with name: Speaker3
Smart Listening Service: New speaker device: Speaker4
Smart Listening Service: Turning speaker on..
Smart Listening Service: speaker with name Speaker4 is playing back stream with description: Random Podcast
EnergySavingService: access changed event
EnergySavingService: Device: Speaker4, new access rights: READONLY
EnergySavingService: access changed event
EnergySavingService: Device: Speaker3, new access rights: READWRITE
EnergySavingService: turned off device: Speaker3
```

Figure 3.14: Raw Output from the Simple Device Conflcit Demonstration

scheduled task activates, playing a reminder audio message intended for user "Kid2". After the playback of the reminder message is over, the playback of the movie continues. The interactions among the services occur at two spots: when the user scheduled task activates and when it ends.

First Half The first half of the interaction unfolds as follows:

1. the time condition of the user scheduled task is fulfilled, which triggers a test for the satisfiability of the condition set of this task.

2. The device access condition set by the user scheduled task can be fulfilled, due to the task's higher priority. Its condition set becomes fulfilled.

3. The platform downgrades the read-write access rights of the home theater service for the speaker to read-only rights and grants read-write access to the user scheduled task.

4. Having lost read-write access rights to the speaker, the condition set of the home theater service becomes unsatisfied and the service temporarily suspends its operation.

5. The read-write access rights of the screen that was used by the home theater service are automatically downgraded to read-only rights.

6. The user scheduled task plays back the reminder audio message, using the speaker.

7. Finally, the energy saving service can now use the screen device. Its access rights for the screen are upgraded to read-write access rights and the screen is turned off.

The raw output of the system for the above steps can be seen in Fig. 3.15. Again, the output of the various services is interleaved due to thread scheduling, without any effect on the handling of the resources.

In this first half of the interaction, the home theater service and the user scheduled task conflict over the use of the device "speaker1". Also, the energy saving service and the home theater service interact over the use of the device "TV1". Since the home theater service suspends its operation, the energy saving service has a small window to use (and turn off) the TV.

```
Condition set for service: Home Theater Service with Conditions became unsatisfiable.
HTS suspending self!
EnergySavingService: access changed event
EnergySavingService: Device: TV1, new access rights: READWRITE
Condition Set for service Scheduled Actions with Conditions became Active!
EnergySavingService: turned off device: TV1
HTS was interrupted!
Service: Scheduled Actions with Conditions, wakeup! perform task...
Service: Home Theater Service with Conditions, waiting on condition..
Speaker Task: playing back stream!
Speaker Task: Stream Description: kids don't stay up late!
```

Figure 3.15: Raw Output from the Condition Set Demonstration (First Half)

```
Speaker Task: task finished!
Speaker Task: release the speaker!
ConditionSet: task with description: Speaker Task: reminding the kids to not stay up late finished!
Task: Speaker Task: reminding the kids to not stay up late finished!
Condition Set for service Home Theater Service with Conditions became Active!
Condition set for service: Scheduled Actions with Conditions became unsatisfiable.
EnergySavingService: access changed event
EnergySavingService: Device: TV1, new access rights: READONLY
Service: Home Theater Service with Conditions, wakeup! perform task...
HTS: turning Screen ON
HTS: playing video stream: Test video stream
HTS: playing audio stream: Test audio stream
```

Figure 3.16: Raw Output from the Condition Set Demonstration (Second Half)

Second Half The second half of the interaction occurs when the user scheduled task finishes the play back of the reminder message. The scenario proceeds as follows:

1. the user scheduled task finishes execution and releases the speaker.
2. With the release of the speaker, the condition set for the home theater service is checked for satisfiability:
   (a) the context conditions are satisfied (user location, time),
   (b) the device access conditions can be satisfied (TV is held by a service of lower priority, the speaker was just released and triggered the check).
3. The condition set is satisfiable and the access rights for the TV and the speaker are upgraded to read-write access rights, while the energy saving service has its TV access rights downgraded to read-only.
4. The home theater service resumes its operation (continue the playback of the movie).

The raw output of the above steps can be seen in Fig. 3.16. In this second half of the interaction, the home service continues its operation after the user scheduled task is over. There was no extra application logic that had to be implemented due to the use of condition sets, overall simplifying service development.

3.7 Conclusions and Future Work

In this chapter the topic of managing devices as resources of the home environment is discussed. To represent the devices in the home, the proposed system models devices in terms of their functionality. With this approach, common tasks such as searching for a device, acquiring appropriate access rights and operating a device are simplified, as the network communication protocol details are abstracted and hidden from the smart service developer, leading to easier and faster development of smart services.

To tackle the problem of runtime conflicts among services over device resources, a set of design primitives for conflict resolution are introduced. Drawing inspiration from the field of operating systems, device access rights, service and user priorities, an event notification mechanism as well as condition sets were designed and implemented as core features of the proposed platform. These primitives have well-defined semantics and are easy to use. With the use of these primitives, a service is now able to react in the face of context changes; it can adapt by searching for alternative devices (should the access rights for a device be lost), it can react to user location events and in a worst case scenario willfully suspend its operation until its prerequisite execution conditions are fulfilled some time in the near future. Regardless of which option a service is going to pursue, its operation now becomes *predictable* leading to more resilient services in the home.

In the demonstration section of this chapter, the effectiveness of the above proposed primitives were demonstrated in some typical device resource conflict scenarios. The services were able to continue their operation successfully, recovering gracefully to device access rights downgrade events and also reacting to user location changes whilst exhibiting predictable and dependable behaviour throughout their execution.

Overall, with the modeling of devices in terms of their functionality and the introduction of primitives for conflict resolution, the difficult task of developing sophisticated but resilient smart services for the home environment now becomes easier and more manageable. These characteristics of the proposed platform are very compelling and should be part of every future home service platform that strives for wide-spread adoption by consumers.

As possible future works, an offline rule based system for access control similar to what HomeOS currently provides is a desirable feature. Another meaningful extension of this work would be the introduction of contextual service priority modifiers. For example, certain services may be able to use elevated priority in different times of the day as well in different rooms. In the latter case, services associated with a user would have elevated priority inside that user's sleeping quarters, making them more resilient to disruption from other services. Finally another area of interest would be to provide the functionality of the proposed platform as a service oriented architecture, a step that would allow the creation of smart services in any programming language.

Another topic that should be addressed as a future extension of the current system is safety considerations. Smart services deployed on the proposed platform have the ability to alter and control the physical environment around the user. However, a malicious service has the potential to cause discomfort or even attempt to physically harm a user. To avoid such possible scenarios, a safety module that can reason whether the operation of a device may be harmful or not should be present. Such a safety module can operate based on explicit conditions formed as rules; should the invocation of a device operation lead to a broken rule, the invocation is suspended.

Regarding the technical aspects of the system, a standardized smart service module description must be pursued. This module description format must capture information regarding the possible requirements that a service has. Using this information, a service may be checked for compatibility issues with a given instance of the proposed platform; should the necessary devices for this service not be present in the house it would make little sense to install that service. A similar approach from which inspiration can be drawn is the Android operating system[1] for smartphones, where the requirements of an application are clearly stated in its manifest file. Furthermore, should a smart service be provided by a third party provider, further information regarding the service level agreement must be present and be readily accessible by the user.

Another improvement on the system may be the addition of non-preemptive device operations. There may be many meaningful cases in which, after the operation of a specific device has started, a preemption of the given operation (by a change in the read-write access rights of the device) may lead to undesirable results. A solution for this would be to mark certain device operations as non-preemptible and forbid any device access rights changes for this device until the non-preemptible operation has finished.

Chapter 4

A User Indoor Location System

4.1 Introduction

A critical component of a home service platform is a user indoor location subsystem. In order to provide sophisticated services to the users, user location information is one piece of context information that is absolutely critical.

In contrast to outdoor location information where the Global Positioning System (GPS) is a defacto standard and upcoming technologies such as Galileo planned for the near future, there is no de facto standard for indoor location systems. Although it may seem initially surprising, the reason for the non-existence of such a de-facto standard becomes readily apparent when the requirements of such an indoor location system are considered. Simply put, the reason behind the existence of multiple indoor location systems lies in the fact that each such system prioritizes different operational aspects of the system, aspects mainly such as location accuracy, ease of use, ease of deployment, privacy and cost considerations.

In the next section a brief introduction into related indoor location systems is presented and in section 4.3 the design and rationale behind the proposed indoor location system is presented, along with a comparison to other systems. The rest of this chapter also deals with implementation details, an experiments section and the final concluding remarks.

The work presented in this chapter was published in [59].

4.2 Related Research

Compared to outdoor location systems where GPS is the de facto standard, there is no such standard for user indoor location systems. In the past many solutions have been proposed, based on technologies such as active RFID[48],

cameras [42](survey), ultrasound[30], passive infrared sensors (most notably ThiLo[31]) as well as a number of solutions based on the Received Signal Strength Indication of different wireless systems (Wi-Fi[72], Bluetooth[21], ZigBee[63]). Furthermore, a work that deals with the detection of moving devices is presented in [61]. A system targeting bigger indoor spaces and based on radio-frequency and base stations is presented in [14].

It is the authors conviction that such indoor location systems will be an indispensable part of a platform on which other smart services can be developed and assist the user in his everyday activities.

4.3 Design Approach and Rationale

The design of the proposed indoor location system is based on the use of passive infrared motion detection sensors. This decision was made on the basis of making the location system easy to use; with passive infrared sensors, a user does not have to use or carry any extra equipment, such as an RFID tag, active badge, a mobile phone or anything else. The presence and motion of the user inside the space where the indoor location system is deployed is enough to obtain data regarding the location of the user.

Having decided on the use of passive infrared sensors as the core for our system, the next challenge was to obtain location information with reasonable accuracy. For the scope of this project, an accuracy of 50cm or less is deemed ideal. As each passive infrared sensor only provides binary information, i.e. the user is present in the sensor's detection area or not, and the sensor's conical detection area can span quite a few cubic meters, a challenge arises: find a method to obtain higher resolution location data from these passive sensors with coarse detection features.

The answer to this challenge lies in overlapping detection areas and space subdivision. A simple example can be seen in Fig 4.1. Intuitively, in this figure, if both the conical detection sensor and the spherical detection sensor are "active" (i.e. both sensors report that a user is present in their detection areas) it logically follows that the user must be located at their overlapping detection area, colored red in this figure. For different combination of sensor states, similar deductions can be made. For example, should the spherical detection sensor report no presence and the conical detection sensor reporting user presence, it is logical to assume that the user is located somewhere inside the conical area, excluding though the overlapping part with the spherical sensor.

Taking this idea to its logical extreme, by increasing the number of sensors used in our system an adequate number of areas that are covered by a unique

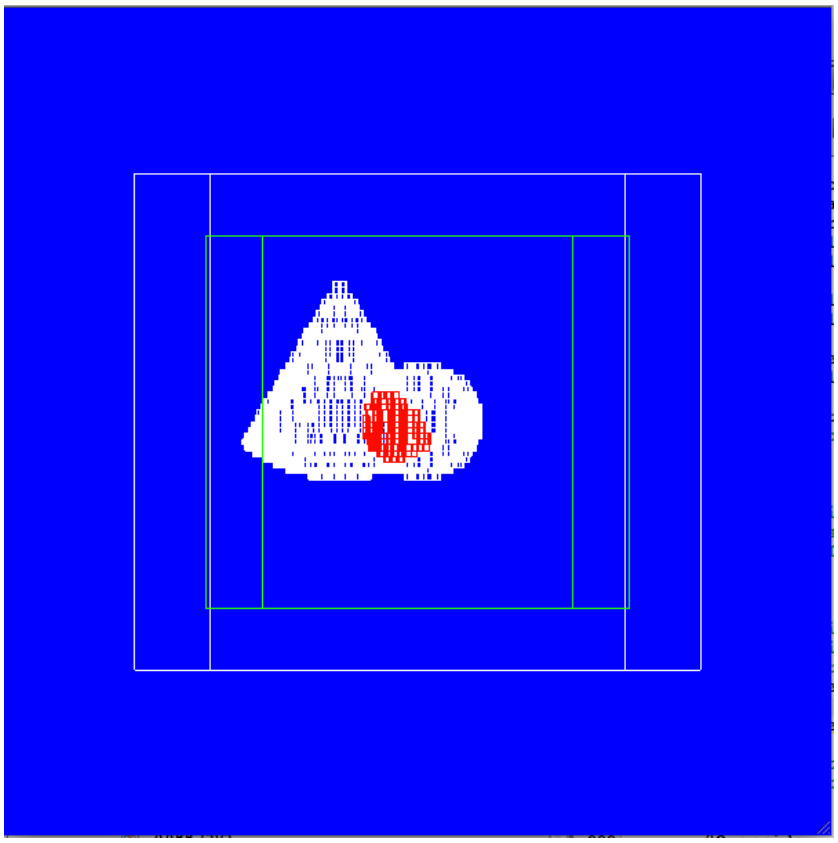

Figure 4.1: Overlapping Area Example

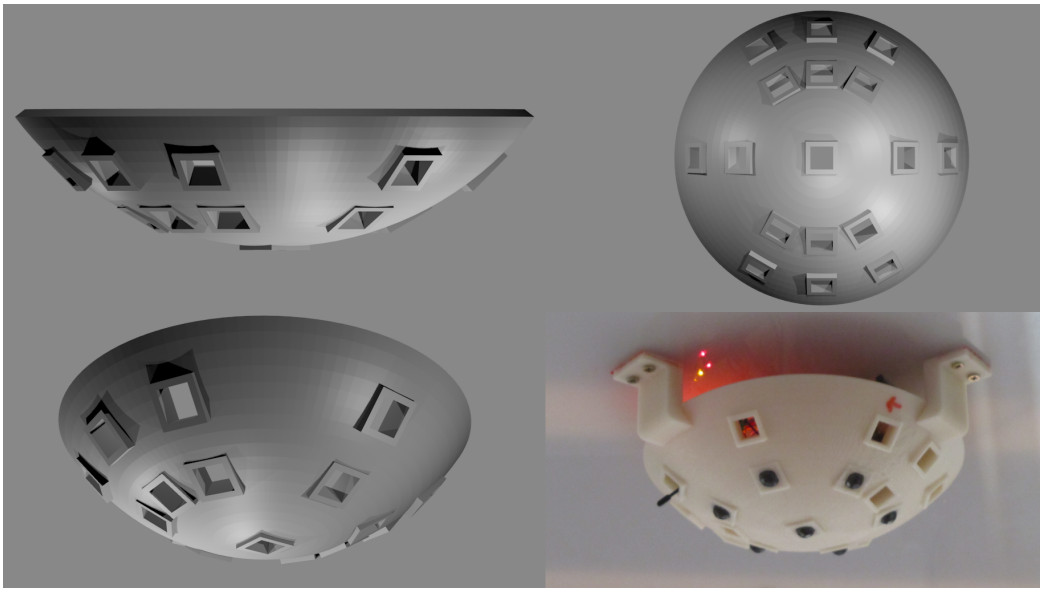

Figure 4.2: Sensor Pod Design and Actual Pod

set of sensors may be generated. To maximize the number of overlapping areas among the sensors, the sensor pod seen in Fig. 4.2 was designed and created with the use of 3D printing technology.

Certain characteristics for the design of this pod must be highlighted. First, the pod is designed to be mounted on the ceiling of a room. Second, a set of 17 sensor inlets have been designed, allowing the pod to be customizable to the needs of a room; should for whatever reason the pod be mounted close to the corner of a room, certain inlets lose their value due to their wall-facing positions. In our experiments, all sensor pods where equipped with a total of seven sensors, laid out as seen in the lower right part of Fig. 4.2. The inner inlet circle is designed with a 30° angle with respect to the vertical axis, whereas the outer inlet circle is designed with a 45° angle respectively. A different combination of inlets may be used to cover an area according to the needs present.

The subdivision of space achieved with a single pod that utilizes the outer 45° inlet ring can be seen in Fig. 4.3. In this figure, areas depicted as grey, yellow, red and finally, black are covered by 1, 2, 3 and 4 sensors respectively. Using the inner inlet ring, even more unique areas are generated, covered by as many as 6 sensors at a time.

To effectively cover the area of a medium sized room, 3 or 4 sensor pods should be used to provide adequate coverage. For smaller rooms of less than 10m^2 in size, two pods with customized sensor placement should be sufficient.

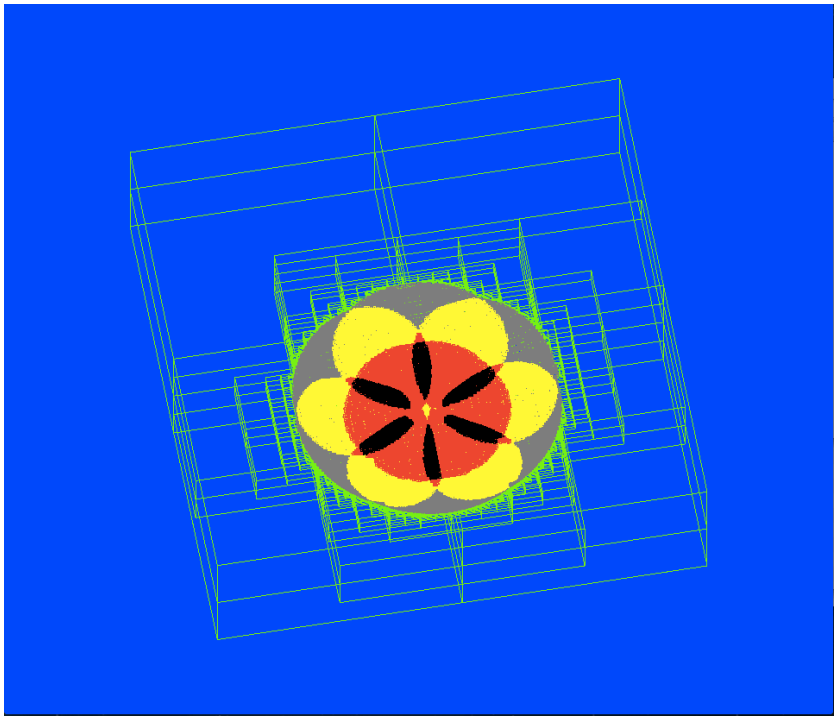

Figure 4.3: Coverage Area of a Sensor Pod, Using the Outer Inlet Ring

The coverage achieved by three pods in a room used for the experiments of this chapter can be seen in Fig. 4.4. With colours starting from light gray all the way to green, black and white, areas are being covered by a unique combination of 5 sensors all the way to 11, 12 and even 13 sensors at a time. Furthermore the size of these unique areas is quite small, appropriate enough to differentiate among different positions with accuracy of 15cm among neighbouring areas.

4.3.1 Subdivision of Space

The unique areas generated by the subdivision of space are expressed as a set of nodes in an octree. The root node of the octree as used for experimental purposes is configured as a 5x5x5meter cube and has 6 levels. The nodes in the deepest level are thus 15x15x15cm in size.

The conical coverage area of a sensor can be calculated by performing a set of intersection tests for that area and the nodes of the octree. The result of this operation is a set of nodes (not necessary leaf level nodes) that express the coverage area of the sensor. Such an example can be seen in Fig. 4.5. It is quite clear (especially in the bottom right corner) that nodes of lower

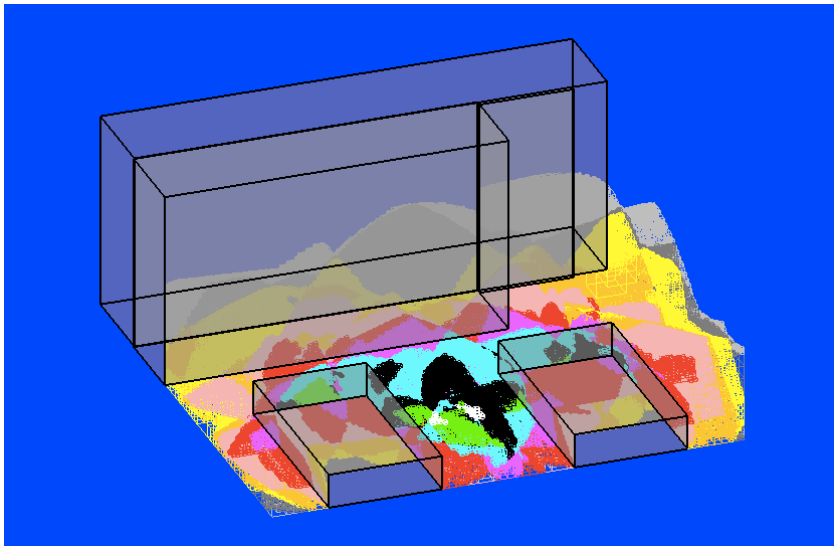

Figure 4.4: Room Coverage of the Experiment Room

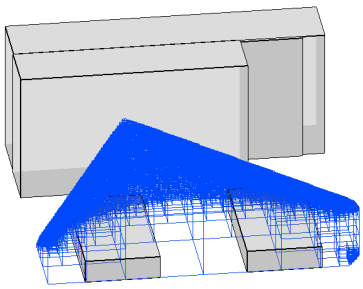

Figure 4.5: Sensor Coverage Area

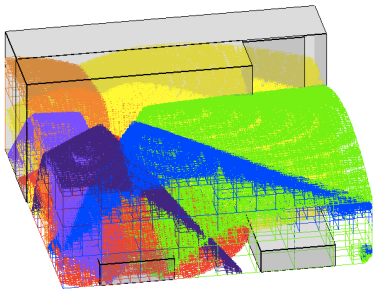

Figure 4.6: Pod Coverage Area

depth are used to express areas of higher volume wherever possible. Leaf nodes (nodes with the highest depth) are used near the edges of the conical coverage area.

The algorithm for computing the unique areas can be seen in Alg. 3. The function *ColorNodes* is the first step of the algorithm. Here, each conical sensor area is calculated in three parts: expand, extend and suppress, the details of which will be presented later. In this first step, each node is assigned a "color" with the color representing the unique id of the sensor it is covered by.

In the next step, represented by the *SpreadPaint* function, the colors of the nodes are spread from parent nodes to child nodes and consolidation of child octree nodes to parent nodes is performed wherever possible. This helps reduce the amount of used nodes in the octree and improve performance.

In the final step of the algorithm, the nodes of the octree are traversed in an in-order fashion, adding the nodes to the appropriate list of a hashmap, with the set of colors a node is painted by acting as a key. In simple terms, all nodes that are painted by exactly the same colors are aggregated to lists. This hashmap holds all the unique areas (list of nodes) as values in the hashmap, and their unique color combinations as key values.

The first step of the algorithm is executed in three phases per conical sensor area. Their visualization on a quadtree can be seen in Fig. 4.7. These three phases are:

1. expand,
2. extend,
3. suppress.

In the expand phase, starting from an appropriate position inside the coverage area (usually close to the center) a starting node of the deepest level is selected. This node should be completely covered by the conical area of the sensor. Then, its parent nodes are checked in succession until a node of lower depth is found that is partially covered by the sensor's detection area. This node is depicted as red in the left image in Fig. 4.7. As soon as this bootstrap node is found, the second phase begins.

In the extend phase, the neighbouring octree nodes of the bootstrap node are checked for collisions with the sensor's detection area. For neighbouring nodes that overlap the check continues recursively; their neighbouring nodes are also check and so on, until no other such nodes that overlap with the sensor area can be found. As soon as the set of overlapping nodes (depicted in red in the center image of Fig. 4.7 is computed, the third phase begins.

In the third and final phase (suppress), each node found in the second set is checked against the coverage area of the sensor. Should the node be wholly included in the area, the node is added to the result set. The check continues recursively for the child nodes that are also partially covered. Finally if a leaf node is also covered, even partially this time, it will be added in the result set. This result set is depicted in yellow in the right picture of Fig. 4.7.

The area calculation algorithm performs thousands of intersection tests. It is for this reason that it should be computed ahead of time. Using our equipment (a 2013 MacBook Air) the algorithm finishes in thirty to forty seconds.

Algorithm 3 Area Computation Algorithm

```
1: function GenerateAreas
2:     ColorNodes
3:     SpreadPaint(rootNode)
4:     CalculateAreas(rootNode)
5: end function
6:
7: function ColorNodes
8:     for each DetectionCone in DetectionCones do
9:         bootstrapNode ← expand()
10:        nodeSet ← extend(bootstrapnode, DetectionCone)
11:        result ← suppress(NodeSet)
12:        for each node in result do
13:            node.addColor(DetectionCone.color)
14:        end for
15:    end for
16: end function
17:
18: function SpreadPaint(Node root)
19:    for each node in root.traversePreOrder() do
20:        color children with node's color
21:    end for
22:    for each node in root.traversePostOrder() do
23:        if all child nodes have the same colors then
24:            color the parent node with their colors
25:            remove children nodes
26:        end if
27:    end for
28: end function
29:
30: function CalculateAreas(Node root)
31:    hashMap(colorSet, nodes) ← ∅
32:    for each node in root.traverseInOrder() do
33:        nodeSet ← hashMap.get(node.getColors)
34:        if nodeSet == ∅ then
35:            nodeSet ← init()
36:            hashMap.put(node.getColors, nodeSet)
37:        end if
38:        nodeSet.add(node)
39:    end for
40: end function
```

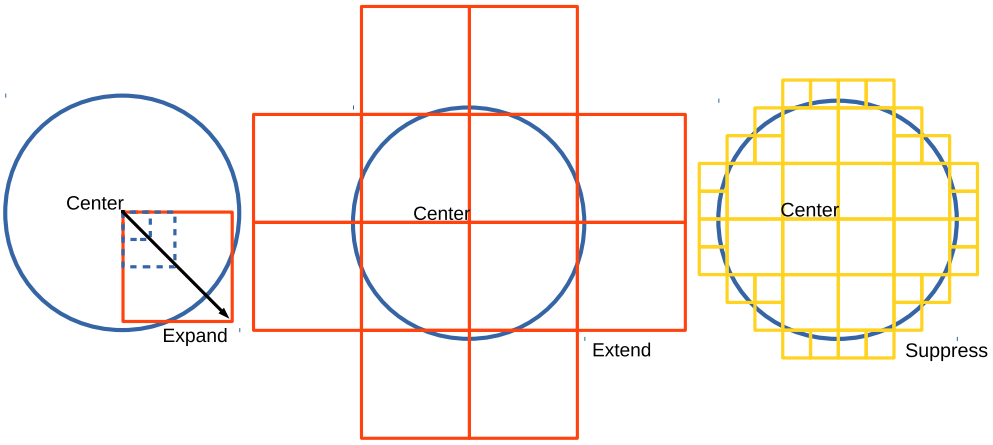

Figure 4.7: Expand, Extend, Suppress on a Quadtree.

4.3.2 Sensor Behaviour

The next topic regarding the proposed indoor location system that must be addressed is the behaviour of the passive infrared motion sensors.

A passive infrared sensor provides no information regarding the distance of the subject from the position of the sensor. These sensors operate in binary mode, offering information about the presence or absence of a subject in their detection area. These two states of the passive infrared sensor will be referred to as "active" and "inactive" or "high" and "low" output correspondingly throughout this chapter.

Before the actual behaviour of the employed sensors is explained, the behaviour of an "ideal" motion sensor is presented. Such an ideal motion sensor has the following properties:

- switches to a "high" state as soon as a user enters its detection area,
- retains the "high" state for as long as the user resides in its detection area,
- switches to "low" output state as soon as the user exits its detection area.

Experiments conducted with such simulated ideal motion sensors provided great accuracy results that verified the overall space subdivision approach proposed for this system.

However, in contrast to this ideal motion sensor behaviour, the AMN31111 passive infrared motion sensor from Panasonic exhibited the following behaviour:

- tardiness to switch to “high” output when the user entered the outer region of the detection area,
- tardiness to switch to “low” output when the user exited the detection area of the sensor,
- the output state would switch to “low” after a few seconds if the user was stationary but still located inside the detection area.

This difference in behaviour between the ideal motion sensor and the actual sensor used in the proposed system produced significant side effects that had to be taken into consideration during the development of this system. The effects of this behaviour as well as how it was reflected in the results obtained in the experimental section will be discussed in depth in Sec. 4.5.3.

4.3.3 Area Evaluation Algorithm

To find the most probable area in which a user is located at, a simple area evaluation algorithm was devised. This algorithm depends on the state of the sensors at a specific time. Each sensor is assigned a score depending on its state. For an ideal sensor, a state of “high” output would be associated with a positive value whereas a “low” output state would be associated with a negative value. For the deployed sensor, more nuanced states where used and their scores will be introduced later. The assigning of scores to a sensor state is the sensor-depended part of the area evaluation algorithm.

To assign an overall score that evaluates an area, the algorithm seen in Alg. 4 is used. Simply put, if a sensor covers an area, its score is added to the current sum that acts as the score of the area. Conversely, if a sensor *does not* cover an area, its score will be subtracted from the current sum. The area with the highest sum is the area that the user is most likely to be located at.

The reasoning behind subtracting the score of a sensor from an area it does not cover is simple. Should a sensor in active state not cover an area, this area is less likely to be the one that the user is located at and thus has to be penalized. Furthermore, if an inactive sensor does not cover an area, that area has a higher chance to be near the actual vicinity of the user, thus it should be rewarded.

Returning to the sensor dependent part of the algorithm, the states and the scores used to express the behaviour of the AMN31111 sensor can be seen in Tab. 4.3.3.

To deduce the state of the sensor, only the data collected in the last one second are taken into consideration. Any data acquired previously is

Algorithm 4 Area Evaluation Algorithm

```
1: function EvaluateAreas
2:     for each Area in Areas do
3:         EvaluateArea(Area, Sensors)
4:     end for
5: end function
6:
7: function EvaluateArea(Area, Sensors)
8:     Sum ← 0
9:     for each Sensor in Sensors do
10:        if Area.coloredBy(Sensor) == true then
11:            Sum+ = Sensor.getScore()
12:        else
13:            Sum− = Sensor.getScore()
14:        end if
15:    end for
16:    Area.setScore(Sum)
17: end function
```

Table 4.1: Sensor States and Associated Scores for the AMN31111 Sensor

State	Rising	Uptrigger	Up	Flux	Falling	Downtrigger	Down
Score	30	100	50	5	-30	-100	-50

considered to be stale. To classify the sensor state as “Up” or “Down”, all the sensor data received in the past second must be “high” or “low” respectively. The absolute score for these two states is 50 points, the second highest score among the sensor states present.

To classify the sensor state as “Uptrigger”, the sensor data produced in the last second must have a transition from “low” state to “high” state, with the “high” state being in place for more than 300ms. The reasoning for this is, that the passing of 300ms in the new “high” state ensures that the sensor indeed transitioned from a previously “inactive” state ton an “active” one, indicating user presence. The 300ms window is adjustable, but after experimentation with the sensors it was deemed to perform better than other settings. To classify the sensor state as “Downtrigger” the same rules apply, but the transition must be from “high” to ”low” state. The “Uptrigger” and “Downtrigger” states represent the most recent and reliable information present in the system that should be highly valued. For this reason, their absolute scores are the largest among the sensor states present in the system.

Should the transition be not as old as 300ms, the sensor state will be characterized as “Rising” and “Falling” accordingly. This state indicates some activity in regards to the sensor. However, since the transition event happened very recently, it might still be just a spurious event or a fringe activation of the sensor. For this reason, the absolute scores for these values is comparatively lower to the “Up”,“Down”,“Uptrigger” and “Downtrigger” states.

The final sensor state is a state of “Flux”, where three or more events occurred in the past second. This state can occur during the entrance of a user in the detection area under an unusual approach angle, or when the user leaves a detection area. With the lowest absolute score (5) among the other states, it indicates that the user is at the fringes of the detection area of the sensor, an information that is unreliable by its nature, but should still be taken into consideration.

The above seven states were introduced in order to account for the behaviour of the sensor used in the system. The state classification procedure acts as a “debounce” mechanism for the sensor’s output, whereas also trying to identify the absolute newest and reliable information regarding sensor state change, codifying it in terms of “Uptrigger” and “Downtrigger” events.

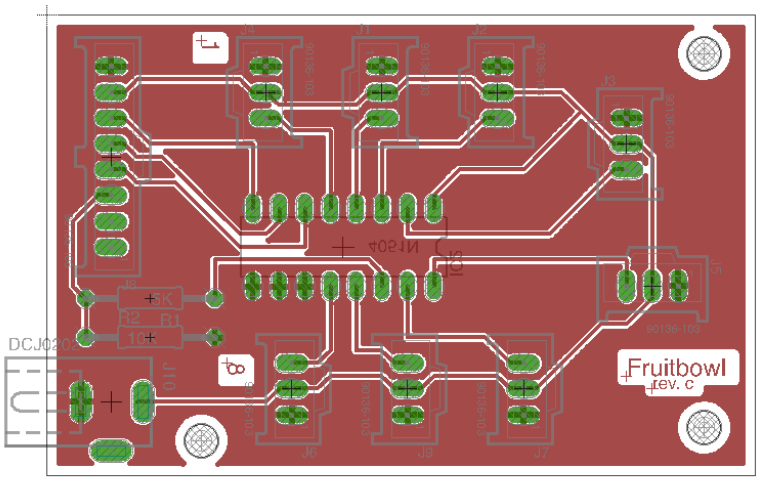

Figure 4.8: Extension Board Schematic

4.4 Implementation Details And Cost

4.4.1 Implementation Details

For the creation of the sensor pod, the popular Arduino embedded platform was utilized. More specifically, the Arduino Fio ver. 3 is the microcontroller unit used in this project. To transmit data, the popular XBee low power wireless ZigBee modules were used. To sample the output sensors, a special extension board was designed that utilizes an 8-channel digital multiplexer, part number 74HC151 of the 7400 transistor-to-transistor integrated circuit series. The layout of the extension board can be seen in 4.8.

The three sensor pods communicate over ZigBee with the base station, which is another Arduino MCU with the XBee module. The overall setup can be seen in 4.9. This base station is connected to a PC via a usb serial connection. This PC is responsible to process the incoming data from the sensors and perform the area evaluation algorithm presented in the previous section, implemented in Java. Furthermore, this PC is also responsible for visualization of the data.

The visualization of data was achieved using Java and an the JOGL library that offers Java bindings for OpenGL. Regarding the room, the wall where the entrance is located, its storage space as well as the two beds present in the room are visualized. Moreover, two special rendering modes exist: one that renders all the unique coverage areas, colored by the number of sensors that cover them (as seen in Fig. 4.4) and another that shows the individual coverage of a pod/sensor (as seen in Fig. 4.6 and Fig. 4.5).

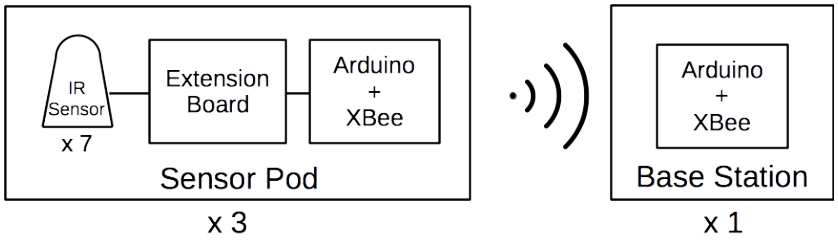

Figure 4.9: Fruitbowl construction

The last visualization mode is responsible for visualizing the most probable area that the user is located at. Such examples will be presented in the experimental section.

Regarding the sensor data acquisition process, each sensor is successively polled three times, for a total polling duration of 21ms. Should there be any change to any of the seven sensors, the pod will send a 1-byte character to the base station, describing the state of each sensor in the pod. Starting from the lower bit, a "zero" value indicates that the sensor number zero is inactive, whereas a "one" value indicates that the sensor is active. The eighth bit remains unused. If there is no change in the state of the sensors, the sensor pod continues polling and does not contact the base station.

The sensor output timeline for one of the experiments presented in the next section can be seen in Fig. 4.10. In this figure, events denoted with a red color indicate that the sensor was reported to be inactive, whereas events marked with green color signify that the corresponding sensor was active at that given time. The three wide bands of clustered events represent the three pods that were used. The lower band of seven sensors with ID from 10 to 16 represent the first pod, the sensors with ID 20 to 26 represent the second pod, and finally the sensors with ID from 30 to 36 represent the third pod. The sensor with a zero unit digit in their ID represent the sensor facing directly downward, and starting with sensors that have a 1 in their unit digit ID are marked with an arrow (see the right lower image of Fig. 4.2), with the rest of the sensors following clockwise.

4.4.2 Cost

A cost breakdown for the deployment in a room using three sensor pods can be seen in Tab. 4.2. The prices reported were those during the finalization of the project (2nd quarter of 2014). Of these, the most notable cost is the custom 3D printed casing, costing $110 in raw filament material. Furthermore, a set of seven IR sensors costs as much as $70. Finally, an Arduino

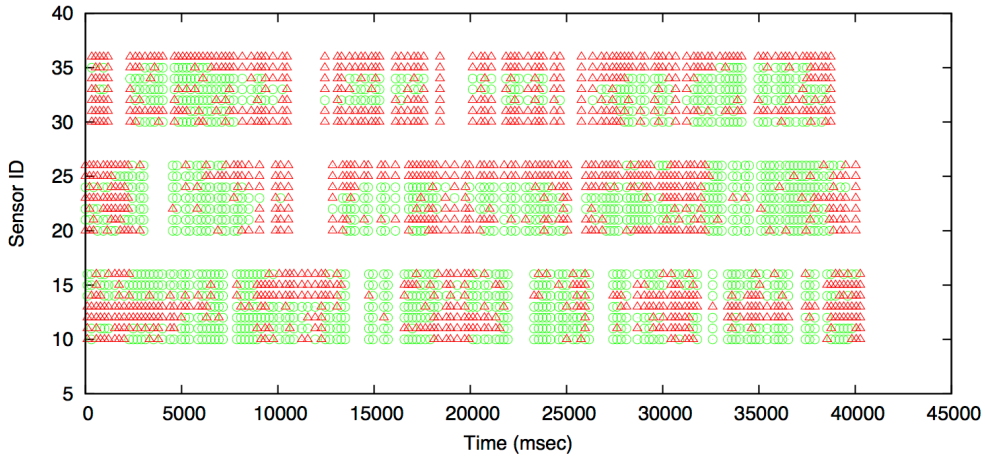

Figure 4.10: Sensor Output Timeline

Fio with an XBee module also costs approximately $68, bringing the total cost of a pod to about $250.

Since the project's finalization, the 3D printing technology has progressed, and filament prices have dropped quite dramatically. It is estimated that the custom 3D pod can now be printed with as few as $15, a huge improvement over the first attempt. The IR sensor pricing has not changed much, and now alternative MCUs to Arduino are widely available and could be used instead, in order to further reduce costs. All in all, a realistic price for a new pod would come down to as low as $120 if it was built today.

4.5 Experiments

4.5.1 Room Setup

The experiments presented in this chapter were conducted in a 5mx4mx2.4m room, as seen in Fig. 4.11. On the ceiling, three sensor pods were installed in the locations depicted as red dots. Major furniture as well as the closet space were taken into consideration and accounted for, properly taking up space in the octree.

Four different experiment runs were conducted. The path of each run can also be seen in Fig. 4.11. In each of these runs, when the user had to perform a turn or reverse his course, the user would also stay in place momentarily, for up to 3 or 4 seconds. The runs were chosen as typical paths that the user would follow inside a room. Moreover, a substantial part of the routes

Table 4.2: Deployment Cost Breakdown

Part Name	Price	Qty.	Total
Sensor Pod (x1)			
AMN31111	~$10	7	$70
Arduino Fio	~$25	1	$25
XBEE pro S1	~$38	1	$38
Custom 3D printed pod	~$110	1	$110
Custom multiplexer PCB	~$6	1	$6
Cables and connectors	~$3	1	$3
Subtotal			~$252
Base Station (x1)			
Arduino Fio	~$25	1	$25
XBEE pro S1	~$38	1	$38
Subtotal			~$68
Deployment Cost in Room			
Sensor Pod	~$252	3	$756
Base Station	~$68	1	$68
TOTAL			$824

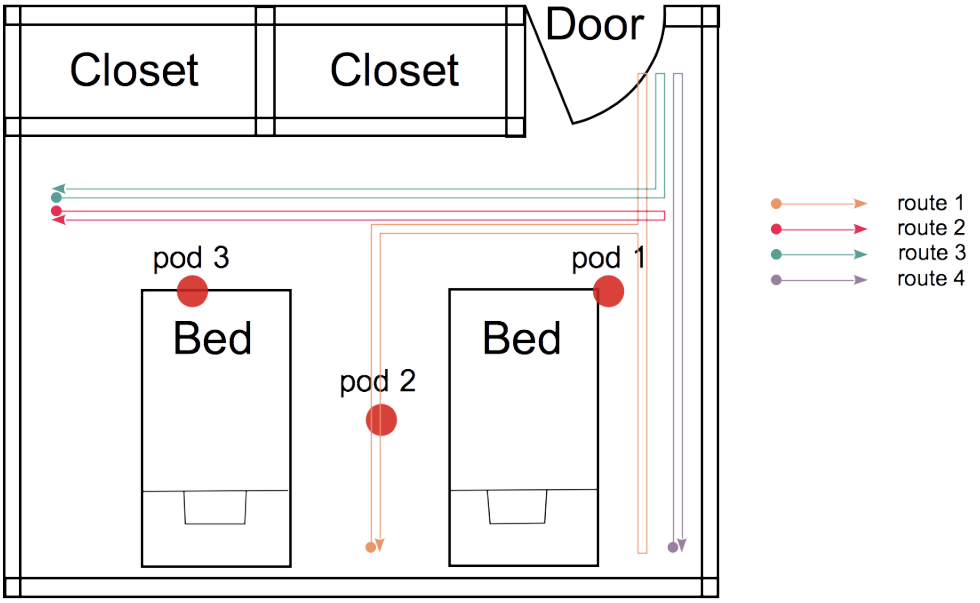

Figure 4.11: Top View of Experimental Room and Run Paths

selected also focus on pathways at the edges of the room, areas that are more challenging for accurate detection due to their lower coverage by sensors.

Finally, for each run, the user would start his movement inside the room after all the sensors have been verified to be in an "inactive" state in order to avoid affecting the results of the experiment.

4.5.2 Evaluation Approach

To evaluate the accuracy of the system, a number of still frames for each run where obtained as follows. First, the runs where recorded using a video camera. Second, the output of the visualization subsystem would also be recorded, with its background removed and its rendering angle matching that of the camera recording. The top 2% of the areas is visualized, with red areass the most probable, with the color decaying towards yellow for slightly less probable areas. Finally, after superimposing the output of the visualization on to the recorded run, a number of still frames were extracted for each run. For the first run, 10 frames per second where evaluated. This proved to be excessive and thus, for the remaining 3 runs, only 3 frames per second where evaluated.

Table 4.3: Frame Evaluation Results

Route	Total Frames	Avg. Score	Bad	Poor	Fair	Good	Excellent
Route 1	497	3.62	22	90	107	112	166
Route 2	57	3.07	3	17	17	13	7
Route 3	88	3.11	13	11	31	19	14
Route 4	74	3.20	6	11	32	12	13

After obtaining the still frames from the run, the system was evaluated using a qualitative method. Each frame was evaluated using a scale of 1 to 5 (bad, poor, fair, good, excellent) in regards to the reported location. The definitions for each level of the scale where the following:

1. Bad - Wide scattered and/or fragmented area, completely unrelated to the user's position.
2. Poor - Distance error of more than 50cm.
3. Fair - Focused area with distance error of less than 50cm, or wide detection area with the user residing inside it.
4. Good - Focused detection area around the user, with small extensions and protrusions.
5. Excellent - Tight detection around the user.

Example frames for each of the above categories can be seen in Fig. 4.12.

4.5.3 Run Result Analysis

Overall Evaluation The frame evaluation results for each run can be seen in Tab. 4.3. In summary, the average score of these runs varies from the lowest score of 3.07 up to the highest score which was 3.62. Thus, the results indicate that the system produced "fair" to "good" location estimation, usually within 50cm of error for these runs. Since the four routes that were selected are thought to be representative, the system performance should not significantly differ in most other cases.

The aggregate frame evaluation paints the system performance in a broad stroke. However, by examining the data of the first run more closely, further nuances of the system are readily perceptible.

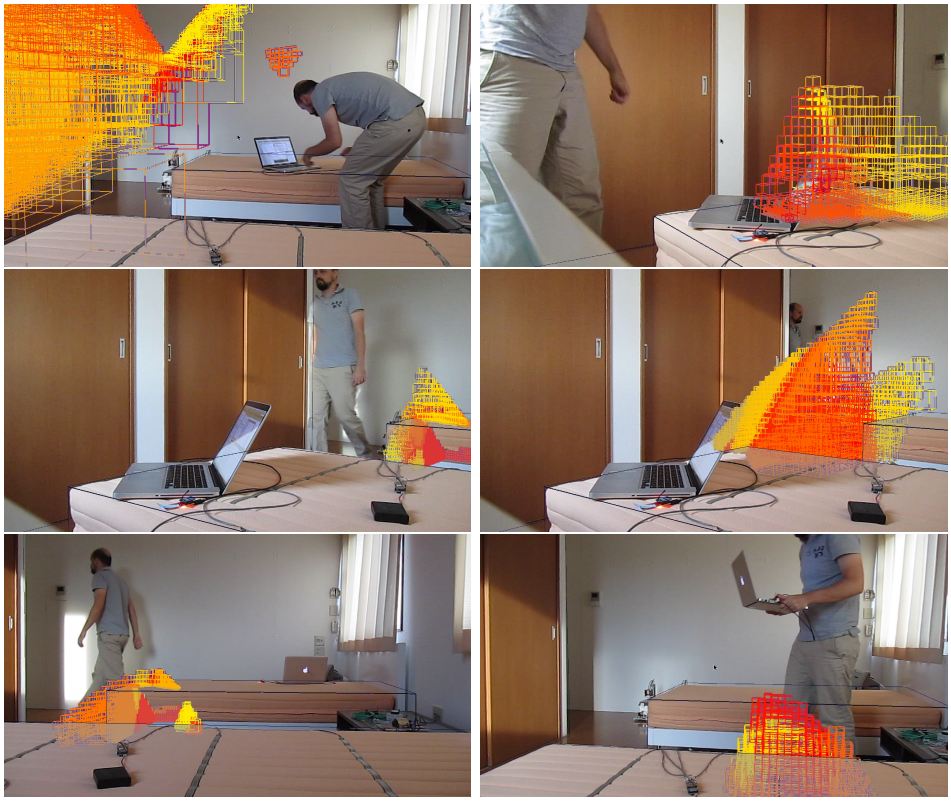

Figure 4.12: Examples of evaluation frames. Top left: a bad frame. Top right: a poor frame. Middle left: a fair frame (near miss). Middle right: a fair frame (large area). Bottom left: a good frame. Bottom right: an excellent frame.

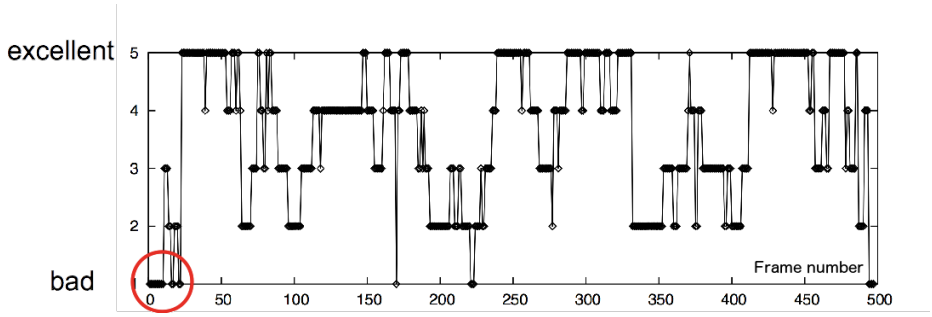

Figure 4.13: Frame Evaluation Timeline for Route 1

Bad Frames and Loss/Lack of Information In Fig. 4.13 the frame evaluation results for the first route are presented in a time line. The first observation that is readily apparent is the tendency of bad frames to appear at the beginning and the end of the run. Out of the 22 frames classified as "bad", only 4 frames appear in the middle of the run. The reason for this behaviour is the starting condition of these experiments; before the user starts to move, all sensors start from an "inactive" state, needing a couple of seconds to respond to the initial movement of the user. During this "start-up" time, location detection is unreliable.

A similar phenomenon also appears at the end of the run, where the user sits still at his final position, for a period of 3 to 5 seconds. In this "wind-down" period, sensors gradually move to an "inactive" state and the location detection again becomes unreliable.

This sort of behaviour is consistent with either *lack or loss of meaningful information* currently in the system; should all (or at least most) of the sensors be inactive, it is impossible to deduce the location of the user with any accuracy. During start-up, this lack of information is slowly overcome after the first couple of seconds where new information from the sensors is entering the system. During the wind-down, the system gradually loses information, as more and more sensors tend to transition to an "inactive" state.

Lagging Detection and "Strong Gravitational Effect" Another observation from Fig. 4.13 is that there are two bands where "poor" frames are prevalent: from frame 190 to frame 235, and from frame 330 to frame 410. In these two time periods, the reported location from the system would consistently be behind the actual location, giving the impression that it was "lagging" behind, trying to catch up with the user's actual location. Two representative frames of these periods can be seen in Fig. 4.14.

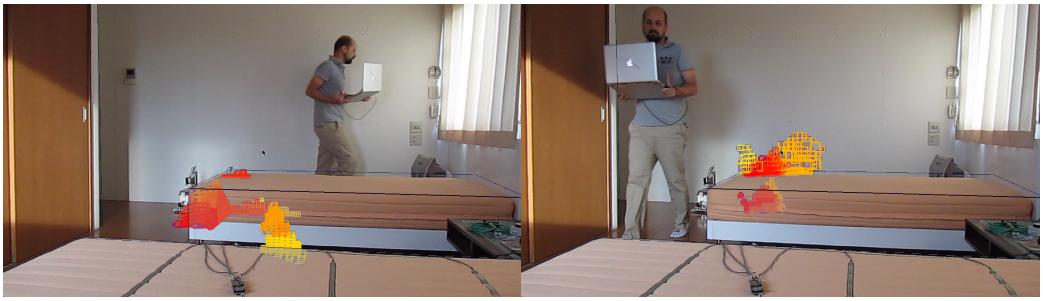

Figure 4.14: Lagging User Location Detection

This readily apparent discrepancy can be explained. The user just happened to pass through an area that is heavily covered by multiple sensors, thus setting most of the sensors present in the system to an "active" state. However, after the user continues his movement through the room, sensors that should no longer detect the user need time to revert back to an "inactive" state. Until these sensors become inactive, the reported location will heavily gravitate towards the areas with coverage by a greater number of sensors (perceived as a "strong gravitational effect"), usually lagging behind the user's current position.

Prospective Countermeasures The phenomena exhibited in the two previous paragraphs are a direct result of the sensor's operational behaviour. Apart from changing the sensor to some other model that has behaviour more closely aligned to the ideal behaviour as described in Sect. 4.3.2, there are a few prospective solutions that could be employed, but have yet to be implemented.

To combat the loss of information resulting from the user not moving, a cut-off point where further sensor transitions to inactive state are ignored, and the user's last position is retained. Deciding this cut-off point may have a relation to the average number of sensors covering the unique areas generated in the space subdivision step. During detection, a greater number of active sensors at a given moment should lead to location predictions with higher confidence, so the cut-off point may be alternatively related to the degree of confidence or accuracy necessary for the occasion.

To combat the "strong gravitational effect" where the predicted user location would gravitate towards areas covered by a greater number of sensors, further prioritizing newly incoming sensor information could lead to an escape from the heavily covered area. Although in the area evaluation algorithm presented in Sect. 4.3.3, newer information (encoded as uptrigger/downtrigger states) is valued with a greater score, the problem still persists. Another ap-

proach would be to have a "supersede" mechanism, where sensors that have very small overlapping areas are treated as exclusive sets. In such a set, the sensor that most recently switches to an active state supersedes the other sensors; its score is the only one considered during the area evaluation step and the score of all other conflicting sensors is reduced to zero. Additionally, it may prove beneficial to remove very small but highly covered areas from the octree itself; such areas seem to be at the core of the "strong gravitational effect", thus by removing such areas the effect may be less pronounced. Finally, it may be interesting to formulate alternative designs of the sensor pods and pod positions as a cobinatorial problem on which local search and genetic algorithms may try to find solutions in terms of optimal coverage and evenness of space subdivision, avoiding areas that could be a hot-spot and thus avoiding the strong gravitational effect by design.

4.6 Conclusions and Future Work

In this chapter, a user indoor location system based on infrared motion sensors and space subdivision was presented. Its main operation principle is the subdivision of space into areas covered by a unique combination of infrared sensors. By examining the state of the sensors in the scene, it is now possible to identify the most probable area in which the user is located at.

As a system which is based on passive infrared sensors, there are several inherent advantages, such as minimum intrusiveness and ease of use. Furthermore, due to the design of the sensor pod, the system is extendable and can be adapted for use in many different types of rooms. Finally, a single sensor pod could be built with a price as low as $120, making the system relatively inexpensive.

In its current state, the proposed indoor location system has the following limitations. First, the system cannot identify and distinguish between individuals, it can only report their location. Furthermore, the system currently cannot detect and report the locations of multiple individuals in a room at the same time. Finally, the system exhibits "loss of information" and "strong gravitational fields" as explained in Sect. 4.5.3.

Except from the first limitation (the inability to identify users), the rest of the limitations are associated closely with the behaviour of the AMN31111 sensor used in the system. Although certain countermeasures can be taken (such as identifying unique sensor states for this sensors and assigning different weights to them), this sensor exhibits behaviour that is significantly different from that of the "ideal sensor" described in Sect. 4.3.2.

Even with these limitations, the system was able to detect the user's

location within 50 cm, thus validating the core idea of space subdivision.

As future work, the use of a different sensor that exhibits behaviour closer to that of an ideal sensor should be pursued. This would solve (or at least mitigate) the "strong gravitational field" effect. Next, simultaneous location information for multiple users in a room should be implemented. A possible approach for this would be to modify the area evaluation algorithm as follows: after the most probable area is found, examine if there is a set of sensors that although report user presence were not in the initial most probable area. Should such a sensor set exist, re-run the area evaluation algorithm but with elevated scores for these sensors. This should be sufficient to identify the next most probable area, and so on, until all sensors reporting presence have been used. Finally, by integrating information from other external systems or different types of sensors (pressure pads, ultrasonic sensors, etc.), non-intrusive user identification with reasonable accuracy may be possible.

Chapter 5

In Conclusion

5.1 Summary of this Research

This research effort led to the development of a modern home service platform. This platform boasts a number of features that should be part of any advanced home service platform, especially those that aspire to achieve widespread use by consumers.

In chapter 2 the management of environmental resources is discussed. To the author's knowledge, the proposed platform is the first home service platform that attempts to handle physical characteristics of space such as illumination as a first-class resource. Other contemporary platforms do not attempt to handle these resources and forfeit control of them to services. However, in comparison to the proposed system, the fact that smart services can only have incomplete knowledge regarding the effects that a device has on its environment makes it significantly more difficult to effectively control environmental resources. Furthermore, although there have been mentions of conflict detection in the physical domain in other works, effective conflict resolution schemes have yet to be proposed. This research addresses this major point.

In chapter 3 the conflict detection and resolution features of the platform in regards to device conflicts were introduced. With primitives such as device access rights, service and user priority, a notification mechanism and condition sets, the development of reactive smart services now becomes simpler, giving the developers the tools necessary to create ubiquitous smart services for the home environment. These services are well behaved and exhibit predictable behaviour. With the use of these primitives, run-time conflict resolution is addressed thoroughly, putting the proposed platform at the cutting edge of this research field.

In chapter 4 an indoor location system based on passive infrared sensors was presented. The system can provide the location of a user with reasonable accuracy, in most cases with a margin of error of only 50 cm. Due to its construction, the location system is extensible, non-intrusive and of comparatively low cost, especially as the prices of 3D printers tend to fall over time. However, in its current iteration it has a few significant drawbacks, namely its inherent inability to concurrently report the position of more than one user at a time as well as to differentiate among different users. Nevertheless, its core idea is simple and demonstrably effective, a solid foundation on which more feature-rich location systems may be built.

5.2 The Importance and Impact of this Research

This research represents the author's vision for a modern, sophisticated home service platform as a vehicle to achieve ubiquitous and pervasive intelligence in the home. Although in each chapter a different facet of the system is introduced, it is the whole picture that is created by these parts that is important; should any of these parts be missing, the value proposition of the home service platform as a whole becomes far less impressive.

It is imperative that any reasonably advanced home service platform addresses the three topics discussed in the chapters of this thesis:

- detection and resolution of device conflicts,
- detection and resolution of environmental conflicts,
- a functional user indoor location system.

Perhaps it may be easier to understand the importance of these three critical components of the platform by conducting a simple thought experiment: imagine what would happen if any of these components were absent. Should run-time conflict detection and resolution be unavailable, user experience will suffer. Unpredictable operation of devices, lights turning on and off and non-deterministic behaviour of services is going to be the result, leading to the frustration of the user at best, or as a worst case scenario creating disillusionment, leading the user to believe that the whole premise of an intelligent home is untenable and a waste of money, time and effort.

The lack of a functional user indoor location system will again significantly hamper the ability of the home service platform to adapt to the user's actions. To the author's opinion, the user's location is possibly the single

most important piece of context information, based on which smart services can react and exhibit intelligence. Furthermore, the lack of user location information would further hinder the ability of the system to perform environmental conflict resolution, which is a major feature of this platform. Although the solution described in chapter 4 is far from perfect, it values the user's privacy above all and does not require active participation from the user. Such a non-intrusive user location system is a perfect fit for the future smart home.

Finally, the last piece of the puzzle towards the home service platform of the future is the management of environmental resources. Conflict over environmental resources again has an adverse effect on the user's experience, but current home service platforms fail to propose a resolution scheme. Furthermore, trusting the services to operate devices appropriately makes the design and implementation of smart services a daunting task. The proposed platform provides a method of handling environmental resources with inherent conflict resolution capabilities which is fast and produces reasonable results as demonstrated in the case of illumination.

As a whole, the proposed service platform becomes more than the sum of its parts. For the first time, the platform provides all the necessary abstractions to program and develop new reactive smart services easily and with relatively little effort. Its feature list is comprehensive and addresses the problem of service conflicts in regards to device and environmental resources adequately.

In closing, this research pushes the boundaries of what is possible with the use of a sophisticated home service platform and advances the state of smart homes. Future service platforms will inevitably have to provide similar or equivalent features or face obsolescence.

Bibliography

[1] Android Operating System. http://www.android.com/.

[2] Apple HomeKit. Accessed: 2015-05-28. [Online]. Available: https://developer.apple.com/homekit/

[3] Bluetooth. https://www.bluetooth.com/.

[4] ECHONET Lite Protocol Specifications. http://www.echonet.gr.jp/english/spec/index.htm.

[5] IFTTT. https://ifttt.com/.

[6] INSTEON. http://www.insteon.com/.

[7] Knx. https://www.knx.org/.

[8] nest. https://nest.com/.

[9] Philips Hue. http://www2.meethue.com/en-US/.

[10] Spin. http://spinroot.com.

[11] X10. http://www.x10.com/about-us/.

[12] Z-Wave. http://z-wavealliance.org/.

[13] ZigBee. http://www.zigbee.org/.

[14] P. Bahl and V. N. Padmanabhan, "Radar: an in-building rf-based user location and tracking system," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, 2000, pp. 775–784 vol.2.

[15] P. Belimpasakis and S. Moloney, "A platform for proving family oriented restful services hosted at home," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 2, pp. 690–698, May 2009.

[16] L. Blair, G. Blair, J. Pang, and C. Efstratiou, "Feature' interactions outside a telecom domain," in *Workshop on Feature Interaction in Composed Systems*, 2001.

[17] F. Calvino, M. L. Gennusa, G. Rizzo, and G. Scaccianoce, "The control of indoor thermal comfort conditions: introducing a fuzzy adaptive controller," *Energy and Buildings*, vol. 36, no. 2, pp. 97 – 102, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0378778803001312

[18] B. Chang and X. Zhang, "Design of indoor temperature and humidity monitoring system based on cc2430 and fuzzy-pid," in *Cross Strait Quad-Regional Radio Science and Wireless Technology Conference (CSQRWC), 2011*, vol. 2, July 2011, pp. 980–984.

[19] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and P. Bahl, "The home needs an operating system (and an app store)," in *HotNets IX*. ACM, October 2010.

[20] ——, "An operating system for the home," in *NSDI*. USENIX, April 2012.

[21] S. Feldmann, K. Kyamakya, A. Zapater, and Z. Lue, "An indoor bluetooth-based positioning system: Concept, implementation and experimental evaluation," in *International Conference on Wireless Networks*, W. Zhuang, C.-H. Yeh, O. Droegehorn, C.-T. Toh, and H. R. Arabnia, Eds. CSREA Press, 2003, pp. 109–113.

[22] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[23] N. Georgantas, V. Issarny, S. B. Mokhtar, Y.-D. Bromberg, S. Bianco, G. Thomson, P.-G. Raverdy, A. Urbieta, and R. S. Cardoso, *Handbook of Ambient Intelligence and Smart Environments*. Boston, MA: Springer US, 2010, ch. Middleware Architecture for Ambient Intelligence in the Networked Home, pp. 1139–1169. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-93808-0_42

[24] K. Gill, S. H. Yang, F. Yao, and X. Lu, "A zigbee-based home automation system," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 2, pp. 422–430, May 2009.

[25] C. Gomez and J. Paradells, "Wireless home automation networks: A survey of architectures and technologies," *Communications Magazine, IEEE*, vol. 48, no. 6, pp. 92–101, June 2010.

[26] M. Gotze, W. Kattanek, and R. Peukert, "An extensible platform for smart home services," in *Systems, Signals and Devices (SSD), 2012 9th International Multi-Conference on*, March 2012, pp. 1–6.

[27] R. Gupta, S. Talwar, and D. P. Agrawal, "Jini home networking: a step toward pervasive computing," *Computer*, vol. 35, no. 8, pp. 34–40, Aug 2002.

[28] A. Gurek, C. Gur, C. Gurakin, M. Akdeniz, S. K. Metin, and I. Korkmaz, "An android based home automation system," in *High Capacity Optical Networks and Enabling Technologies (HONET-CNS), 2013 10th International Conference on*, Dec 2013, pp. 121–125.

[29] R. Harper, *Inside the Smart House*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003.

[30] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster, "The anatomy of a context-aware application," in *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, ser. MobiCom '99. New York, NY, USA: ACM, 1999, pp. 59–68. [Online]. Available: http://doi.acm.org/10.1145/313451.313476

[31] D. Hauschildt and N. Kirchhof, "Advances in thermal infrared localization: Challenges and solutions," in *Indoor Positioning and Indoor Navigation (IPIN), 2010 International Conference on*, Sept 2010, pp. 1–8.

[32] F. Herrera, M. Lozano, and J. Verdegay, "Tuning fuzzy logic controllers by genetic algorithms," *International Journal of Approximate Reasoning*, vol. 12, no. 3–4, pp. 299 – 315, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0888613X9400033Y

[33] H. Hu, D. Yang, C. Fu, and W. Fang, "Towards a semantic web based approach for feature interaction detection," in *Software Technology and Engineering (ICSTE), 2010 2nd International Conference on*, vol. 2, Oct 2010, pp. V2–330–V2–334.

[34] H. Ishikawa, Y. Ogata, K. Adachi, and T. Nakajima, "Building smart appliance integration middleware on the osgi framework," in *Object-Oriented Real-Time Distributed Computing, 2004. Proceedings. Seventh IEEE International Symposium on*, May 2004, pp. 139–146.

[35] M. Jahn, M. Jentsch, C. R. Prause, F. Pramudianto, A. Al-Akkad, and R. Reiners, "The energy aware smart home," in *Future Information Technology (FutureTech), 2010 5th International Conference on*, May 2010, pp. 1–8.

[36] D.-S. Kim, J.-M. Lee, W. H. Kwon, and I. K. Yuh, "Design and implementation of home network systems using upnp middleware for networked appliances," *IEEE Transactions on Consumer Electronics*, vol. 48, no. 4, pp. 963–972, Nov 2002.

[37] J. E. Kim, G. Boulos, J. Yackovich, T. Barth, C. Beckel, and D. Mosse, "Seamless integration of heterogeneous devices and access control in smart homes," in *Intelligent Environments (IE), 2012 8th International Conference on*, June 2012, pp. 206–213.

[38] M. Kolberg, E. Magill, and M. Wilson, "Compatibility issues between services supporting networked appliances," *Communications Magazine, IEEE*, vol. 41, no. 11, pp. 136–147, Nov 2003.

[39] P. Leelaprute, T. Matsuo, T. Tsuchiya, and T. Kikuno, "Detecting feature interactions in home appliance networks," in *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008. SNPD '08. Ninth ACIS International Conference on*, Aug 2008, pp. 895–903.

[40] P. Leelaprute, "Resolution of feature interactions in integrated services of home network system," in *Communications, 2007. APCC 2007. Asia-Pacific Conference on*, Oct 2007, pp. 363–366.

[41] K. Li, H. Su, and J. Chu, "Nonlinear model reduction for simulation and control of temperature distribution in air conditioned rooms," in *Control Conference (CCC), 2012 31st Chinese*, July 2012, pp. 1301–1306.

[42] R. Mautz and S. Tilch, "Survey of optical indoor positioning systems," in *Indoor Positioning and Indoor Navigation (IPIN), 2011 International Conference on*, Sept 2011, pp. 1–7.

[43] M. Minakais, S. Mishra, and J. T. Wen, "Groundhog day: Iterative learning for building temperature control," in *Automation Science and Engineering (CASE), 2014 IEEE International Conference on*, Aug 2014, pp. 948–953.

[44] H. Mirinejad, K. C. Welch, and L. Spicer, "A review of intelligent control techniques in hvac systems," in *Energytech, 2012 IEEE*, May 2012, pp. 1–5.

[45] R. Morla and N. Davies, "A framework for describing interference in ubiquitous computing environments," in *Pervasive Computing and Communications Workshops, 2006. PerCom Workshops 2006. Fourth Annual IEEE International Conference on*, March 2006, pp. 4 pp.–635.

[46] S. Munir and J. A. Stankovic, "Depsys: Dependency aware integration of cyber-physical systems for smart homes," in *Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on*, April 2014, pp. 127–138.

[47] T. Nakajima and I. Satoh, "A virtual overlay network for integrating home appliances," in *Applications and the Internet, 2002. (SAINT 2002). Proceedings. 2002 Symposium on*, 2002, pp. 246–253.

[48] L. Ni, Y. Liu, Y. C. Lau, and A. Patil, "Landmarc: indoor location sensing using active rfid," in *Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference on*, March 2003, pp. 407–415.

[49] M. Okada, H. Aida, H. Ichikawa, and M. Miki, "Design and implementation of an energy-efficient lighting system driven by wireless sensor networks," in *Mobile Computing and Ubiquitous Networking (ICMU), 2015 Eighth International Conference on*, Jan 2015, pp. 114–119.

[50] M. S. Pan, L. W. Yeh, Y. A. Chen, Y. H. Lin, and Y. C. Tseng, "A wsn-based intelligent light control system considering user activities and profiles," *IEEE Sensors Journal*, vol. 8, no. 10, pp. 1710–1721, Oct 2008.

[51] N. Papadopoulos, A. Meliones, D. Economou, I. Karras, and I. Liverezas, "A connected home platform and development framework for smart home control applications," in *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on*, June 2009, pp. 402–409.

[52] H. Park, J. Burke, and M. B. Srivastava, "Design and implementation of a wireless sensor network for intelligent light control," in *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, April 2007, pp. 370–379.

[53] T. Perumal, A. R. Ramli, and C. Y. Leong, "Interoperability framework for smart home systems," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 4, pp. 1607–1611, November 2011.

[54] A. Rathnayaka, V. Potdar, and S. Kuruppu, "Evaluation of wireless home automation technologies," in *Digital Ecosystems and Technologies Conference (DEST), 2011 Proceedings of the 5th IEEE International Conference on*, May 2011, pp. 76–81.

[55] D. Romero, G. Hermosillo, A. Taherkordi, R. Nzekwa, R. Rouvoy, and F. Eliassen, "Restful integration of heterogeneous devices in pervasive environments," in *Distributed Applications and Interoperable Systems*, ser. Lecture Notes in Computer Science, F. Eliassen and R. Kapitza, Eds. Springer Berlin Heidelberg, 2010, vol. 6115, pp. 1–14. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13645-0_1

[56] A. Sempey, C. Inard, C. Ghiaus, and C. Allery, "Fast simulation of temperature distribution in air conditioned rooms by using proper orthogonal decomposition," *Building and Environment*, vol. 44, no. 2, pp. 280 – 289, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360132308000413

[57] M. Sioutis, J. Kim, A. Lim, and Y. Tan, "A home service deployment platform with support for detection and resolution of physical resource conflicts," in *Consumer Electronics (GCCE), 2012 IEEE 1st Global Conference on*, Oct 2012, pp. 333–336.

[58] M. Sioutis, Y. Lim, and Y. Tan, "Achieving optimal illumination conditions using local search," in *Consumer Electronics (GCCE), 2015 IEEE 4th Global Conference on*, Oct 2015, pp. 168–172.

[59] M. Sioutis and Y. Tan, *Distributed, Ambient, and Pervasive Interactions: Second International Conference, DAPI 2014, Held as Part of HCI Interational 2014, Heraklion, Crete, Greece, June 22-27, 2014. Proceedings.* Cham: Springer International Publishing, 2014, ch. User Indoor Location System with Passive Infrared Motion Sensors and Space Subdivision, pp. 486–497. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-07788-8_45

[60] M. Sioutis, K. Tanaka, Y. Lim, and Y. Tan, *Ambient Assisted Living. ICT-based Solutions in Real Life Situations: 7th International Work-Conference, IWAAL 2015, Puerto Varas, Chile, December 1-4, 2015, Proceedings.* Cham: Springer International Publishing, 2015,

ch. Towards Resilient Services in the Home, pp. 113–124. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-26410-3_11

[61] A. Smith, H. Balakrishnan, M. Goraczko, and N. Priyantha, "Tracking moving devices with the cricket location system," in *Proceedings of the 2Nd International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '04. New York, NY, USA: ACM, 2004, pp. 190–202. [Online]. Available: http://doi.acm.org/10.1145/990064.990088

[62] C. Soares, R. S. Moreira, R. Moria, J. Torres, and P. Sobral, "Prognostic of feature interactions between independently developed pervasive systems," in *Prognostics and Health Management (PHM), 2012 IEEE Conference on*, June 2012, pp. 1–8.

[63] M. Sugano, T. Kawazoe, Y. Ohta, and M. Murata, "Indoor localization system using rssi measurement of wireless sensor network based on zigbee standard," in *Wireless and Optical Communications*, A. O. Fapojuwo and B. Kaminska, Eds. IASTED/ACTA Press, 2006.

[64] M. Taylor, A. Chandak, Q. Mo, C. Lauterbach, C. Schissler, and D. Manocha, "Guided multiview ray tracing for fast auralization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 11, pp. 1797–1810, Nov 2012.

[65] B. Ur, J. Jung, and S. Schechter, "The current state of access control for smart devices in homes," in *Workshop on Home Usable Privacy and Security (HUPS)*. HUPS 2014, July 2013.

[66] P. A. Vicaire, Z. Xie, E. Hoque, and J. A. Stankovic, "Physicalnet: A generic framework for managing and programming across pervasive computing networks," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*. IEEE, 2010, pp. 269–278.

[67] E. U. Warriach, "State of the art: embedded middleware platform for a smart home," *Int. J. Smart Home*, vol. 7, pp. 275–294, 2013.

[68] T. Weise, *Global Optimization Algorithms – Theory Application*, 2nd ed. Thomas Weise, 2008, Accessed: 2015-08-30. [Online]. Available: http://www.it-weise.de/projects/book.pdf

[69] M. Wilson, E. Magill, and M. Kolberg, "An online approach for the service interaction problem in home automation," in *Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE*, Jan 2005, pp. 251–256.

[70] C.-L. Wu, C.-F. Liao, and L.-C. Fu, "Service-oriented smart-home architecture based on osgi and mobile-agent technology," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, no. 2, pp. 193–205, March 2007.

[71] J. Wu, L. Huang, D. Wang, and F. Shen, "R-osgi-based architecture of distributed smart home system," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 3, pp. 1166–1172, August 2008.

[72] J. Xiao, K. Wu, Y. Yi, and L. Ni, "Fifs: Fine-grained indoor fingerprinting system," in *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, July 2012, pp. 1–7.

[73] M. Yagita, F. Ishikawa, and S. Honiden, "An application conflict detection and resolution system for smart homes," in *Software Engineering for Smart Cyber-Physical Systems (SEsCPS), 2015 IEEE/ACM 1st International Workshop on*, May 2015, pp. 33–39.

[74] M. A. Zamora-Izquierdo, J. Santa, and A. F. Gomez-Skarmeta, "An integral and networked home automation solution for indoor ambient intelligence," *IEEE Pervasive Computing*, vol. 9, no. 4, pp. 66–77, October 2010.