Area of effect and compromising techniques for the detection and resolution of environmental conflicts between services in the Home Network System

By Marios Sioutis

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Professor Yasuo Tan

March, 2011

Area of effect and compromising techniques for the detection and resolution of environmental conflicts between services in the Home Network System

By Marios Sioutis (0910030)

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Professor Yasuo Tan

and approved by
Professor Yasuo Tan
Professor Yoichi Shinoda
Associate Professor Mikifumi Shikida

February, 2011 (Submitted)

Thank you!

This thesis could not have happened if I had not been blessed with the support and love of many people. The following is just a short list of people I would like to thank and dedicate this work.

Professor Yasuo Tan and all of my colleagues in Tan laboratory for the immense support and guidance that I received over the years. Especially I would like to thank my senpais, Okadasan, Nakatasan and Kiyoumisan, that helped me so much to get accustomed to life and research at JAIST.

My close friends at JAIST over the years, Peter, Dominik, Ming, Matou, Nakamura, Blaise, for the good and hard times we had together. Everyday living without you guys gets boring, I wish you the best!

My friends back at home, Nikos Kereres, Bidis, Antonis, Dimitris, Christoforas da Best and others, for their support, comic relief moments and countless hours of skype calling - I love you guys.

Professor Christos Douligeris, Dimitris Glynos, Antonis Petropoulos and all my friends back in cslabs at Unipi for shaping my undergraduate years. You have been a great influence and example for me to follow.

My family. Living away from you is hard.

My girlfriend Noula. Without your support I could not have pulled through this. Without your love, not many things would matter.

Dedicated to all of you, you help me shape my path in life. Thank you!

Abstract

In this research we deal with the problem of service conflicts in the home network system (hereby referred to as HNS). Conflicts arise due to the competition for resources among the services. We propose two methods for the detection of conflicts based on information regarding the ”Area of Effect” of those services and the use of compromising techniques for the resolution of a subset of these conflicts know as ”environmental conflicts”. To demonstrate the above ideas, we created a centralised resource management system, capable of performing conflict detection and resolution in the HNS using the above ideas.

Recently, networking technology is widely being deployed inside houses, allowing devices to communicate and share resources. There has been steady progress in the field of communication protocols and standardization and this has resulted in many protocols and technologies (Bluetooth, UPnP, DLNA, Echonet, others) that allow devices to interoperate.

Having solved the problem of interoperability for devices, the next step is to offer services to the home environment from an external service provider. The home is expected to become a standardized service deployment platform and services running on top of it will be able to harness its resources. Such a service platform is described by the Service Intermediary model. The characteristics of the environment (such as temperature, humidity, illumination and sound/noise levels) are also treated as resources.

There are two possible types of service conflicts in the service deployment environment described above:

- device conflicts, where services send conflicting requests towards the same device,
- environmental conflicts, where services operate devices that have conflicting effects on the environment (e.g. operating a heater and an air condition unit at the same time).

The algorithms that have been proposed so far for the conflict resolution involve *suspending at least one of the conflicting services in order to allow the remaining services run successfully*. While this is mostly true for device conflicts, we argue that in the case of environmental conflicts it may be possible to *avoid suspending a conflicting service and still having these services run with relative success*.

To be able to do so, we propose a centralized resource management system for the environmental properties which can be integrated in the deployment platform. This system has complete control over the devices that can affect the environment and it is tasked to fulfil the requests of services regarding the four environmental properties (illumination, temperature, humidity, sound/noise levels), acting as an abstraction layer between the services and the devices. The proposed system provides a service application programming interface (API) so that services can make requests for resources as well as an internal device API to control the devices in the HNS in a unified manner.

The system uses information regarding the ”Area of Effect” (AoE)of services and devices as well as compromising techniques to detect and resolve conflicts between services. We defined the AoE of a service (or a device) as the physical space for which the effect of that service (or device) lies between upper and lower intensity bounds that we are interested in. The system uses limited physics simulation to predict the effects and make estimation about AoEs.

Using this information, the system is able to detect conflicts by discovering overlapping AoEs. We take a two step approach: one *per-device* conflict detection step where the conflict is discovered by conflicting settings applied on a device inside the intended AoE and and a second step based on *estimations of overlapping AoEs of devices outside the originally intended scope*.

Regarding conflict resolution, we propose two main categories of compromising algorithms: space-based and intensity-based. Both of these types of algorithms do not require the suspension of any conflicting service.

Space-based compromising algorithms try to limit the AoEs of the conflicting algorithms. These algorithms strive to impose the requested intensity around the *anchor point* of a service. The anchor point is the epicentre of execution of the service and the point which the service is most interested in. These algorithms are effective when there is a large number of conflicting devices and their *locality* is good (i.e. the effects of the service tend to weaken sharply as the distance from the source increases).

Intensity-based compromising algorithms try to choose an *intermediate setting* for devices such that all conflicting services can be satisfied to the maximum possible degree. These algorithms are effective when there is a small number of devices that may also have bad locality.

The above algorithms search the solution space (possible combinations of settings for devices) so that the requests of the services are fulfilled as best as possible. We identified the problem as being an *optimization* problem and proceeded to examine various optimization techniques and their applicability. The implemented algorithms in this prototype system can be said to be

variations of local search algorithms combined with evaluation mechanics borrowed heavily from constraint programming.

Two space based algorithms (”greedy” and ”forfeit rights”) and one intensity based algorithm (”range search”) were implemented. Various combinations of conflict detection and conflict resolution techniques were tested in a total of three experiments (scenarios for illumination, temperature and sound/noise levels).

We argue that the use of the prototype system is a significant improvement over the alternatives: suspending a service, or suffering from non-deterministic or erratic behaviour of devices. Finally we believe that the proposed system augments the SI model and there is potential value in a combined system.

Contents

Chapter 1

Introduction

1.1 Home network system and services

In the past decade the number of home appliances that have networking features has been steadily increasing. Such devices have increased perceived value for the customer because of the extra features (remote control, access to the internet, others) that they provide. At the beginning, these devices used to have a very limited application scope, provided a very specific service to the user and were not designed with interoperability in mind.

It was not before long that device manufacturers realized the potential of designing network-capable appliances that can co-operate and share resources. From this fundamental change in design, protocols such as UPnP[17], Echonet[18] and DLNA[16] were developed. These protocols are backed by major organizations that are considered to be industry leaders (such as SONY, Microsoft Corp. and others) and have been adopted in the design of many commercial products that are readily available to consumers now.

Having somewhat solved interoperability and communication problems between devices, the next challenge is to offer services to the home environment in a flexible and standardized manner. Recently, various vendors offer services to the home using some proprietary hardware and software combination. These solutions have quite a few disadvantages such as redundant hardware, configuration complexity, possible conflicts between services and others. These problems will be explained in detail in section 2.1.

This research is focusing on the detection and resolution of environmental conflicts between services in the home network system (hereby referred as HNS).

1.2 Target of this research

In our research we discuss service conflicts in the home network system. We proceed to categorize conflicts between services and then propose new methods to detect and resolve a subcategory of these conflicts, named as ”environmental conflicts”. Our approach is based on the notion of the *”Area of Effect”* (from now on referenced as AoE) and the application of *compromising techniques*.

We propose a centralized system in the HNS that acts as a *resource manager*. This system will undertake the task to receive requests on behalf of the services and try to fulfil them to the best possible degree. Furthermore, this resource management system has a comprehensive set of programming APIs that can be used to design a service with minimum effort and for the control of devices and can also be integrated with the *SI model* for the provision of more complex and feature-rich services that adapt to the deployment environment.

The proposed system employs new techniques for the detection and resolution of conflicts between services based on the notion of AoE and compromising techniques. It is important to note that these compromising techniques can be used effectively for only a specific type of service conflict, which we named ”environmental conflict” .

1.3 Structure of this document

In chapter 2 we will explain the background regarding the home network system (from now on referenced as HNS) and services. We will proceed to describe different architectural models for the provision of services in the HNS and then describe what is know as feature interaction between services. We will finish chapter 2 by mentioning the methods currently available to detect and solve service conflicts and explain their weaknesses.

In chapter 3 we will discuss the various types of conflicts between services, focusing on the environmental conflicts. Then we will proceed to give the various definitions regarding the AoE. Namely, we will talk about the effect of a device and the effect of a service at a given point in space, as well as the AoE of a device and the AoE of a service. We will then proceed to express the problem of assigning settings to devices as an optimization problem and evaluate different methods to tackle it.

In chapter 4 we will give an overview of the design of the prototype system we developed, and describe the various APIs used internally to handle requests from the services. We will proceed to provide some implementation details about the system.

In chapter 5 we will present some experimental setups along with the results attained. The experiments will demonstrate the advantages of the proposed conflict detection and conflict resolution algorithms in various scenarios that may occur in the HNS.

Finally in chapters 6 and 7 we present possible future work and improvements as well as the final conclusions and remarks respectively.

Chapter 2

Background Research

2.1 Services in the home network system

In this section we will discuss the past and the current status of home services in the HNS. We will provide a definition of what constitutes a service in the HNS and present the different methods that are used for the provision of such services

A service in the HNS is a service provided to the user that satisfies a specific need. These services may offer a wide area of functionality, such as ”smart” security systems, regulation of energy consumption, context-aware environment and others. Such services are quite often provided by an external entity (a company) which from now on we will refer to as the Service Provider (SP).

2.1.1 Conventional models

The conventional method to provide services to the user in the HNS is to supply the user with specific, proprietary hardware that integrates tightly with the provided service. For example, a typical home security system (that essentially provides a home security service to the users) usually consists of many sensors that have to be retrofitted to almost all the doors and windows in a house, a device that provides the main user interface to the user and also acts as a general computation system, and possibly utilize some network connection to notify the security company of possible security breaches in the house.

Another example of a service provided in a very traditional fashion is netflix streaming services[15]. Using the netflix services a user is able to stream a wide variety of video content to a select array of ”netflix-ready” devices[1]. The video content is streamed directly from the service provider and arrives directly to a certified device able to handle the video stream.

[1] for a list of supported devices, see http://www.netflix.com/NetflixReadyDevices (accessed on 2010-02-20)

A representation of the conventional service provision model can be seen in figure 2.1. Lines between the SP and the HGW show the direct relationship between the two parties.

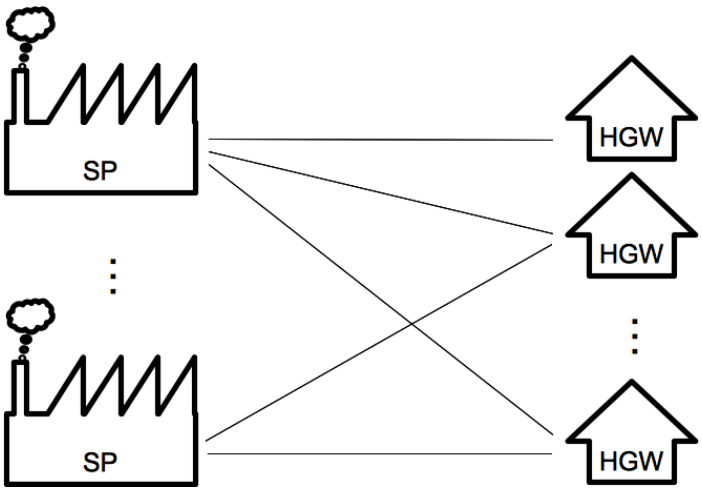

Figure 2.1: The conventional service provision model.

The common elements of the above solutions are the following:

- custom hardware equipment to support the service
- A client-server architecture between the service provider and the required service hardware installed directly at the HNS of the user
- the user must enter a contract with the service provider for every service he may wish to use.

Despite the simplicity of this conventional model of providing services, there are some severe disadvantages that prohibit the realization of a more rich, unified HNS. The biggest disadvantages of this conventional approach to services are the following:

1. *Usage of custom hardware*. In most services the hardware is restricted to only provide the service it was designated to provide. Other services usually cannot take advantage of the hardware designed to be used with another service. This may also lead to redundant or duplicate hardware for similar services.

2. *Configuration of the service*. Due to the fact that each service is using dedicated hardware, the configuration of this hardware is usually left to the inexperienced user.

3. *User interface*. The user interface for each service is usually provided through some proprietary software/hardware solution. This ruins the feeling of consistency between services and it is also more difficult to master the peculiarities of the different interfaces.

4. *Conflict between services*. Because the services act independently from each other, there is a high chance that some services may conflict over the use of resources or produce conflicting results. This degrades the perceived quality of experience for those services.

The above disadvantages all stem from the fragmentation of the deployment hardware. These inherent limitations of the conventional model hamper its scalability.

2.1.2 The Service Intermediary model

Service Provider, Service Intermediary, and Home Gateway

After realizing the limitations of the traditional model to provide services in the HNS, the Service Intermediary (SI) model was conceived. The core idea of this model is presented in [1]. The SI platform is designed to solve the problems of the traditional service model.

In the SI model, there are three main entities.

- The back-end Service Provider (SP). This is the actual provider of a specific service. The SP designs services that will be provided to the HNS, the final deployment target. This role is the only one that remains unchanged compared to the traditional model.
- The Home Gateway (HGW). The HGW acts as the gateway of the HNS to the outside world. Services can be executed directly on the HGW hardware. Moreover, the HGW acts as a software bridge, capable of controlling or passing command to other devices in the HNS.
- The Service Intermediary (SI) ,from which this model for service providing takes its name. The SI acts as an intermediary between the SP and the HGW. In this model, the HGW does not contact the SP to have access to a service. Instead, the SI aggregates services from various SPs and presents them to the user. The user makes a contract with the SI to have access to the services of the various SPs.

A representation of the relationships between the three parties can be seen in figure 2.2. Usually, SIs aggregate services from the SPs and then a house can enter a contract with an SI.

Advantages of the SI model

The presence of the SI is the major difference between the SI model and the traditional model of services. By having a service intermediary between the various service providers and the HNS, the problem of a fragmented deployment environment is resolved. This is because, the SI can enforce a specification of the hardware platform on which the services will be executed. The SI will be responsible to verify that all services deployed on the controlled platform do not misbehave and are consistent.

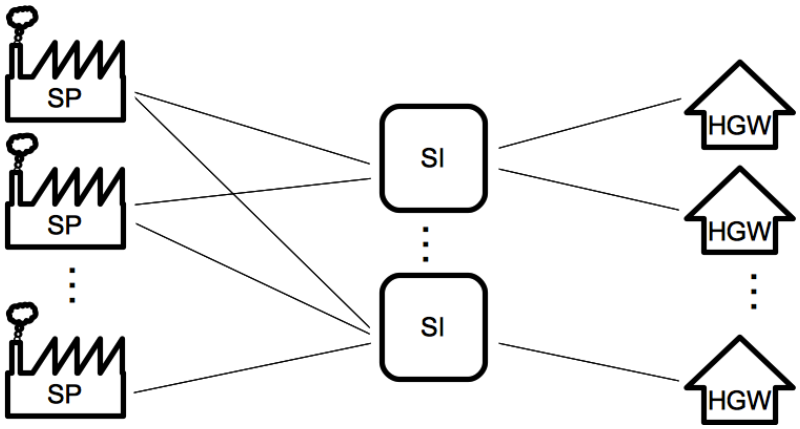

Figure 2.2: The SI service provision platform.

The existence of an SI saves the user of the hassle of having to make contracts with a substantial number of different SPs. Moreover, the problem of the configuration of a multitude of different devices is resolved, since the SI can take care of it. The configuration of the services will also be taken care by the SI, and we expect that the SI will be able to offer a comprehensive and consistent user interface, not possible otherwise. Furthermore, the duplication of hardware is minimized and the same hardware may now be used by many different services, increasing its value. Moreover, the SI will be able to provide the user with all the equipment that is necessary (depending on the services the user is subscribing to), or designate compatible equipment that can be used effectively as part of the platform. Finally, according to [2], it is possible to have computationally intensive services execute on computational nodes of the SI transparently, as in the case of cloud computing.

The benefits for the service providers are also quite substantial. Firstly, the SI is now able to provide the SP with a set of application programming interfaces (APIs) for the design of services. By adhering to the APIs, now the SP does not have to worry about inconsistencies and differences in the deployment HNS. The SP now must target the service platform provided by the SI. It is the responsibility of the SI to guarantee the provision of a consistent service platform on which services can be deployed. Thus, the SP is freed from the mundane task of having to test the service with possibly hundreds of different configuration environments, simplifying the design of services targeted for the HNS.

2.2 Feature Interaction

One of the problems that the SI model is also well suited to solve is the feature interaction (from now on referenced as FI) between services. Actually, since the services now run on a unified

platform, they now compete for resources more often than before. It is critical to look into the possible feature interaction scenarios and try to classify them appropriately based on their characteristics.

Pattara in [3] gives the following definition for the feature interaction in the home network system:

> ”Feature Interaction is a type of functional conflict among multiple services not expected from the behavior of the services in isolation. FIs bring about unexpected (and often undesirable) behaviours of the service and lead the service to malfunction.”

Regarding FI, there are two major areas of research; feature interaction detection and feature interaction resolution. In the following sections we give an overview of the classification of FI, as well as present some of the traditional methods for detection and resolution of FI.

2.2.1 Type of feature interactions in the HNS

In [3], Pattara gives a brief explanation about the different types of Feature Interaction.

Device conflicts

Originally mentioned as ”Inconsistency in the execution of functions of an appliance”, this type of conflict occurs when two (or more) services send contradicting functionality requests to a device. For example, it is easy to imagine two services that independently want to set the state of an illumination device (possibly a simple light bulb). The final state of the device cannot be deterministically predicted. It could either be turned off, turned on, or the services may be able to detect that the state of the device is not the desired one, and keep flipping it on and off continuously.

Environmental conflicts

Originally referred to as ”Inconsistency in the execution of functions, which affects the environment”, this type of conflict occurs when two (or more) services send requests to appliances whose functions have contradicting effect on the same environment property. For instance, a service might turn on the heater in a room, but at the same time, another service may send a request to turn on the air conditioner which would result in a conflict, since the device effects are countering each other.

2.2.2 Traditional conflict detection techniques

Over the years different approaches for the detection of conflicts have been proposed. In this section we review some of the literature.

In [6] and [7], a model-checking approach using the SPIN model checker is proposed. Furthermore, in [4], temporal logic is used to express the correctness criteria. To produce the temporal logic formulas, a method to express the behaviour of the services in higher level logic and a classification of the causes of FI is described.

In [10] ”Use Case Maps” are used for the design and documentation of features, whereas LOTOS([9]) is used for formal verification, including the detection of undesired interactions in a PBX telephone system.

In [8] Metzger and Webel propose a series of detection methods for building control systems based on product model and the PROBAnD requirement specification method. In that paper, four different detection strategies are introduced, each used to refine a list of possible FI. Those methods are:

- Detection at requirements level. A dependency graph between the needs and the tasks is generated.
- Detection at strategy level. Dependencies between tasks and their realization methods can occur at this level.
- Detection at object structure level. Interactions that are impossible to occur are pruned in this level using information from the object hierarchy and instantiation of objects.
- Detection at environment level. This is the first notion for detection of conflicts based on characteristics of the environment. The use of a simulator that can detect implicit interactions is suggested. In research, we further elaborate on this concept introducing the notion of ”Area of Effect”.

2.2.3 Traditional conflict resolution techniques

Stemming heavily from traditional operating system concepts, in [3], six techniques of conflict resolution between services are proposed. We will briefly describe each one of them.

Inquiry to the User

When a conflict occurs, an inquiry is made to the user regarding the method of the resolution. The following are the possible methods which the user can choose from: 1) execute one service and terminate the other or 2) if possible, reset and change the parameters of the conflicting services so that FI does not occur.

Priority of Service

According to a predetermined service priority, if FI occurs, only the service with the highest priority is executed. This method is considered to be more effective in cases were multiple services exist, and possibly many users. To implement such a resolution scheme, a database for managing the order of priority of services is considered to be necessary.

Priority of User

When FI occurs, only the service activated by the user with the highest priority is executed based on the predetermined order of priority among users. This method is considered to be more effective in scenarios were multiple users exist.

Priority of Interface

When FI occurs, only the service activated from the interface of highest priority is activated. A predetermined priority order of interfaces is assumed. This resolution scheme works best when there are multiple possible user interfaces for a given service.

Priority of Timing

There are two methods in determining the service that is to be given priority based on the order of service execution, namely, to give priority to the service that is executed first or to the one that is executed last. This resolution method can be used in any classification of FI. In order to implement this method of resolution, a database for managing the order of priority of timing is required.

Meta Priority

Meta priority is the prioritization of all the above-mentioned priority levels. When a conflict occurs, it is possible to utilize more than one type of priority level in resolving the conflict. At that point, it is also likely that interactions could occur between different types of priority levels. In order to avoid such situation, it is necessary to prioritize the different types of priority levels.

2.2.4 Advantages and disadvantages in traditional methods of conflict resolution

The conflict resolution methods presented in section 2.2.3 can be said that are derived from equivalent resource allocation and scheduling techniques already found in modern operating systems. As such, they offer some clear advantages:

- Easily understood and well researched.
- Easy to implement. Already existing algorithms can be slightly modified for the needs of conflict resolution.

However, all of the above conflict resolution methods (except maybe the ”Inquiry to the User” method) have one distinct disadvantage: **the suspension of at least one service**. This is necessary to allow the remaining services to continue without any conflicts. Regarding device conflicts, it may be inevitable to completely suspend/stop a service to at least allow other services run successfully. However, it is our opinion that in cases of environmental conflicts, it may be possible to *avoid suspending a service and still have it execute with relative success*. This would thus improve the ”availability” and effectiveness of services in the HNS.

Chapter 3

Area of Effect and compromising techniques

3.1 Resolving environmental conflicts without suspending services

Traditional compromising techniques need to suspend a service in order to resolve a conflict between services. Starting from this observation, we propose a new set of conflict resolution methods that do not have this limitation. The idea is based on the observation that *a meaningful compromise may exist between the requests of the conflicting services*.

In the following sections we present the notion of "Area of Effect" of a service, and how it can be utilized for the detection and resolution of conflicts. Furthermore, we also present various compromising algorithms and their categorization. It must be said that a compromise may not be meaningful in some cases . For example, in the vast majority of device conflicts, it is almost impossible to reach a meaningful compromise. Nevertheless, it is our belief that in many cases of environmental conflicts there may exist such a compromise between the requests of the services that allows them to continue execution with a relative degree of success.

In this research we examine the cases of environmental conflicts regarding the following *four physical properties of space: illumination, temperature, humidity, sound and noise levels*. The above properties of space are those that are more often changed by the set of devices commonly available in the HNS.

3.2 Proposed system architecture

To be able to enforce a compromise between services, we took the approach of having a centralized system that has absolute control over any device that affects the four physical properties

of space. This management system knows the state of the devices at any given point in time. Furthermore, it has the task of accepting requests from services regarding the physical properties for some area inside the house. Finally, the proposed system is responsible for the detection and resolution of any environmental conflict that may occur.

In more detail, the proposed system can be broken down to three design blocks: a service API, a device control API, and the core system responsible for the resolution of conflicts. The reasons behind this three-layered approach are quite simple:

1. By providing a service API that will be used by the services, an abstraction layer is added. Using this service API, a service will be able to make a request for a resource available in the HNS. The properties of physical space (illumination, temperature, humidity, sound and noise levels) are all treated as resources. This abstraction layer frees the service designer from the daunting task of designing services that are compatible with a multitude of devices, thus simplifying the design process.

2. The design of a device control API allows uniform control over devices that exist in the HNS. Thus, it is enough that a device that wants to be part of the system implements a specific control interface.

3. By taking advantage of the above APIs, it is now possible to design the logic of a conflict detection resolution mechanism with clear separation. Thus, if the device and service APIs remain constant, it is possible to have interchangeable implementations of the conflict resolution core.

Due to the clear separation between the service requests and the actual device control, the proposed model can easily be introduced as a part of an SI service delivery platform. The implementation details of a prototype system will be explained in chapter 4.

The most important job of this centralized system is to handle the incoming requests and select proper settings for the devices in the requested area. For every request made, the system proceeds to find an *ideal solution* (that is, a series of settings for devices that fulfil the request). At this step, the system does not take into consideration any previous settings that may have been applied.

After the system has found an ideal solution, it then proceeds to detect conflicts. Conflict detection is based on two ideas presented in section 3.6

3.3 Area of Effect

In this section we will present the definition of ”Effect” and ”Area of Effect” of a device and a service. These ideas will come in handy during the explanation of the proposed algorithms developed for the detection and resolution of environmental conflicts.

3.3.1 Effect of a device at a given point in space

The four physical properties that were considered during this research are illumination, temperature, humidity, sound and noise levels.

$$Pro = \{Illumination, Temperature, Humidity, SoundNoise\} \tag{3.1}$$

Regarding a physical property Pro of space, we define the effect E of a device Dev at a given point in space p as:

$$E_{Dev,Pro}(p) \tag{3.2}$$

The above expression results in a numerical value that expresses the *intensity of the physical property* at the specific point p in space. The measurement unit of the intensity differs depending on the physical property measured. For example, $E_{Dev,Illumination}(p)$ is measured in lux, whereas $E_{Dev,SoundNoise}(p)$ is measured in db

Using physics simulation, we are able to model the behavior of various devices inside the home environment, such as air condition units, heating devices, illumination devices and others. The modelling of those devices is usually based on simple physical laws, like the inverse square law for the propagation of sound. The modelling of devices requires an implementation of function (3.2).

3.3.2 Effect of a service at a given point in space

Assuming that a service S is currently utilizing n devices (Dev_1 to Dev_n), the effect of that service in regards to property Pro at a point p in space is defined as:

$$E_{S,Pro}(p) = F\big(E_{Dev_1,Pro}(p), \ldots, E_{Dev_n,Pro}(p)\big) \tag{3.3}$$

Function F is dependent on the property that is being examined. The target of this function is to essentially compute the combined intensity of property Pro given the already known individual effects of devices $Dev_{1\ldots n}$.

For example in the case of illumination, the individual effect of devices can be summed to express the effect of a service at a given point p

$$E_{S,Illumination}(p) = \sum_{i=1}^{n} E_{Dev_i,Illumination}(p)$$

The definition of function F allows us to predict the effect of a service at a given point p. In the prototype system developed and discussed in chapter 4, physics simulation was used for the implementation of various such functions for each physical property.

3.3.3 Area of Effect of a device

For a property Pro of physical space, given upper and lower thresholds $Thresh_{up}$ and $Thresh_{low}$ the Area of Effect (AoE) of a device Dev is defined as:

$$AoE_{Dev,Pro} = \{p : Thresh_{low} \leq E_{Dev,Pro}(p) \leq Thresh_{up}\} \tag{3.4}$$

In other words, the area of effect of a device is the set of points p for which the effect of that device is between the requested thresholds. In many cases, it may be meaningful to only have one of the thresholds present and not both. In such cases, the remaining inequalities must still be fulfilled.

It is worth noting that although it is possible to get an exhaustive set of the points inside the AoE of a device, this is not usually the case. In most computations carried out by the proposed system, checks of whether a single point of interest is in the AoE of a device (or a service) is performed.

3.3.4 Area of Effect of a service

Expanding the definition of AoE of a device, the AoE of a service S is the set of points p in physical space for which the effect of the service lies between the two given thresholds $Thresh_{up}$ and $Thresh_{low}$

$$AoE_{S,Pro} = \{p : Thresh_{low} \leq E_{S,Pro}(p) \leq Thresh_{up}\} \tag{3.5}$$

For the computation of $E_{S,Pro}(p)$ the computational formula 3.3 is used. Again, as with the AoE of a device, we can perform checks against the thresholds to see if a point p in space belongs to the AoE of the service.

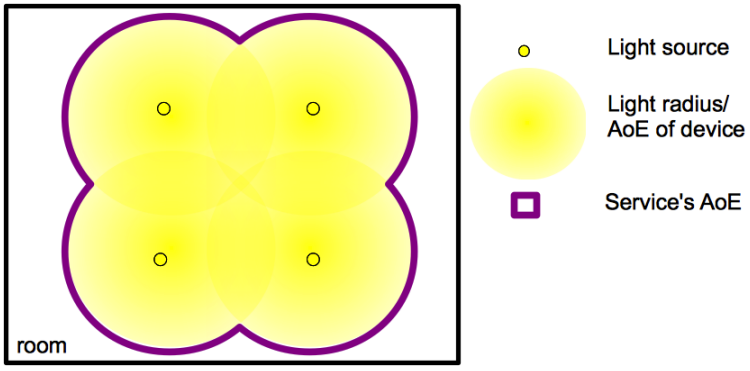

Figure 3.1: A visualization of various areas of effect.

3.3.5 Conflicting services at a point p

Given two services S_1 and S_2 and upper/lower thresholds $Thresh_{up}$ and $Thresh_{low}$ for S_1, we can perform checks to find out whether S_2 conflicts with S_1 at a specific point p in space.

$$CheckAgainst_{S_1,S_2,Pro}(p) = \begin{cases} true & \text{if } E_{S_2,Pro}(p) \leq Thresh_{low} \\ true & \text{if } E_{S_2,Pro}(p) \geq Thresh_{up} \\ false & \text{otherwise} \end{cases} \quad (3.6)$$

When the above equation returns true, service S_2 is conflicting with service S_1. However, if the result is false, this does not mean that a conflict has not occurred. A check of $S1$ against the thresholds of $S2$ must be made. If both checks return false, then there is no conflict.

To check if there is any kind of conflict between two services S_1 and S_2 we use the following formula:

$$Conf_{S_1,S_2,Pro}(p) = \begin{cases} true & \text{if } CheckAgainst_{S_1,S_2,Pro}(p) = true \\ true & \text{if } CheckAgainst_{S_2,S_1,Pro}(p) = true \\ false & \text{otherwise} \end{cases} \quad (3.7)$$

When the above equation returns true, a conflict between services S_1 and S_2 has occurred.

3.4 Request for resources

In this research, we assume that any service executing in the HNS is able to make a request for resource. The properties of physical space (illumination, temperature, humidity, sound and noise levels) are treated as such resources. Now, the need to be able to express requests regarding these resources arises.

More specifically, a service needs to be able to express the desired AoE of the service, as well as the desired intensity of the effect over that AoE. In the following sections, we describe the method we used to express the above requests in a relatively simple but effective manner.

3.4.1 Desired AoE of a service

The first important thing that a service request for resources must do is specify the desired area of effect. The desired area of effect is the physical space whose characteristics the service wants to control. For example, a service that wants control over the illumination of the space three meters around a user must be able to designate that desired area. To do so, we use the notion of *anchor point* combined with some distance specifier to describe an area. Furthermore, it is possible to designate whole rooms as the desired AoE of a service.

Almost any object inside the home environment that has a perceived position can be designated as an anchor point. For example, an anchor point can be any kind of object in a room that is

traceable (e.g. a device, a piece of furniture), an abstract point in space (e.g. the center of the room), or even a moving object (such as the user).

Combining an anchor point with a distance specifier will yield an area of space in a disc shape. This is the designated area over which the service request control. It is possible to ”expand” that area and have it to ”snap” to the size of the current room. In spite of the fact that two services may end up having equal AoEs, special care must be taken regarding the anchor point. The information regarding the anchor point (i.e. its position) plays a critical role during the later steps of conflict resolution.

3.4.2 Desired intensity of Effect

After specifying an AoE, a service must proceed and describe the actual intensity of the desired effect. Again, the service must specify the desired physical property out of the four physical properties that our system can handle. Next, the service must provide an numerical value, that describe the intensity of effect in terms of the measuring unit corresponding to the physical property selected in the previous step.

The final step to describe the desired intensity of the effect is to select between the three types of settings for the intensity: upper threshold, lower threshold and exact setting. These thresholds provide hints on how to judge the intensity of the effect.

An upper threshold states that *any intensity value below the specified intensity is acceptable*. Conversely, a lower threshold is used when a service request for *at least as much* as the specified intensity. The above two threshold definitions allow for a broader and more flexible expression of the request of a service. For example, a context-aware service that manages the environment around the user, could make requests regarding the sound and noise levels around the user when he is asleep. Such a service would request that the physical property of noise constantly remains below a specific point; such a request can be expressed with the use of upper thresholds.

Finally, the final option is to select an ”exact” setting. Using such a setting, the service can express that the intensity of the effect in the designated AoE must be as close as possible to the specified intensity. Any setting above or below that is considered to not be fulfilling the request of that service. These kind of settings make sense, for example, in the case of temperature control. With today’s technology, the user can specify a very concrete temperature with accuracy of ± 1 degrees ^oC. A service that would possibly want to control the temperature of an area can use this kind of exact setting to express its needs.

3.4.3 The importance of service requests and a centralized control system

The idea of having services making requests for resources to a centralized management system is advantageous in many ways. in this section we will describe the advantages of this method.

The first most obvious advantage of following such a design pattern is that conflict resolution in a compromising manner is now possible. Without a centralized model and the information provided by the service requests, it would be very difficult to have compromising solutions to problems of FI due to the lack of complete information regarding the situation. The proposed system, having compete knowledge of the situation, can provide a compromising solution.

The second advantage of a standardized method for resource request is that the information contained in these requests plays a crucial role during the later steps of conflict detection and resolution. For example, using the information regarding the AoE of two services, it is possible to check whether there is an area conflict between these two services. Furthermore, the compromising solution changes significantly depending on the threshold setting. For example, two services that make a request for a resource and both have a setting of upper (or lower) setting don't necessarily compete with each other, and a compromising solution is very easy in this case.

3.5 Conflicts between services

Depending on the requests made by various services, we have different types of conflicts happening in the HNS. Based on some basic characteristics, we attempt a rough classification of these types of conflicts.

3.5.1 Type of conflicts

Let us assume two services S_1 and S_2 that make a request regarding a physical property Pro and that their AoEs are at least partially overlapping. For the sake of simplicity let us examine the case of a single point p in physical space that belongs to the AoEs of the two services. Finally, the desired intensity for a service S is denoted as I_S and the possible threshold settings and the setting of a service S are the following:

$$Settings = \{UPPER, LOWER, EXACT\}$$

$$Setting_S \in Settings$$

We can now try to detect conflicts and even vaguely find a compromising solution for the intensity (I_{sol}) in the following cases:

1. $Setting_{S_1} = UPPER$ and $Setting_{S_2} = LOWER$. In this case if $I_{S_1} > I_{S_2}$ it is possible to select an intermediate value for I_{sol} such that

$$I_{S_2} < I_{sol} < I_{S_1}$$

and the requirements of both services will be fulfilled successfully as per 3.7. If $I_{S_1} < I_{S_2}$ then it is impossible to satisfy the requests of the services at the same time. Nevertheless, a solution in the space of (I_{S_2}, I_{S_1}) may still yield acceptable results.

2. $Setting_{S_1} = UPPER$ and $Setting_{S_2} = EXACT$. In this case, if $I_{S_1} \geq I_{S_2}$ then $I_{sol} = I_2$. This is because $CheckAgainstS_2, S_1, Pro$ will return false. In other words, the exact setting requested by S_2 does not conflict with the threshold setting of service S_1. In fact, it actually fulfils it. Now if $I_{S_1} < I_{S_2}$, there is no possible setting that can fulfil both requests. Again, a solution in the space of (I_{S_1}, I_{S_2}) may still yield acceptable results.

3. $Setting_{S_1} = LOWER$ and $Setting_{S_2} = EXACT$. This case is similar to the above case. With the same reasoning we deduce that if $I_{S_1} \leq I_{S_2}$ then $I_{sol} = I_2$ fulfils both services. If $I_{S_1} > I_{S_2}$ it is impossible to fulfil the conditions of both services. A compromising solution in the space of (I_{S_2}, I_{S_1}) might yield acceptable results.

4. $Setting_{S_1} = EXACT$ and $Setting_{S_2} = EXACT$. In the rare case that $I_{S_1} = I_{S_2}$ then $I_{sol} = I_{S_1}$. Otherwise, it is impossible to fulfil the demands of both services. Again, a compromising solution in the space of (I_{S_2}, I_{S_1}) might still yield acceptable results.

From the above types of conflict, it is easy to deduce that services that use the $EXACT$ threshold setting express a very strict request that may be impossible to fulfil. However, even if we are unable to fulfil the request of a service, it is possible to obtain a setting that is a compromise between the conflicting requests. This is the target of the compromising techniques presented in section 3.7.

3.5.2 N-service conflicts

As the number of services increases from 2 to $n > 2$, using the same logic described in the previous section it is possible to estimate the solution space. With logic that is similar, we can detect if it is possible to fulfil all the requests of the services and if that is not possible, find out the possible solution space.

To check whether a solution is possible for all n services at a point p, the following algorithm can be followed:

```
1  For i = 1; i< n ; i++
2    For j = i+1; j<= n ; j++
3    If (Conflict(S1,S2,p))
4      return true;
5  return false;
```

The above algorithm checks each service against all the other competing services. If there is even a single conflict, we know that there is no possibility of fulfilling all the requests of the services. The function `Conflict(S1,S2,p)` is actually the equation 3.7 presented in section 3.3.5

3.6 Detection of conflicts

In the proposed system, two main ideas are used for the detection of conflicts between services.

The first method involves the detection of conflicts on a *per-device* fashion. After the system finds an ideal solution to satisfy the request of a service, certain settings must be applied to devices. The system keeps track of all the settings that are applied to a device using a list. If the list of settings for a device is empty, the device is not being used. If the list of settings has only one entry, it means that the device is used by exactly one service.

Now, if in the setting list for a device there are more than one settings, a conflict may have occurred. The next step is to make a check to see if the requests are contradicting, and if they are, then the two services that made these requests may have contradicting settings to more than one device. The system proceeds to discover the remaining conflicting settings on other devices used by the two services, and the conflict is successfully detected.

The second method to detect conflicts in the proposed system, is to make an estimation of the AoE of surrounding devices. For example, imagine that a service makes a request for sound and noise levels with a maximum threshold (i.e. limit the amount of noise) in a room. The system would search the devices in the intended AoE of the service (i.e.) the room, and see if their effects violate the threshold. However this is not enough.

The system must make an attempt to identify if the effects of devices *outside* the intended AoE have an effect on the service. To make an estimation of the area of effect, a physics simulator can be used. If indeed the effect of a device (or a combination of effects) exceeds the threshold, then the system marks these devices and proceeds to cast appropriate settings to them. Again, the conflict detection was successful, and the information from this step will be used during the conflict resolution phase.

3.7 Compromising Techniques

In this section we present the basic ideas behind compromising techniques. The main idea behind compromising techniques is to have the system decide appropriate settings for all the devices involved in a conflict. The final settings combination should be deemed acceptable based on some predetermined criteria. Finally, a compromised solution may not honour the original requests (and their threshold settings) made by the services.

We divide the compromising algorithms into two categories, ”Space-based” resolution methods and ”Intensity-based” resolution methods.

3.7.1 Space-based conflict resolution

The target of space-based conflict resolution methods is to mitigate the conflict effect by clearly separating the AoEs of the conflicting services. To do so, a space-based conflict resolution algorithm must decide which should be the prevailing setting for a contended device.

Assuming that we have two competing services $S1$ and $S2$ and a contended device Dev , the centralized management system will first go through the process of deciding proper settings for Dev in order to fulfil the request of each service separately. Now the system is faced with the choice between two settings for Dev: the setting that would fulfil $S1$ or the setting that would fulfil $S2$. Selecting one of the two settings in essence means that Dev was assigned to one of the two competing services. If the number of conflicting devices is n the system is faced with the challenge of assigning proper devices to services. With a good assignment of the devices to services, it is possible to create areas in space where the effect of one of the two services is prevailing, thus creating a strong illusion of separation of AoEs.

One of the initial strategies is to *assign each device to the service with the closest anchor point*. The anchor point is assumed to be the point in space that the service is most interested in controlling. Considering the fact that in most cases the effect of the device is weakening as the distance between the device and the anchor point is growing (illumination and sound decaying, heat losses), the solution described above is intuitively the simplest solution that we can have: devices closer to the anchor point have greater effect, devices further away have less of an impact.

Such a simple device assignment will not really yield interesting results. Nevertheless, it is actually a good starting point. In [5] we used that initial assignment and then proceeded to improve the device assignment in an iterative manner.

The space-based algorithms can yield acceptable solutions to scenarios that involve *many devices that exhibit strong locality*. A device that has a clearly discernible (and possibly small in radius) AoE is said to have strong locality. As an example, a spot light device that is affixed to the ceiling can be said that it exhibits good locality, since the area around the device will be illuminated sufficiently, but areas that are just a few centimetres further will have a severe drop in the measured illumination intensity. Such is the case of devices that were assumed for the experiment presented in [5].

3.7.2 Intensity-based conflict resolution

Contrary to space-based conflict resolution algorithms, the target of an intensity-based algorithm is to *decide an intermediate setting* that would combine the settings of the conflicting services. Sometimes, a clear separation of the AoEs of the conflicting services might not be possible due to the limited number of devices that can control the requested physical property. For example, in the case of temperature and humidity, the AoE of a device can be substantially large, something that can hinder the separation of AoEs. Furthermore, operating more than two temperature controlling devices with different settings could potentially be a conflict, since the effects of these devices could counter each other.

For example, imagine that two conflicting services wish for different room temperatures, and the only available device in the room is an air condition unit. Let us assume that service $S1$ and service $S2$ want corresponding settings of 25^oC and 28^oC with $EXACT$ threshold settings. We soon realise that separation of AoEs in this case is not an option because there is only a single temperature controlling device. So the best possible solution is to settle for an intermediate solution for the setting of the contended device. The actual deciding parameters for the final intensity setting can defer depending on the physical property being examined, the policies enforced by the system and the implementation characteristics of the intensity-based algorithm used.

Intensity based conflict resolution methods usually work better in the following cases:

1. when the number of conflicting devices is small (possibly just a couple of devices)
2. when devices have effects that counter each other (such as in the case of temperature)
3. when the conflicting devices have poor locality (AoEs that extend in the size of a room or even expand to several adjacent rooms)
4. when the target of the compromising is to achieve a homogeneous intensity for the target property in the conflicting area.

3.7.3 Combined and Hybrid conflict resolution

A combined conflict resolution algorithm is an algorithm that combines the characteristics of space-based and intensity-based resolution algorithms. Such algorithms could potentially select to completely assign some device to a service completely or apply an intermediate setting to a device if this is deemed appropriate.

As hybrid conflict resolution mechanisms, we categorize any combination between a conflict resolution mechanism and a conventional conflict resolution mechanism, such as those presented in 2.2.3. Although we did not proceed to implement any such algorithm in this research, we

believe that such a combination is not only possible, but could potentially yield even better solutions for service conflicts.

3.8 Optimization

The target of a compromising algorithm is to decide suitable settings for all the devices that are involved in a conflict. To evaluate the appropriateness of a solution, an evaluation function is used. Using this evaluation function and by estimating the total effect of the devices on a service, we can produce a numerical score that expresses how good the given solution is. The true nature of the problem is that of an optimization problem; find the solution with the best score given specific criteria.

3.8.1 Formalization

Let us assume that n devices Dev and m services $Serv$ are involved in a conflict. For each device Dev_i there is a corresponding setting Set_i for it. Furthermore let us have:

$$D = \{Dev_1, Dev_2, \ldots, Dev_n\}$$

$$S = \{Set_1, Set_2, \ldots, Set_n\}$$

$$Services = \{Serv_1, Serv_2, \ldots, Serv_m\}$$

where D is the set of all devices involved in a conflict,S is the set of settings for those devices and $Services$ is the set of services involved in that conflict. Furthermore, let us assume an evaluation function

$$Eval(Serv_i, D, S) \tag{3.8}$$

that evaluates the total effect of devices D with settings S on service $Serv_i$. We also assume that if $Eval(Serv_i, D, S) \geq 0$ then the service request is fulfilled successfully, whereas if $Eval(Serv_i, D, S) < 0$ the request of the service is not fulfilled.

Finally let us define a function that computes the total score of the solution, i.e. the selected settings for the devices:

$$SolutionScore(Services, D, S) = G\big(Eval(Serv_1, D, S), Eval(Serv_2, D, S), \ldots, Eval(Serv_m, D, S)\big) \tag{3.9}$$

Without any loss of generality, we can say that an optimal compromising solution must maximize $SolutionScore$. Thus we need to select S_{max} such that:

$$SolutionScore(Services, D, S_{max}) \rightarrow max$$

There are a couple of points worth noting regarding the above functions. Firstly, the function $Eval(Serv_i, D, S)$ is responsible to

- compute the total effect of devices in relation to $Serv_i$ (for example, evaluate the total effect of the devices at the anchor point of the service)
- assign a score for the service by comparing the total effect of the devices to the original request made.

In essence, this function combines the measuring method used to estimate the total effect of the devices (for example, take samples from points at random and average them) as well as the scoring strategy. The scoring strategy can be different depending on the physical property discussed. For example, in the case of an illumination request, it makes sense to score the services based on a logarithmic scale, because the human perception of light changes in a logarithmic fashion [11]

Secondly, the function $SolutionScore$ can be used to express a desired scoring scheme. Depending on the characteristics of this scoring function, different policies can be implemented, thus enabling flexibility and the ability to adopt our scoring methods depending on the situation.

3.8.2 Tackling the optimization problem

Having established that the selection of appropriate settings for devices is an optimization problem, we examined several options to solve the problem. A brief commentary about each method's characteristics and an evaluation follows.

3.8.3 Linear Programming

Using linear programming and the simplex method, it is possible to find an optimal maximum to an optimization problem. However, some prerequisites must hold, namely, that of the linearity of the system.

Suppose that we have n number of variables x_i and we want to find the maximum of a target function $f(x_1, \ldots, x_n)$. Each variable x_i may have some constraint e.g. being non-negative. If the problem can be expressed as a set of linear equations and inequalities in regards to the unknown variables, then we can use linear programming techniques to find an optimal solution.

To use linear programming techniques, we have to answer whether the problem of finding an optimal compromising solution can be expressed as a linear problem. Looking back at the problem of finding an optimized compromising solution, it boils down to whether the $SolutionScore$ equation (3.9) and the service score function (3.8) can be combined to form a linear equation.

If we assume that both of the above equations are linear (for example, both equations are simple summation equations) it is possible to derive the following linear function

$$SolutionScore(Services, D, S) = c_1 Set_1 + \ldots + c_n Set_n$$

where $c_1, \ldots, c_n$ are constants and Set_i is the setting for Dev_i. Thus, the system would be expressed as a linear programming problem and solved accordingly (e.g. by using the simplex method).

Unfortunately, it is not always possible to do so. There are cases where the equations (3.8) and (3.9) are not linear equations. For example, in the case of equation (3.8), for the evaluation of sound and noise levels it is desirable to express the intensity as a logarithm of a desired setting. Moreover, the scoring function (3.8) is not always guaranteed to be linear. The total scoring strategy can change, and to express interesting scoring schemes usually a linear equation is not enough.

To avoid limiting the expressiveness of equations (3.8) and (3.9), we avoid the use of linear programming as a prospective solution to the problem.

3.8.4 SAT and MaxSAT

We briefly consider the possibility of expressing the optimization problem as SAT / Max-SAT boolean satisfiability problem.

In a typical SAT problem, we assume n boolean variables. Let us also assume that there is a function f that consists of the aforementioned variables and a combination of the operators AND, OR, and NOT. The problem is to find an assignment for variables such that the whole formula evaluates to true.

There are several special cases of the SAT problem (e.g. formulas are required to be in a conjunctive normal form, 3SAT, others), but one with many interesting properties is Max-SAT. Max-SAT is the problem of determining the maximum number of clauses of a given boolean formula that can be satisfied by some assignment.

Taking into consideration the fact that the intensity settings of devices may span a continuous space, it becomes difficult to express the problem of finding a compromising solution as a SAT/ Max-SAT problem. Depending on the number of devices and their possible settings, a SAT encoding of the problem may lead to unrealistically high execution times. Furthermore, the complexity of the evaluation functions makes matters even worse, not allowing an easy transition from the problem as defined in 3.8.1 to be expressed as SAT/ Max-SAT. Thus, we believe that such an approach is not realistically possible.

3.8.5 Game Theory

Citing [12] ”game theory is the formal study of conflict and cooperation. Game theoretic concepts apply whenever the actions of several agents are interdependent. These agents may be individuals, groups firms, or any combination of these. The concepts of game theory provide a language to formulate, structure, analyze and understand strategic scenarios.”

A *game* is the formal model of an interactive situation. In non-cooperative game theory, *Players* play the game in order to maximize their personal gain (a rational gamer). Furthermore, there is cooperative game theory where players may form a coalition to dominate the game. In a game, each player follows a strategy and it is also assumed that the player is rational (i.e. the player strives for maximum personal benefit).

There are two forms in which games can be expressed: *a strategic form* (also called a *normal form*) and an *extensive form* (also known as *game tree*). The strategic form of a game lists the strategies of the players and also the outcome of each possible combination of them. The outcome is presented as a separate payoff for each player.

The extensive form of a game is more detailed than the strategic form, and holds complete information on how the game evolved over time. This includes the order in which players take actions, the information the players have at each major point in time, and the times at which any uncertainty in the situation is resolved. A game expressed in extensive form can be converted to a game in strategic form.

A final concept in game theory that is interesting and potentially useful in a compromising solution, is the concept of Nash equilibrium A Nash equilibrium recommends a strategy to each player that the player cannot improve upon unilaterally, i.e. the other players behave rationally and follow the recommended strategies.

I \ II	Co-op	Deflect
Co-op	2 2	3 0
Deflect	0 3	1 1

Figure 3.2: The prisoner’s dilemma

Having explained the above notions, let us have a quick look at the *prisoner’s dilemma*. The strategic form can be seen in figure 3.2. In this game, each player has two strategies: cooperate or deflect. The payoffs for player I is the number in the lower left corner of each combination whereas the payoffs for player II are on the upper right corner.

In this game, strategy ”Deflect” dominates over the strategy of ”Co-op”. This means that, regardless of the choice of the opponent, choosing ”Deflect” is always going to bring more personal gain to the players. Since both players are rational, they will not choose the dominated strategy ”Co-op”. The Nash equilibrium for this game is for both players to select the ”Deflect” strategy, although this will paradoxically yield a lower gain for both players compared to a scenario of mutual cooperation.

The enticing features of game theory (the notion of players , strategies and Nash equilibrium) may make game theory a candidate method to solve the problem of optimization of a compromising solution. However, there are some severe disadvantages that do not enable us to express this optimization problem as a game.

- Unrealistically big strategic forms. As soon as the number of possible settings, devices and ”players” (in this case possibly services in the HNS) increases, the strategic form is going to have an exponential growth. The cost of just creating such a table is equal to an exhaustive search in the solution space which could prove to be unrealistic.
- The extensive form does not have any tangible advantage over other techniques, such as local search, branch and bound and others. The game tree could very well be expressed by a minimax or alfa-beta search algorithm.
- Players are rational and want to maximize their personal gain. This leads to having players making choices that instead of alleviating conflicts, introduce even more conflicts during the conflict resolution phase.

3.8.6 Constraint programming

In [13], a thorough review of the literature regarding the field of constraint programming can be found. Constraint programming deals with the problem of constraint satisfaction; assigning a value to every variable of a set of problem variables so that all the constraints are satisfied. The constraints are used to specify limitations on the value domain of a problem variable and relationships between problem variables.

Citing [13], page 17, a typical definition of a Constraint Satisfaction Problem (CSP) is the following. A CSP P is a triple $P = \langle X, D, C\rangle$ where X is an n-tuple of variables $X = \langle x_1, x_2, \ldots, x_n\rangle$, D is a corresponding n-tuple of domains $D = \langle D_1, D_2, \ldots, D_n\rangle$ such that $x_i \in D_i$, C is a t-tuple of constraints $C = \langle C_1, C_2, \ldots, C_t\rangle$. A constraint C_j is a pair $\langle R_{S_j}, S_j\rangle$ where R_{S_j} is a relation on the variables in $S_i = scope(Ci)$. In other words, R_i is a subset of the Cartesian product of the domains of the variables in S_i.

A solution to the CSP P is an n-tuple $A = \langle a_1, a_2, \ldots, a_n\rangle$ where $a_i \in D_i$ and each C_j is satisfied in that R_{S_j}, holds on the projection of A onto the scope S_j. In a given task one may

be required to find the set of all solutions. $sol(P)$, to determine if tat set is non-empty or just to find any solution, if one exists. If the set of solutions is empty the CSP is unsatisfiable.

When a CSP is unsatisfiable it is said to be over-constrained. Sometimes, we may want to express some preferences that may not necessarily be fulfilled. Other times it may be impossible to fulfil all of the supplied constraints but we still want to fulfil the maximum amount of constraints as possible. Soft constraints and weighted CSP, an extension to classic CSP allows us to express the above needs.

A *weighted constraint network* (weighted CN) is a triple $\langle X, D, C \rangle$, where X and D are the set of variables and their domains as defined in classical CNs, and C is a set of weighted constraints. A *weighted constraint* $\langle c, w \rangle$ is just a classical constraint c, plus a weight w (over natural, integer, or real numbers).

The cost of an assignment t is the sum of all $w(c)$, for all constraints c which are violated by t. An *optimal solution* is a complete assignment t with minimal cost.

The above properties of weighted CSP are very well fitting to express the problem of finding an optimized compromising solution. Firstly, X and D are well defined, and can be used to express the number of devices and their possible settings accurately. Secondly, we can easily express the requests of services as soft constraints Thirdly, minimizing the total cost of an assignment is in a sense a compromise; the decision to minimize the cost might involve having a bigger number of unsatisfied requests but with minimum ”badness” i.e. the final assignment of settings makes a compromising between the requests.

Finally, for the solving of CSPs many different approaches have been made. Constraint propagation, backtracking search algorithms, randomization and random restarts, no-good recording, randomized iterative improvement algorithms, local search algorithms, dynamic programming and other global optimization techniques have all been used to tackle cases of CSPs with different characteristics and various success rates. In the algorithms described in section 4.6 we were heavily influenced by the ideas of weighted CP.

Chapter 4

System Architecture and Implementation

In this chapter we explain the implementation details of the prototype system. This system is based on the characteristics described in 3. As a development platform, we selected the Java SE 1.6 development platform. The total code for the prototype system amounts to about 6500 lines of java code. Furthermore, initial integration with the OSGi platform has begun.

4.1 Architecture Overview

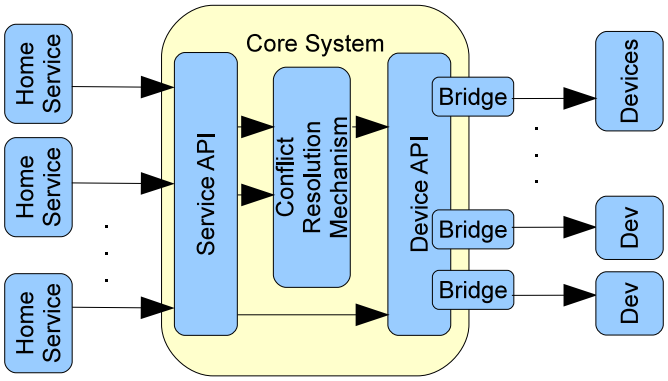

Figure 4.1: System Overview

As we can see in figure 4.1, the prototype is split in three main design blocks: the device API, the service API and the core conflict resolution system.

The device API is the API used internally to control a number of devices inside the HNS. To add a device to the proposed system, it is enough to have the target device implement one of the device interfaces already designed. The service API will be used by the services to make request for resources. Finally, the conflict resolution mechanism implements the conflict detection and

conflict resolution mechanisms. Although both the device API and the service API still need further refinement, after a stable API is reached, it will be possible to have interchangeable conflict resolution mechanisms.

4.1.1 Class diagrams and important concepts

To represent the details of the home environment, the class hierarchy presented in figure 4.2 was used. In this figure we can see the main classes that are used to handle information regarding a scene. As a scene, we refer to the details of the HNS; the position of devices, the sizes of the rooms, the connectivity between rooms and the characteristics of of wall are all being handled using the classes presented in this figure.

More specifically, the class `AbstractSceneObject` is the first non-abstract class that also acts as a parent class for many other important classes. The `AbstractSceneObject` class provides a default implementation the `Nameable`, `Countable`, `HasSize`, `Positionable` interfaces, subsequently used by other subclasses. The above interfaces allow us to change the name, size, position and also assign a unique serial number to every object in the scene. Furthermore, the `AbstractSceneObject` offers the ability to clone the information of another `AbstractSceneObject` object. Finally, each such object can hold a reference to its ”parent”. For example, when a device is registered in a room, the room is responsible of registering that device to its list of devices, but also makes sure that the room is registered as the parent of this device.

Deriving from the `AbstractSceneObject` are the `Room`, `Conduit`, `Wall` and `AbstractDevice` classes. These classes represent information regarding the rooms of the house, the connectivity between the rooms, the walls of the house and the devices inside the home environment respectively.

Each `Room` object has a list of devices inside that room, plus a list of the walls of the room. At the current stage, each room is considered to be box-shaped, and always has six walls. Furthermore, a conduit that connects two rooms is always associated to a wall. A conduit also has a ”behaviour”; the behaviour gives us information about how the two rooms are connected. The rooms could be connected using doors, windows or even nothing at all (a part of the wall is missing). Finally, walls are represented as geometrical planes, and are always assigned to a room. During start up, the walls are computed automatically from the sizes and positions of the rooms and any aliasing (a same wall being computed twice) are resolved.

4.1.2 Implementation of the Core System

In figure 4.3 we can see the implementation and the class dependencies of the core system. First of all, the system implements the `Runnable` interface. In essence, the core system runs

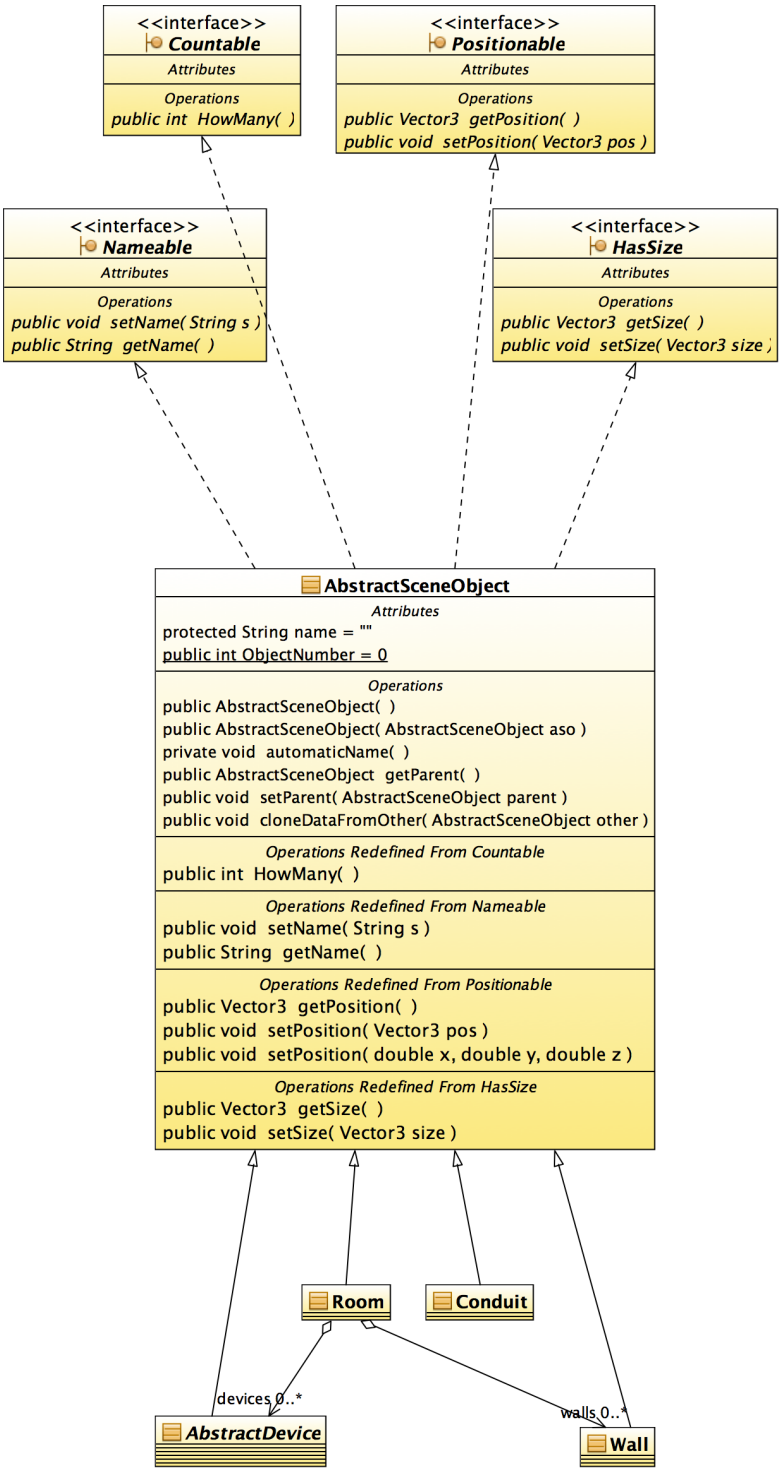

Figure 4.2: Basic objects in the system

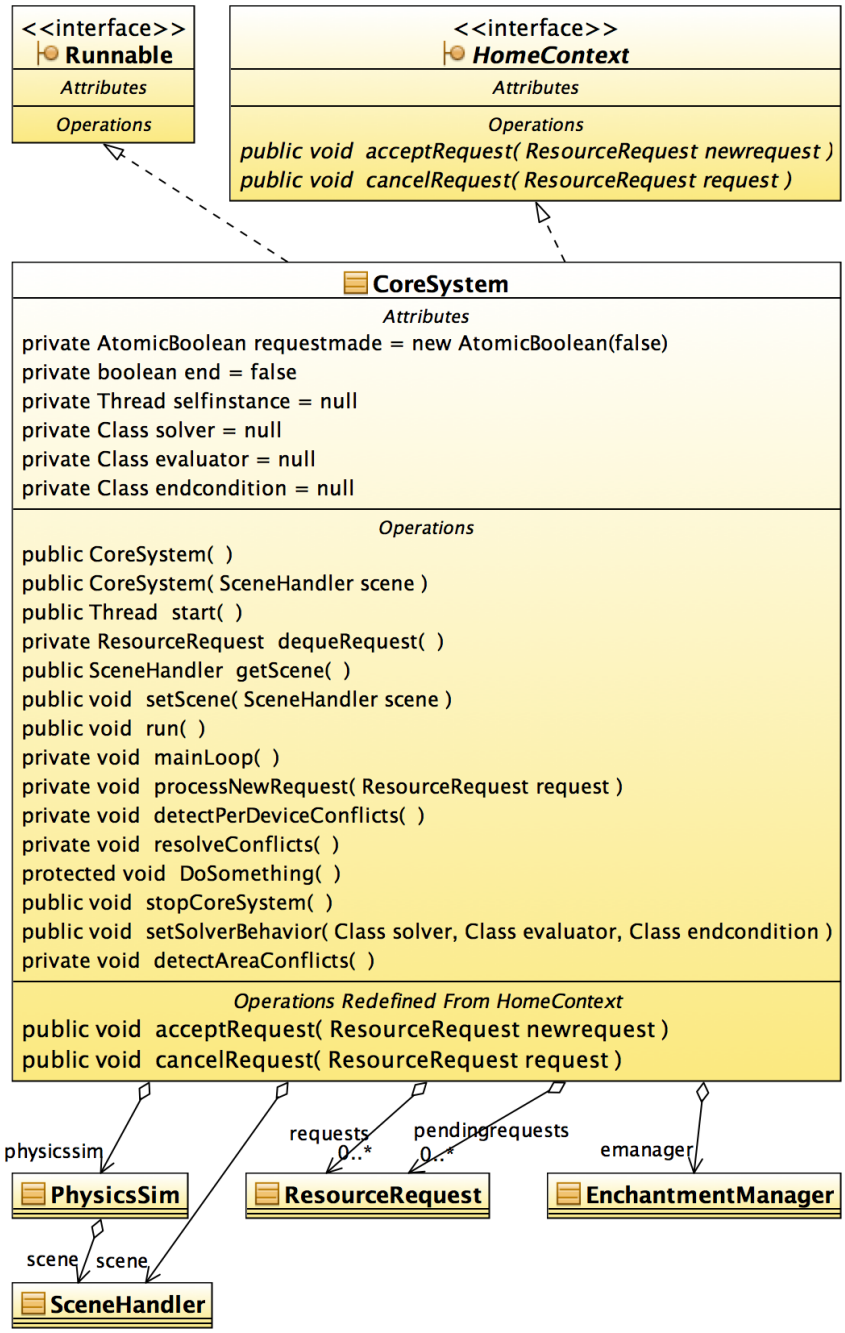

Figure 4.3: Core System Implementation

in its own thread. The call to function `start()` is used to start the system, and the call to `stopCoreSystem` is used to bring the core system to a halt.

Secondly the core system implements the `HomeContext` interface. This is the interface which the services use to interact with the core system. For the time being, the implementation is pretty minimal and it consists of only two functions: `acceptRequest()` and `cancelRequest()`. Using this interface, the services can submit or cancel already submitted resource requests.

Thirdly, the core system internally employs the following components:

- Two request lists, one for pending requests (requests that have yet to be processed) and already processed service requests.
- A physics simulator. The physics simulator is responsible for the simulation of physics. Its interface is rudimentary and the final goal is to design programming APIs so that the physics simulator can be interchangeable. Finally, the physics simulator has access to the scene handler.
- A ”scene” handler. This particular class manages all the spatial information about scene objects (rooms, devices, others). It provides search-by-name and search-by-position functionality as well as some basic path finding.
- An ”enchantment” manager. This is a manager that keeps track of the what we named as ”enchantments”. Enchantments are settings for devices that the system decides in various stages. The notion of enchantments will be explained in section 4.4.2.

The configuration of the home environment is stored in `xml` files such as these seen in appendix chapter A. For the time being, the configuration is static. However, it is possible to add devices or modify the properties of already existing devices during runtime.

Finally, the core system has the following attributes: `Class solver`, `Class endcondition` and `Class evaluator`. These three properties are initialized at start up, and control which solver, which end condition and which evaluation function will be employed as the compromising algorithm. Currently, there is no dynamic selection of algorithm available, but it is considered as a later addition to the system.

4.2 Devices and Device API

In this section, we first present how the devices are seen in relation to the `AbstractDevice` superclass. In figure 4.4 we can see that all concrete device classes in our system so far are derived from the `AbstractDevice` class.

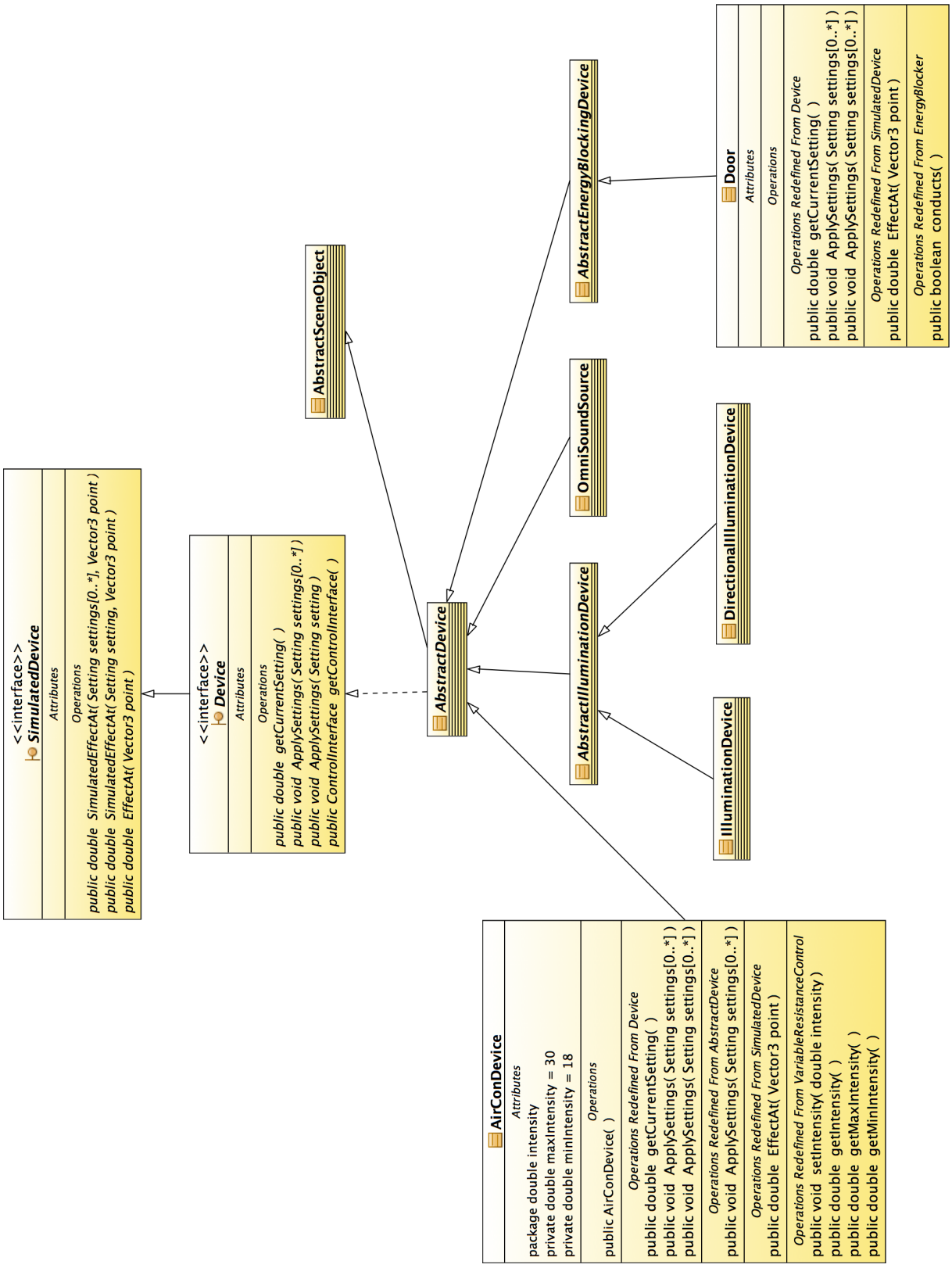

Figure 4.4: Device hierarchy

The abstract class `AbstractDevice` implements the interfaces `SimulatedDevice` and `Device`. The `SimulatedDevice` interface defines functions that allows us to model the effects the device has on the surrounding environment. the function with name `EffectAt(p)` can make an estimation of the effect the device has at point p. The function `SimulatedEffectAt(settings, p)` makes an estimation of the effect of the device if *settings* are applied to that device. `AbstractDevice` also provides a default implementation of the function `SimulatedEffectsAt()` so there is no need to override this function.

Since the effect of a device and the settings management varies wildly from device to device, `AbstractDevice` as an abstract class does not implement function `EffectAt` or any of the functions in interface `Device` so these have to be implemented by the classes derived from it.

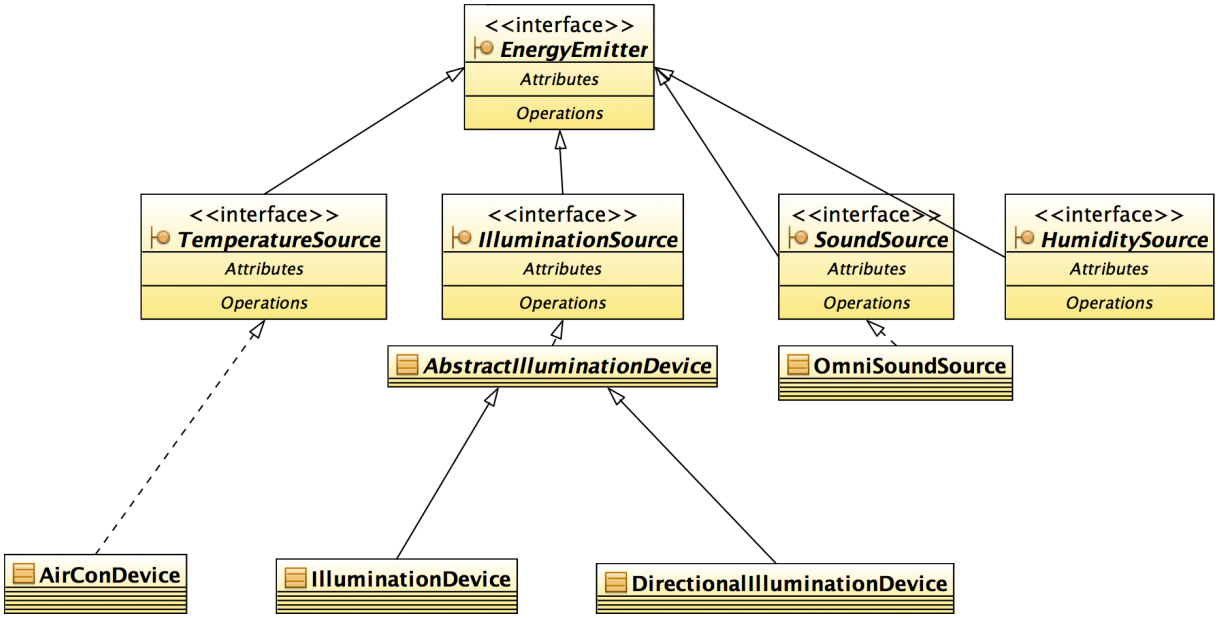

Figure 4.5: Device categorization: Energy Emitters

Now we will proceed to look the devices from a different angle, roughly categorizing them depending on the effect that they have on the environment. In 4.5 we see that any energy emitting design is indirectly implementing the `EnergyEmitter` interface, and directly implementing one of the sub-interfaces of that. There are four sub-interfaces, one for each physical temperature (illumination source, temperature source, humidity source, sound source). These interfaces are used for the categorization of devices, and do not actually include any method declarations.

Furthermore, the conduits are considered to be energy blockers. The `Conduit` class is not an energy blocker by default, but its ”behaviour” component is. That component is usually of the type `AbstractEnergyBlockingDevice`. So far we have only implemented a `Door` class that simulates the usage of a door as a conduit between two rooms.

Now, we will have one last look on devices and their categorization. This time, the focus is on what control interface they implement. In figure 4.6 we can see the proposed interfaces as well as which of these interfaces are implemented by the devices developed so far.

The basic `OnOffControl` interface is used to simulate devices that behave like switches. The devices that implement this interfaces are guaranteed to have at least the two basic states: on and off. Using this interface, we can set the device to any of these two states. The abstract class `AbstractDevice` also implements this interface, guaranteeing that all devices inheriting from it will behave like a switch. More importantly, this interface provides us with a convenient functions to turn on and off a certain device. Example of devices that implement the `OnOffControl` interface include the `Door` and the `IlluminationDevice` classes of devices.

Furthermore, we designed the `FourWayContro` and `VariableResistanceControl` to simulate the controls of other devices. For example, the four-way control can be used to control any device that usually provides the user with a ”low-medium-high” setting, whereas the variable resistance control will be used by devices that provide the user with a potentiometer for the control of the intensity. The variable resistance interface comes handy when we need to control, for example, the volume of a Hi-Fi system or the intensity of lights in the room.

Our implementation decisions led to the design seen in figure 4.6. An air condition unit and an omni-directional sound device are using the variable resistance interface, whereas most other devices are using a typicial on/off control, derived from the `AbstractDevice` class.

4.3 Service API

As seen in section 4.1.2, the core system implements the interface `HomeContext`. During the initialization of a service, the service gets a reference to a `HomeContext` object. Using this object, a service is able to make requests for resources and also cancel previously issued requests.

The `HomeContext` interface is exposed over OSGi (more on that on 4.7.1). Since the system is a prototype, the functionality implemented is the bare minimum. Over time, the `HomeContext` interface will provide facilities for services to make queries about information regarding the execution environment, such as for example, who invoked the service, how was the service invoked and what is the service’s anchor point.

`ResourceRequest` object

To make a request, a service must create an object of class `ResourceRequest`, seen in figure 4.8

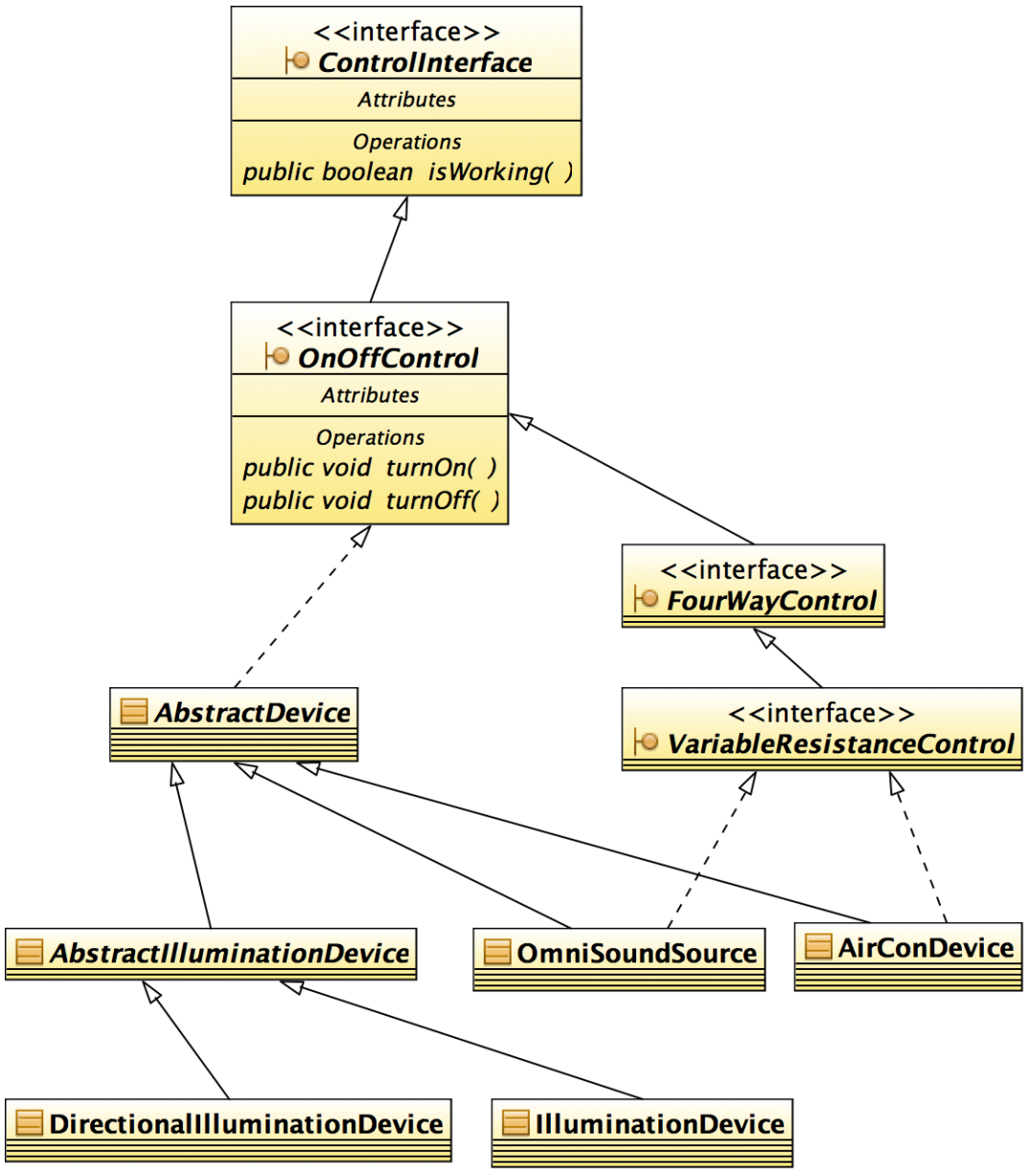

Figure 4.6: Device categorization: Control Interfaces

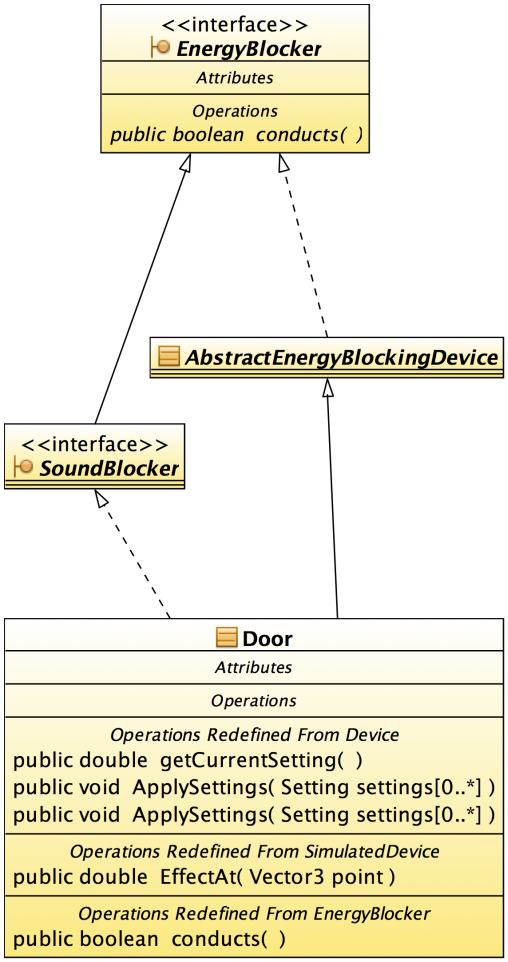

Figure 4.7: Device categorization: Energy Blockers

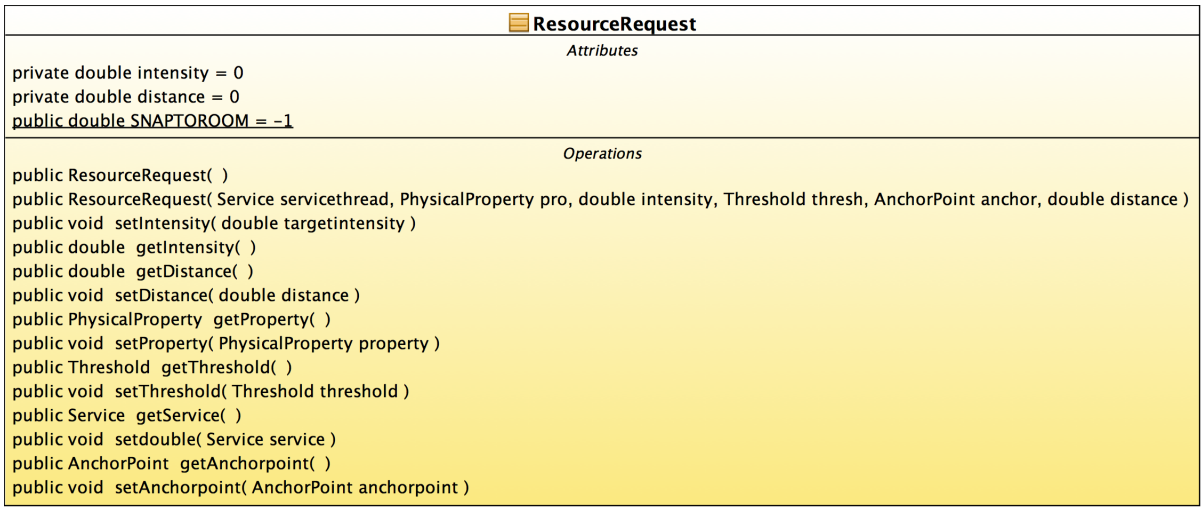

Figure 4.8: Resource Request

This object provides setter and getter methods for access to its internal information. The information represented by an object is the information described in section 3.4, that is information about the intensity of effect and the intended area of effect.

More precisely, each `ResourceRequest` object holds the following information:

- the physical property to be controlled
- the intended intensity of the physical property
- the anchor point of the service
- the distance around the anchor point that the service wants to control
- the threshold type regarding the intensity
- a reference to the service that made the request.

The combined information regarding the anchor point and the distance, expresses the area of effect. If distance is set to -1, then the area of effect automatically ”snaps to” the limits of the current room, i.e. the whole room is considered to be the target of this request. The information regarding the intensity, the physical property and the type of threshold dictate the desired intensity of the effect. Finally, each `ResourceRequest` object holds a reference to the service that actually made the request. This is used internally by the system.

4.4 Detection and Resolution of conflicts

4.4.1 Main loop of the system

The prototype system implements the ideas expressed in sections 3.2, 3.6 and 3.7 for the handling of requests, the detection and resolution of conflicts accordingly. We start by commenting on the main loop of the core system, as seen in figure 4.9.

```
private void mainLoop() {
      //First phase: handle new requests

      //three steps:
      //1. deque request. as loong as we have requests, keep deqeuing
      ResourceRequest rr;
      while ((rr = this.dequeRequest()) != null) {
         //2. process the request
         processNewRequest(rr);

         //3. add it to the processed requests.
         this.requests.add(rr);
      }

      //Second phase: Detect conflicts
      detectPerDeviceConflicts();
      detectAreaConflicts();
      resolveConflicts();

   }
```

Figure 4.9: The main loop of the core system

As it can be seen from the source code, upon the receipt of a resource request, the system proceeds to do the following things:

1. process the new request. Processing a new request is handled by the function `processNewRequest`. To process a request, an *ideal solution* is found. Next, the request is added to the proper list of processed requests.

2. The system attempts a per-device detection of conflicts and after that, detection of area

conflicts is performed. The basic idea in the detection of conflicts is described in section 3.6.

3. Attempt conflict resolution. The methods implemented in the prototype system will be explained in section 4.5.

4.4.2 Enchantments, Enchantment Manager and ideal solutions

Before we proceed any further, we must explain how the prototype system manages the settings of devices.

Enchantments

The building block of information is the `Enchantment` class. To manage enchantments we also use an enchantment manager, implemented by the class `EnchantmentManager`. *An enchantment holds information regarding the settings to be applied*. The `Enchantment` object hold a reference to a `Setting` object, a reference to the device that the settings will be applied to, and finally a reference to the `ResourceRequest` that caused this enchantment (figure 4.4.2).

An `EnchantmentEntry` is used to keep track of the enchantments cast to a device. For each device we have *exactly one* enchantment entry that holds the following information:

- a reference to the device whose list of enchantments are taken care of in this enchantment entry
- a list of the enchantments cast on this device. Proper API for the addition and removal of enchantments has been provided.
- a reference towards the current enchantment. The ”current” enchantment is the enchantment whose settings are currently applied to the device.

We must mention the auxiliary class `Setting` that is used to hold a variety of settings. Depending on the implemented interfaces of a device, the fields of the `Setting` class may have different meaning. Usually, a double is used to convey a specified intensity whenever possible, a boolean variable to set the device on or off, and an enumeration when the intensity can only be controlled in steps of `LOW, MEDIUM, HIGH`.

In figure 4.10 we can see a class diagram of the enchantment related classes and in figure 4.11 we can see a block diagram.

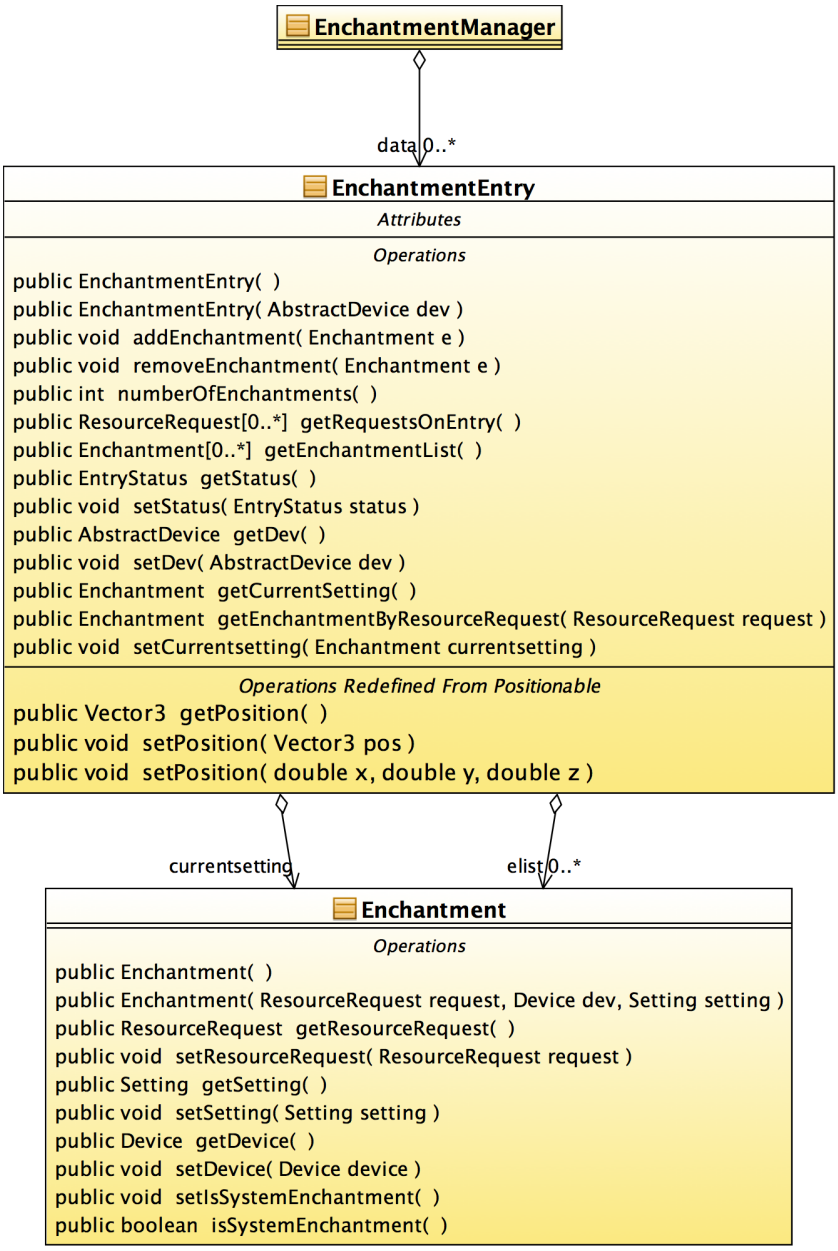

Figure 4.10: Enchantment, EnchantmentEntry and EnchantmentManager

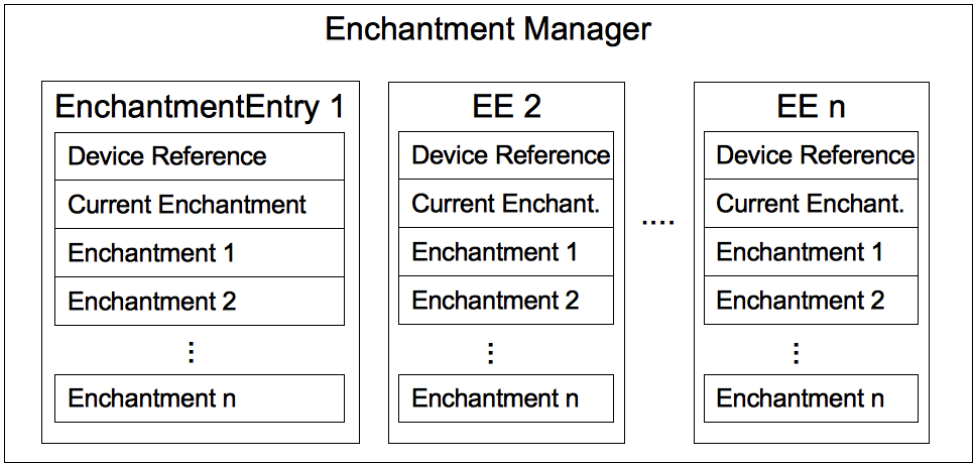

Figure 4.11: A block diagram of the enchantment manager.

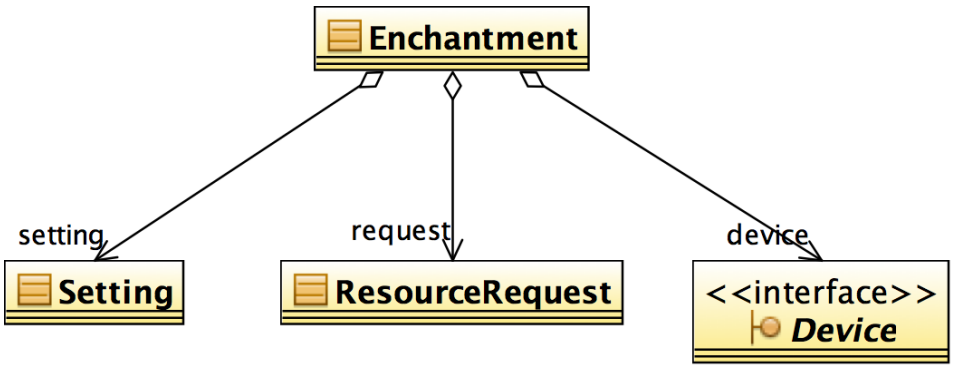

Figure 4.12: Enchantmnent, Setting, ResourceRequest

Ideal solution

One of the first steps that the core system does when it receives a resource request, is to generate an ideal solution for it. When the system tries to decide an idea solution for a request it intentionally ignores the enchantments cast to devices due to previous request. This is a deliberate course of action, as the enchantments cast in this stage provide crucial information for the detection and resolution of conflicts in later steps.

To generate an ideal solution we use the following, somewhat simplistic approach:

1. Check the physical property of the original `ResourceRequest` object.
2. Get a list of the devices inside the intended AoE of the service that can affect the physical properties of step 1. To calculate the intended AoE, the anchor point and distance specifier information of the resource request object is used.
3. Check the type of threshold. Depending on the type of threshold, take appropriate actions.
   (a) If the threshold type is `UPPER` then the service has asked that an upper limit must be imposed on a property. To fulfil this request, we impose the lowest setting to all the devices obtained in step 2. In many cases, the selected setting is to turn off these devices.
   (b) If the threshold type is `LOWER`, the service has requested for a lower limit to be imposed on a property. In the same spirit as the `UPPER` case, we set the settings of affected devices to the conceptional maximum possible setting. Depending on the interface of the device, we can just turn on a switch device, set intensity to `HIGH` for a `FourWayControl` device, or just set the maximum possible intensity for a device that implements the `variableResistanceControl` interface.
   (c) If the threshold type is `EXACT`, then the system proceeds to find a solution as close as possible to the exact requested intensity.
4. Create the appropriate enchantments and add them to the corresponding enchantment entries for the devices.

The above ideas are implemented in the ideal solvers of this system. However, we must mention that for request with `EXACT` threshold types, the problem of finding an ideal solution may turn into *a discrete knapsack problem*. For example, in a scenario where a specific illumination intensity is requested, and all the devices only support a switch interface, it is not easy to guarantee that a good solution will be found. Thus, in the current prototype system an ideal solution is provided only for temperature.

4.5 Conflict detection algorithms

As discussed earlier, two techniques for the detection of conflicts were implemented in the prototype system: *per-device conflict detection* and *area conflict detection*.

4.5.1 Per-device conflict detection

After the initial processing of a request, the core system proceeds to detect conflicts on a per-device basis. After the initial ideal solution has been decided, a set of new enchantments is cast upon the appropriate devices. Then, the corresponding `EnchantmentEntry` objects are marked appropriately to be checked for conflicts. An `EnchantmentEntry` object is mark `UNUSED` when it has no enchantments on it, `NORMAL` when it has only one enchantment on it, and `CONFLICTING` if there are more than two enchantments that are conflicting. For two enchantments to conflict, one must be of threshold type `UPPER` and one of type `LOWER`, or at least one of them being of type `EXACT`.

Before a conflict resolution algorithm is applied, the system generates a list of conflicting services and a list of conflicting devices used by those services. The lists will be passed as arguments to the conflict resolution algorithm. To generate the lists, the system uses the following algorithm:

1. Initialize conflicting requests list `conf_r.empty == true`
2. Initialize conflicting enchantment entry list `conf_ee.empty == true`
3. Until all `EnchantmentEntry.status != CONFLICTING` do:
4. For enchantment entry `e`, If (`e.status == CONFLICTING`)
   (a) Add current enchantment `e` to `conf_ee` list
   (b) Add all resource requests on `e` to `conf_r`
   (c) For all other `EnchantmentEntry ee` do:
   (d) If `ee` contains enchantments from requests already in `conf_r` do:
      i. add all resource requests that have enchantments on `ee` to the `conf_r` list.
      ii. add `ee` to `conf_ee`
   (e) `Solve(conf_ee, conf_rr)`
   (f) for each `Enchantment entry ee` in `conf_ee`, set `ee.status == RESOLVED`
   (g) Clear `conf_r` and `conf_ee`

By the end of each iteration, the `EnchantmentEntry` list `confl_ee` and the list of conflicting `ResourceRequest` references `conf_r` are created. These are fed into the compromising algorithm to get a solution.

4.5.2 Conflict detection using AoE information

In this detection scheme, checks are made to discover whether the effect of a device outside the specified in the request AoE has an effect on the service. For example, a `SoundSource` device could potentially raise the sound/noise levels in an adjacent room.

To check if such devices exist, we estimate the effect of such potential devices at the anchor point of the service. If the effect exceeds a specified threshold then an enchantment is cast on that device on behalf of the request. The enchantment cast depends on the threshold type of the resource request and it is considered as part of the ideal solution of the previous step. The conflict resolution step will take measures to decide an appropriate setting for that device.

Finally, to estimate the effect of such devices, a simple *physics simulator* is used. For the time being, only illumination and sound/noise are supported for area detection.

4.6 Conflict resolution algorithms

In this section we present the main ideas and concepts used for conflict resolution using compromising techniques as well as explain the details of the algorithms implemented for the prototype system.

The behaviour of the compromising algorithm is dictated by three things:

- The *end condition*. The end condition evaluates whether the current solution is better than that of the previous round and provides a hint as to when the solving algorithm should end or not. This decision is made based on the data of the current and previous round of changes in the solution.

- The *service evaluation function*. The service evaluation function (also known as evaluators) embodies two things: a) the intensity measuring method and b) an evaluation strategy of the estimated intensity against the one requested. The evaluation function will defer depending on many different factors, such as the physical property examined, or the desired accuracy of measurement.

- The actual *algorithm implementation*. The implementation controls which part of the solution space will be examined. Each algorithm may employ different techniques to achieve vastly different results. The algorithms developed are variations of local search algorithms.

The system is designed in such a fashion that any combination between the above three components is possible (even though in many cases some combinations might not make sense). We proceed to explain each and every one of the above implemented features of the prototype system.

4.6.1 End conditions

Two end conditions where implemented in this system. The first one is known as the ”highest score” end condition and the second one as the ”constrained programming” end condition. Both conditions evaluate the total score of an iteration and compare the result to the previous iteration. If the result of the current iteration are worse than the previous iteration, the end condition will trigger, and the algorithm can choose to either a) terminate the algorithm (hence the term end condition) or b) to ”undo” the last change in the solution and try another possible combination. The decision to end the search for a solution is left to the algorithm itself.

Highest Score end condition

This end condition (which is implemented by the class named `HighestScore`) computes the total sum of the scores of the services. If the current sum is greater than of the previous round, the algorithm should continue the search for a solution, since the last change in the solution set was favourable. Conversely, when a change is not favourable, the algorithm may opt to ”undo” the last change (to get the best solution so far) and return that solution.

The equation (3.9) now becomes:

$$SolutionScore(Services, D, S) = \sum_{i=1}^{n} Eval(Serv_i, D, S)$$

Constrained Programming end condition

The idea for this end condition was directly derived from the weighted constrained programming theories presented in 3.8.6. In typical weighted constrained programming, the total score of the assignment is the weight summation of the violated constraints. In the same spirit, this end condition only sums the negative scores of services whose requests are not fulfilled.

The equation (3.9) now becomes:

$$SolutionScore(Services, D, S) = \sum_{i=1}^{n} w \cdot Eval(Serv_i, D, S)$$

$$\text{where } w = \begin{cases} 1, & if \quad Eval(Serv_i, D, S) < 0 \\ 0, & if \quad Eval(Serv_i, D, S) \geq 0 \end{cases}$$

A point worth mentioning here is that, in contrast to traditional weighted constraint programming where the weights of the constrains are predetermined, the weight of a violated constrain is computed using the equation $Eval(Serv_i, D, S)$ (more on that in the next section).

4.6.2 Evaluators

To evaluate how well a service request is satisfied, we use what we named as *evaluators*. An evaluator is responsible to evaluate how well a service's request is fulfilled. A positive score indicates that a request is fulfilled (the greater the number the better the service is fulfilled), whereas negative numbers indicate that the request is not being fulfilled. Evaluators are the implementation of the equation (3.8) presented in earlier sections.

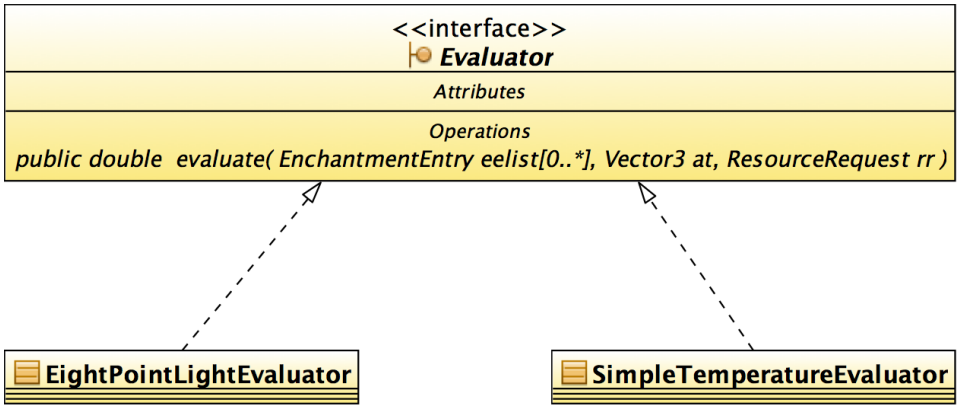

Figure 4.13: Evaluators

An `Evaluator` object implements a very simple interface, as seen in figure 4.13. To evaluate the satisfaction degree of a service, we must provide the three following pieces of information:

1. a list of all `EnchantmentEntry` objects present in the conflict. Each `EnchantmentEntry` reference holds information about the current setting of the device, a reference to said device as well as the list of enchantments cast upon it.

2. an optional argument of type `Vector3` to provide a hint to the system where we want the epicentre of evaluation. In case this argument is `null` the information from the third argument is used.

3. a reference to the actual `ResourceRequest` object. It holds valuable information as to what the service has requested. Furthermore, it contains information regarding the anchor point of the service, that can be used in cases where the second argument is `null`.

EightPointLightEvaluator

One of the implemented evaluators is the `EightPointLightEvaluator`. What this evaluator does is to take eight illumination measurements around the point of interest at r distance (usually 1 meter). The places of measurement can be seen in figure 4.14. For each of the eight points, the total estimated intensity will be calculated and then averaged. This can provide a good idea of what the illumination at the center of that circle feels like. This evaluator is used only in the case where the requested property was illumination.

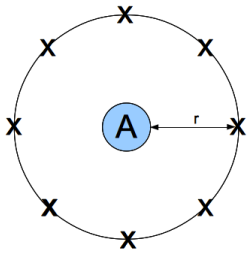

Figure 4.14: Eight measuring points around anchor point A.

To actually calculate a final satisfaction score for the service, the evaluator uses the following formula:

$$score = \begin{cases} 10 \log_{10}\left(\frac{I_{Est}}{I_{Req}}\right) & \text{, if threshold is met} \\ -10 \log_{10}\left(\frac{I_{Est}}{I_{Req}}\right) & \text{, if threshold is not met} \end{cases} \tag{4.1}$$

where I_{Est} is the total effect estimated using physics simulation and I_{Req} is specified by the resource request. As we can see the evaluation function employs a logarithmic scale. The reason we followed this strategy is because of the strong indications that the human perception (and by extent the perception of illumination) behaves in a logarithmic fashion. This behaviour is know as the ”Weber-Fechner Law” [11].

If the threshold requested was not met, the score is negative. For requests with `EXACT` threshold types, the score is always negative or (in really rare cases) zero.

SimpleTemperatureEvaluator

To evaluate the satisfaction degree regarding a request about temperature, we take a measurement at the point of interest, and then evaluate the score for the request using this simple formula:

$$score = \begin{cases} a(I_{Est} - I_{Req})^2 & \text{, if threshold is met} \\ -a(I_{Est} - I_{Req})^2 & \text{, if threshold not is met} \end{cases} \tag{4.2}$$

Again, I_{Est} is the total effect estimated using physics simulation and I_{Req} is specified by the resource request. The function used for evaluation ”punishes” (or rewards) in an exponential fashion and the coefficient a is used to control the steepness. Using such a formula, and temperature difference more than a couple of degrees will start to incur heavy penalties.

4.6.3 Compromising Algorithms

In this section we present the algorithms that were implemented as part of the prototype system.

Space-based algorithms

Simple space-based algorithm During the early prototyping of this system, this simplistic algorithm (which is presented in [5]), was the first attempt to a compromising conflict resolution algorithm. Although simple, many ideas came out of it that were later implemented in other algorithms, such as the following:

- *An initial device assignment to services*. The first job of the algorithm is to assign the contended devices to the services with the closest anchor point to them. This is because usually devices that are closer to an anchor point have a stronger effect at the area near the anchor than devices further away.
- *An iterative process*. In each step the solution would be improved little by little, until no further improvement was possible.
- *Supporting the losing service*. The compromise has the meaning of helping the service whose threshold is not met to get a more favourable solution for it.

This algorithm can handle only two competing services at a time. After the initial device assignment is evaluated, the service with the smallest score is picked as the ”losing” service. In the next iteration step, the losing service is given a 50cm advantage each round over the winning service. Then, a contended device would be assigned to the service with the closest anchor point, again, taking into consideration the above mentioned accumulated advantage.

However, this algorithm was monolithic: the evaluation function, the total score function the end condition, as well as the measuring method were all combined in a single class. Furthermore, it could only cope with illumination requests of a very limited type. Thus, we reference this algorithm mostly as background knowledge and not as an algorithm with practical value.

Greedy algorithm This is the successor to the simple space-based algorithm. It is modular, so it can be initialized with a specific end condition and service evaluation method (presented earlier in this chapter), thus it can be used to get a compromised solution for any physical property. It can

achieve compromising for more than two service with no upper limit to the number of services, although the final results may not be very noteworthy.

This algorithm uses an initialization step. In this step, every device is assigned to the service with the closest anchor point. The assignment of devices forms the initial solution which will be improved iteratively. Furthermore, for each service, a list with the contended devices is created. This list is sorted based on the distance to the anchor point, starting with the device closest to it at the top of the list.

The following procedure takes place in each iteration step.

1. Select the losing service. This is the service with the lowest score so far.
2. If the losing service changed since the previous round, clear the blacklist of used devices.
3. Find the device closest to the anchor point of the losing service that has not yet been assigned to it and is not in the list of black listed devices, and apply to it the settings of the losing request. Add the device to the blacklist.
4. Was the end condition triggered?
   (a) if no, continue to the next iteration (step 1).
   (b) if yes, are there any more candidate devices left on the sorted list?
      i. if yes, ”undo” the last change, and continue to the next iteration (step 1).
      ii. if no, end the algorithm.

The nature of the algorithm is simple: let the losing request grab as many devices as necessary in order to improve its score. Eventually, the end condition will trigger for the losing service, and all the possible devices will already be in the blacklist, and the algorithm will end.

Forfeit rights algorithm Another space-based algorithm is the *forfeit rights* algorithm. Again, this algorithm can be initialized with specific end criteria and evaluators. The same concept of helping the losing service is prevalent. During the initializing step, the same initial state is as the greedy algorithm is selected. Also, the devices are sorted again for each service, but this time in a stack; the element on the top of the stack is the device farthest away from the anchor point of the service. This time a blacklist is used, but not for devices, it is used for services. Despite the many similarities, the iteration procedure defers.

These are the steps that take place in each iteration:

1. Select the losing service. This is the service with the lowest score so far.
2. If the losing service changed since the last round, clear the blacklist of services

3. Select a *donor service*. The donor service is the service with the highest score, that also happens to conflict with the losing service (i.e. two services with `UPPER` thresholds do not conflict). The donor must not be in the black list.

4. Was there a donor?

   (a) If yes, have the donor forfeit control of a device, and pass control of that device to the losing service. The device selected was on the top of the donor's device stack. The settings of the losing service are applied to that device.

   (b) If no, end the algorithm.

5. Was the end condition triggered?

   (a) If yes

      i. "Undo" last donation

      ii. Add the donor to the blacklist

6. Continue to the next iteration (step 1)

A donor forfeits control of devices that are on top of his stack. These devices are less important for the donors, so they can afford to hand over control to the losing service in hope that the total score will improve. If the end condition is triggered as a result of a "bad" donation, the donor is blacklisted i.e. the losing service has exhausted all the possible gain that it can get out of this donor (remember that the devices on the stack are ordered by importance). Trying to get some of the other devices that belong to the donor will result in even greater loses, so this is not attempted.

Intensity-based algorithms

There are many cases were instead of assigning complete control over to a service (as is the case with most space-based algorithms), it is better to assign an intermediate setting to a device. As proof of this concept we present a very simple example of an intensity solver, the range search intensity solver.

Range Search intensity solver The range search algorithm works in two phases:

1. For each device, examine the enchantments already cast to decide the *range* of requests (upper bound and lower bound of intensity).

2. Proceed to an exhaustive search of the problem space to find the best solution.

The search range is defined as:

$$Range = [I_{lower}, I_{upper}]$$

where I_{lower} and I_{upper} are the lowest and the highest intensity levels that were requested by the conflicting services.

To apply this algorithm, all the devices involved in the conflict must support the `VariableResistanceControl` interface. Starting from the lower bound, the intensity setting is increased steadily until it eventually reaches the upper bound of requested intensity. The setting that produces the best results (according to an evaluation function e.g. highest score) is the final solution.

This algorithm has the obvious limitation that if $Range$ is very big, the search is inefficient. Furthermore, assuming that I_{test} is the intensity value tested in each iteration the equation $TotalScore$ is not guaranteed to be monotonous, something that further hinders the search for a global maxima. To deal with this problem, other global search algorithms could be used, such as stimulated annealing.

4.7 Implementation details

4.7.1 OSGi and Java

In our research we wanted to simulate an environment like the one provided by the SI model. Eventually we settled to use the OSGi framework [14]. OSGi is a dynamic module system for Java. This framework helps structure software in a modular fashion and expose desired code functionality as a *service* with a well defined API that can be utilized by others.

The OSGi also provides *an execution environment*, on top of which *bundles* can be executed. For the OSGi execution environment, several implementations exist, such as Apache Felix[19], Eclipse's Equinox [20] and makewave's Knopflerfish [21]. An OSGi bundle is the basic deployment unit of software in an OSGi platform. The platform supports a dynamic life cycle for bundles and thus it is able to install, uninstall, start, stop and update them dynamically. In turn, bundles may expose a service to the OSGi platform which can later be used by other bundles. A bundle is a usual `.jar` file that contains some extra metadata. The metadata are used by the OSGi platform to successfully start the bundle. In OSGi terms, both our prototype system and the services utilizing it should be designed as OSGi bundles. Even the software bridge of a device can be a bundle, exposing the control interfaces to the core system as a service.

There were several reasons to use OSGi in this project. First, OSGi provides the notion of the execution platform. The execution platform could be the home gateway of the home environment. Furthermore, the services designed by the SP can be deployed easily as bundles and be

handled with ease. Finally, the ability to discover services is a nice feature that can be utilized by the prototype system.

Due to time limitations, the usage of the OSGi framework was limited to only a few sample tests. The final goal is to have the prototype system expose the `HomeContext` interface to the home service bundle. The `HomeContext` interface will be the interface through which services will be able to make requests and queries to the core system.

4.7.2 Sample Service

To implement a new service that utilizes the prototype system, the user must subclass the `Service` abstract class. This class already implements the core functionality of a conceptual service, along with the methods required to make and cancel requests to the system. During start up, each service object is given a reference to the `HomeContext` object that implements our system.

To implement the service logic, the user has to override the `realRun()` method. For each service, a `Thread` object is created that, after running some initialization routines, proceeds to execute the overridden method `realRun()`. No limitation or assumption is made about the number of threads or the libraries that a service utilize; they can vary depending on the needs of the service.

For the experiments shown in chapter 5, a single sample service was designed. That sample service would make a request towards the system and after a few seconds it would complete execution.

Chapter 5

Test cases and results

5.1 Space based resolution experiment

5.1.1 Scenario Explanation

We evaluated the proposed space-based resolution algorithms by simulating a scenario where conflict regarding illumination occurs.

In this scenario the following assumptions were made regarding the environment:

- The scenario takes place in a room with dimensions set to 7 meters long, 5 meters wide and 2 meters high.
- No external illumination enters the room
- 35 spot lights are affixed to the ceiling. The spot lights are laid out in a $7x5$ matrix pattern as seen in figure 5.1.
- We named each spot light as $Spot_{ij}$ where $0 \leq i \leq 5$ and $0 \leq j \leq 7$. i represents the row of the device and j represents the column of the device in the matrix.
- The spot lights only support the `OnOffControl` interface, i.e. only two possible states: power on and power off.
- Each spot light sheds light in a conical fashion. We assume that the measured amount of illumination from a distance of 1 meter is $1000lux$ inside the cone.
- If two points in space $P1$ and $P2$ have the same distance from the spot light and $P1$ is in the light cone whereas $P2$ is not, we assume that the intensity measured at $P2$ is 10% of the intensity measured at $P1$. This 10% amounts for the amount of light that reaches $P2$ due to reflections.

- As the distance from the light source increases, the intensity of illumination is modelled to fade away according to the inverse square law. For example, an illumination measurement taken 2 meters away from the source (and still inside the light cone) should yield $250lux$.
- The evaluation for each service is done using the `EightPointLightEvaluator` evaluator (described in section 4.6.2).

The coordinates of the devices can be calculated as follows: $Coord(Light_{ij}) = (50 + (j - 1)100, 200, 50 + (i - 1)100)$ where each dimension is expressed in centimetres. For example, $Spot_{11}$ has coordinates $(x, y, z) = (50, 200, 50)$ and $Spot_{75}$ has coordinates $(x, y, z) = (650, 200, 450)$ In figure 5.1 we can see a top down view of the room. The upper left room corner has coordinates $(0, 0)$. The lower right corner of the room has coordinates $(700, 500)$.

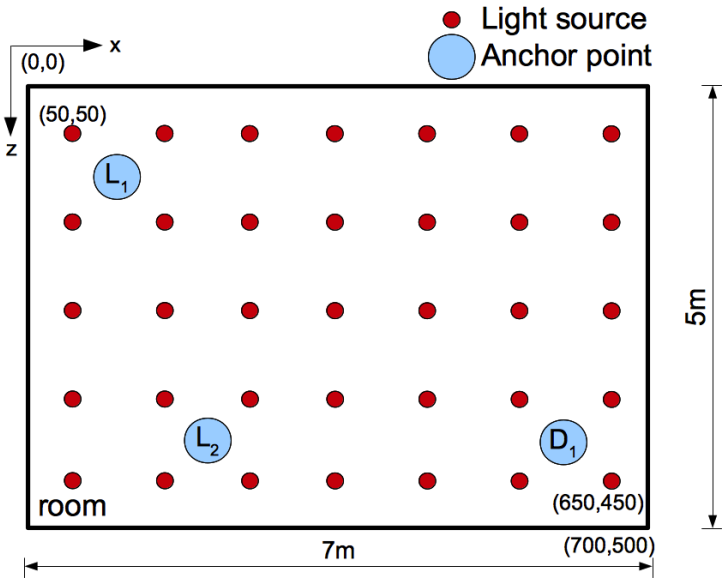

Figure 5.1: A downwards view of the experiment room

For this scenario we assumed two types of services:

1. Service L. This service makes a request for illumination with a *minimum* threshold of $1000lux$.
2. Service D. This service makes a request for illumination with a *maximum* threshold of $10lux$.

Two instances (L_1 and L_2) of service L are active in this room. Instance L_1 has an anchor point with position $(100, 100, 100)$. Instance L_2 has an anchor point with position $(200, 400, 100)$.

Furthermore, one instance of service D, D_1 is active with an anchor point at $(600, 100, 400)$. All three services make requests with an intended AoE being the room.

The three services make their requests. Then, the core system proceeds to the detection of conflicts. The conflict will be discovered using the *per-device* detection of conflicts. After the conflict has been detected, the core system enters the conflict resolution phase. The initial device assignment can be seen in figure 5.2

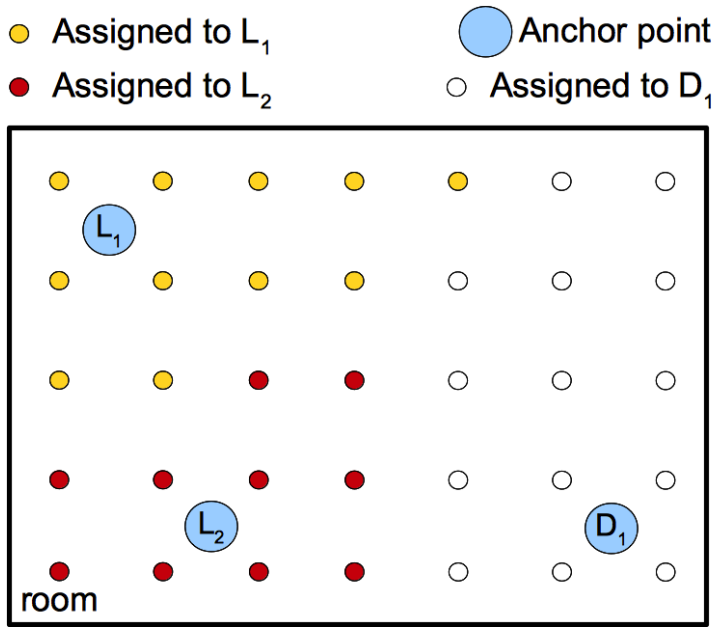

Figure 5.2: Initial device assignment

For the conflict resolution the combinations of end conditions and resolution algorithms seen in table 5.1 were used. For each of the possible combinations we calculated the following:

1. the individual score of each service
2. the ”badness” of the solution (the summation of the negative service scores only)
3. the total score for the solution.

5.1.2 Case 1: greedy algorithm and highest score end condition

The results can be seen in table 5.2. Due to the ”highest score” end condition, service D_1 was able to gain control of six extra devices compared to the initial state. If more devices would be assigned to service D_1, the total score would deteriorate, and thus algorithm chooses to stop. Services L_1 and L_2 have a surplus of illumination (their scores are more than satisfactory), but

Case No.	Resolution algorithm	End condition
1	Greedy	Highest Score
2	Greedy	Constrained Programming
3	Forfeit rights	Highest Score
4	Forfeit rights	Constrained Programming

Table 5.1: Tested combinations of algorithms and end conditions

the score of service D_1 is disappointing. The amount of illumination around the anchor point of service D_1 exceeds many times the requested.

The final assignment of settings to devices can be seen in figure 5.3.

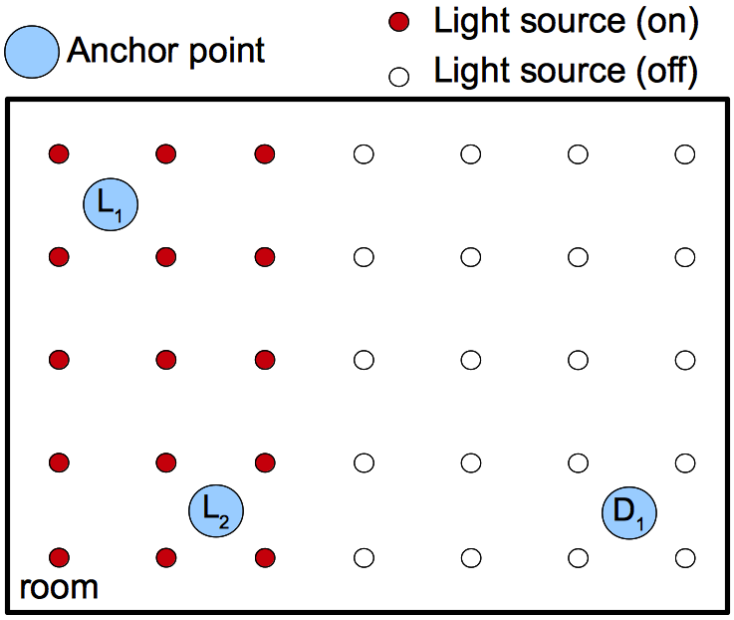

Figure 5.3: Final device assignment for case 1

5.1.3 Case 2: greedy algorithm and constrained programming end condition

Using the constrained programming end condition, service D_1 was able to get more devices than case 1. The total score is lesser than that of case 1. However the ”badness” factor has improved significantly. Moreover, services L_1 and L_2 have a score that is really close to zero (above and below accordingly). This is the effect of the constrained programming end condition; the excess

Highest Score + Greedy algorithm	
L_1	2.579174857150577
L_2	3.347747929964837
D_1	-8.406866232629483
$Badness$	-8.406866232629483
$TotalScore$	-2.479943445514069

Table 5.2: Case 1 results: Greedy algorithm and Highest Score end condition

amount of illumination of case 1 was traded in so that service D_1 may have a chance to improve (and it did significantly improve).

The results can be seen in table 5.3 and the final assignment of settings to devices can be seen in figure 5.4.

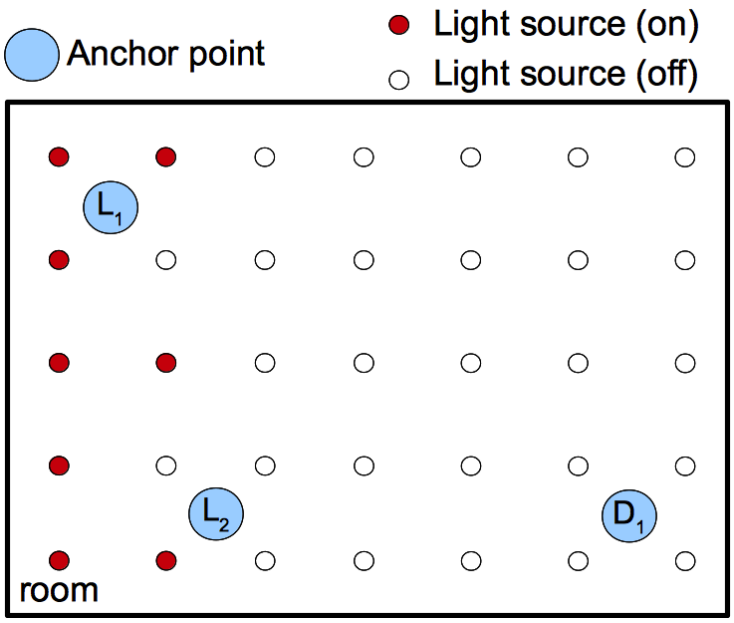

Figure 5.4: Final device assignment for case 2

5.1.4 Case 3: forfeit rights algorithm and highest score end condition

Using the forfeit rights algorithm in conjunction with the ”highest score” end condition led to some interesting results: the device assignment is the same as in case 1 (greedy algorithm + highest score end condition). The end condition has a very strong impact on the evaluation of the total scores. This does not leave any room for service D_1 to claim any more devices.

Constrained Programming + Greedy algorithm	
L_1	0.775441771757775
L_2	-0.5719388815562672
D_1	-4.397132159713623
$Badness$	-4.96907104126989
$TotalScore$	-4.193629269512115

Table 5.3: Case 2 results: Greedy algorithm and Constrained programming end condition

The results can be seen in table 5.4 and the final device assignment can be seen in figure 5.3 in case 1.

Forfeit rights algorithm and high score end condition	
L_1	2.5791748571505764
L_2	3.347747929964837
D_1	-8.406866232629483
$Badness$	-8.406866232629483
$TotalScore$	-2.4799434455140688

Table 5.4: Case 3 results: Forfeit Rights algorithm and High Score end condition

5.1.5 Case 4: forfeit rights algorithm and constrained programming end condition

In this final case we combined the forfeit rights algorithm with the constrained programming end condition. With this combination we were able to get quite favourable results, namely:

- the thresholds for services L_1 and L_2 where upheld (although barely)
- the score for the losing service D_1 is the best so far.

Here we can see the true nature of the forfeit algorithm: services will hold onto what is ”barely enough”; they will hold on to their most important devices i.e. those that were at the bottom of the stack. If there is surplus of intensity, they will pass on control of those ”surplus” devices to other services as needed.

The results can be seen in table 5.5 and the final device assignment can be seen in figure 5.5.

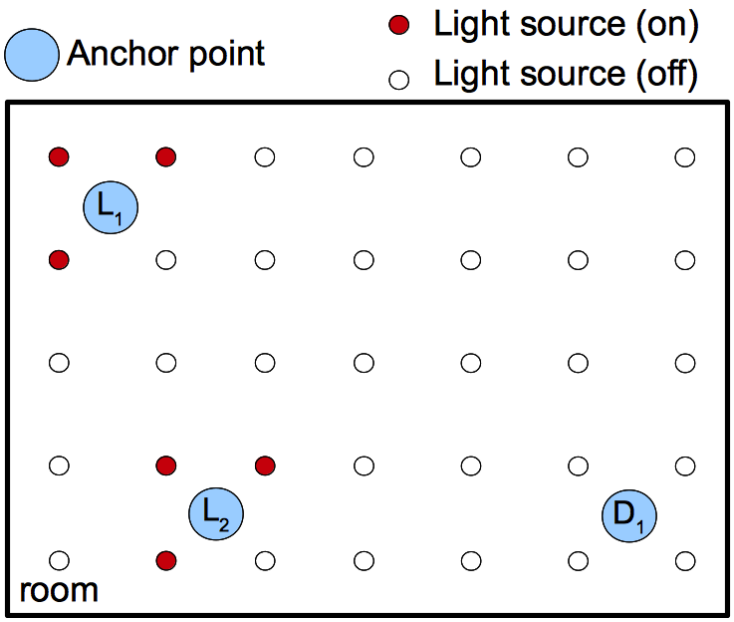

Figure 5.5: Final device assignment for case 4

Forfeit rights algorithm and high score end condition	
L_1	0.06466447241442536
L_2	0.06748716191621316
D_1	-4.185663037695831
$Badness$	-4.185663037695831
$TotalScore$	-4.05351140336519248

Table 5.5: Case 4 results: Forfeit Rights algorithm and constrained programming end condition

5.1.6 Comments

In table 5.6 we can see the results of the previous cases gathered in a single table.

The various combinations of end conditions and compromising algorithms yielded quite different results. However, the most promising results in our opinion is that of case 4. The forfeit algorithm manages to produce a solution where badness was minimum. This is the most ”fair” solution. If we compare the results of case 1 to those of case 4, we can clearly see that we failed completely to honour the request of service D_1, which is not ”fair” at all. Therefore, whenever possible, the forfeit rights algorithm should be used, in conjunction with the constrained programming end condition.

The highest score end condition proves to be quite limiting because it does not allow for

flexible device assignment. Whereas the constrained programming end condition may allow an assignment that, although worsens the total score it improves the overall badness, the highest end condition cannot do that.

Finally, the greedy algorithm is quite limited. As the number of resource requests goes up, the greedy algorithm is very well likely to stop prematurely, without reaching any decent solution. Furthermore, it may mistakenly encroach into the area of another service and take control of some critical for that service device , leading to overall lower scores. The greedy algorithm should not be used in all but the simplest of scenarios.

5.2 Intensity based resolution experiment

In this section we present the *range search* intensity-based algorithm. This algorithm is a proof of concept, and its practical usage is quite limited. A prerequisite for this algorithm is that all devices involved in the conflict are the *same* and that these devices support the `VariableResistanceControl` interface.

5.2.1 Scenario details and execution

In this scenario, there are three services (S_1, S_2, S_3) that want to control the temperature of a room. The following assumptions were made:

1. One air condition unit is available in the room. It supports the `VariableResistanceControl` interface.
2. Every point in the room will be heated to the exact intensity set by the air condition over some finite time.
3. Service S_1 makes a request for temperature set to 29^oC with an `EXACT` threshold setting.

Forfeit rights algorithm and high score end condition			
	cases 1 and 3	case 2	case 4
L_1	2.579174857150577	0.775441771757775	0.06466447241442536
L_2	3.347747929964837	-0.5675937638160895	0.07183227965639047
D_1	-8.406866232629481	-4.397132159713623	-4.185663037695831
$Badness$	-8.406866232629481	-4.964725923529713	-4.185663037695831
$TotalScore$	-2.4799434455140665	-4.189284151771938	-4.04916628562501517

Table 5.6: Summary of results

4. Service S_1 makes a request for temperature set to 26^oC with an `EXACT` threshold setting.
5. Service S_1 makes a request for temperature set to 25^oC with an `EXACT` threshold setting.
6. The `SimpleTemperatureEvaluator` was used for the satisfaction evaluation of services.

Service	Highest Score	Constrained Programming
S_1	-18	-18
S_2	-2	-2
S_3	-8	-8
$TotalScore$	-28	-28

Table 5.7: Results of the Range Search algorithm. The best solution is 27^oC.

The experiment was performed twice, once with a highest score end condition and then with a constrained programming end condition. First, the conflict is detected using per-device conflict detection. Then, the search range $Range$ is set to $[25, 29]$. All the discrete values in $Range$ are tested. The result with the best score was achieved for a setting of 27^oC. Finally, we can see the state of the `EnchantmentEntry` object after the compromise has occurred in figure 5.6. The active enchantment is an enchantment cast by the system and it is marked appropriately as system enchantment.

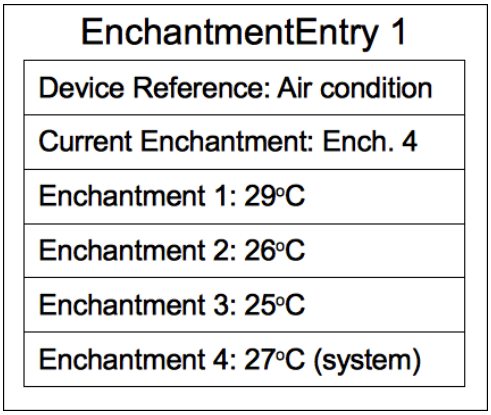

Figure 5.6: Final status of `EnchantmentEntry` for device Air condition.

5.2.2 Comments

What is immediately clear from the results is that the score does not change even if the end conditions change. This is because of the negative scores of the services. A service with an `EXACT` threshold can only hope to achieve a score of 0 as the highest possible score; anything below and above the threshold results in penalties. When the scores of all services are negative, the constrained programming end condition and the highest score condition behave the same.

Although the algorithm can take care of such simple scenarios, its prerequisites severely limit the deployment opportunities. Furthermore, the exhaustive search of the problem space may sometimes be impossible. For example, in an illumination scenario where all devices are the same, a different approach should be used because the intensity range might be much bigger.

However, the only characteristic that sets it apart from the other algorithms is the ability to impose a homogeneous setting to the conflict AoE. Depending on the situation, this may be the most desirable solution.

5.3 Conflict detection using Area of Effect

In the previous two experiments, conflict detection was done on a per-device fashion. In this experiment we demonstrate conflict detection using information about AoE in a scenario involving resource requests for sound and noise levels.

5.3.1 Scenario details and execution

In this scenario, we assumed the following things about the environment.

- Two rooms connected via a conduit.
- The size of the rooms is 3 meters long, 2 meters high and 3 meters wide (3x2x3).
- The conduit connecting the two rooms is a door. The door supports the `OnOffControl` and `SoundBlocker` interface.
- The initial state of the door is assumed to be open.
- An omni-directional sound source exists at position (550,100,150).
- Any sound passing through the closed door incurs a heavy penalty: the resulting amount of sound is ten times smaller than the original one.
- Any sound passing through the open door suffers no penalties.
- Modelling of sound is based on sound pressure. Reflections have not been considered.

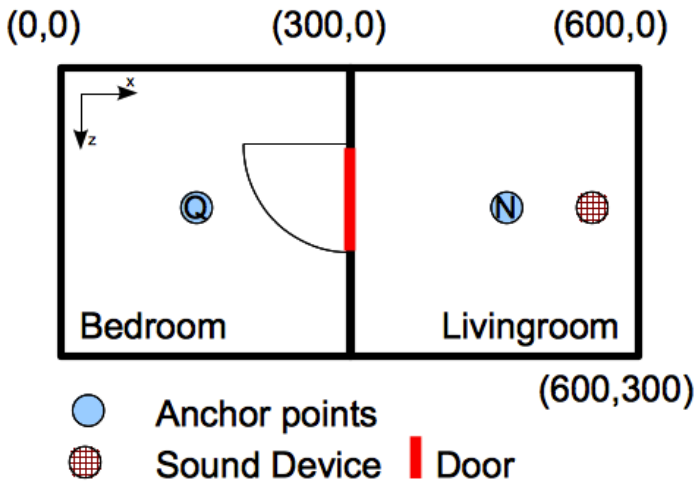

Figure 5.7: A downwards view of the experiment environment

Furthermore, we assume two services $Noisy$ and $Quiet$ with anchor points N at $(450, 100, 150)$ and$Q$ at $(150, 100, 150)$ respectively. $Noisy$ service wants to control the omni-directional sound device (maybe as part of a set of other devices) and requests an intensity level of $50db$ with an $EXACT$ threshold setting. The $Quiet$ service makes a request for sound and noise levels with intensity $25db$ and an $UPPER$ threshold setting. In both cases the requested area of effect is the whole room (the room where the service executes).

During the ideal solution creation step, an enchantment is cast to the sound device on behalf of service $Noisy$. No enchantment is cast on behalf of service $Quiet$, because no device implementing the `SoundSource` interface is found inside that room.

During the per-device conflict detection, no conflict is detected, because the enchantment lists for all devices either contain zero or one enchantments.

During the area conflict detection phase, a conflict is discovered. This is because the estimated effect of the omni-directional device at the anchor point of $Quiet$ exceeds the requested threshold. The system proceeds as follows:

1. Cast an enchantment to the omni-directional sound device on behalf of service $Quiet$. The enchantment dictates the device to be turned off.

2. Find a *path* from the device to the anchor point of service $Quiet$. Cast enchantments to all conduits in that path. The enchantments specify that the conduits should be closed.

The final state of enchantments and enchantment lists can be seen in figure 5.8. I_{Quiet} and I_{Noisy} describe the intensity at anchor points Q and S respectively. The initial state is to assign the omni-directional sound device to service $Noisy$. Control of the door is normally assigned to

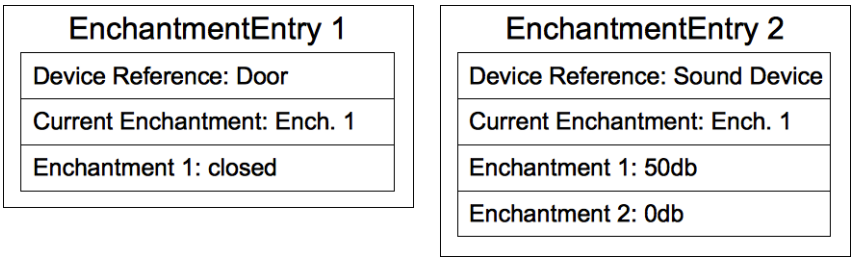

Figure 5.8: Final enchantment state.

	Before	After
I_{Quiet}	37.95880017344075	18.027737252919753
I_{Noisy}	50	50
$Score_{Quiet}$	N/A	1.419987888958309
$Score_{Noisy}$	N/A	0
$TotalScore$	N/A	1.4199841167448681

Table 5.8: Results using AoE conflict detection

service $Quiet$, since only one enchantment is cast upon it. The score evaluation for services as well as the total intensity before and after the detection can be seen in table 5.8. We can see that, although initially there was a conflict (the threshold request of service $Quiet$ was not upheld), the system was able to detect that conflict and resolve it.

5.3.2 Comments

Again, this scenario helps to demonstrate the benefits of employing an area of effect conflict detection algorithm. Although the physics simulation used in the current system is not accurate and makes some assumptions regarding the environment, we can still argue that this method is a valid method to discover conflicts and that has potential value when applied in real life situations.

Chapter 6

Future work and improvements

The current prototype system serves as a successful demonstration of a system that utilizes the basic concepts behind AoE and compromising techniques. However, it has some severe shortcomings that must be improved before the system can be deemed worthy of deployment in real life. Further research and improvements must be made on these shortcomings.

Firstly, the *physics simulation* used is very simplistic. Developing a full-blown physics simulator is a topic worthy of its own research altogether, and was deemed to be out of the scope of this research. Improving the physics simulation can increase the accuracy of predictions and improve the compromising results vastly. As future improvements on the prototype system, a new, modular physics simulator should be designed, allowing for easier modelling of devices and the surrounding environment.

Secondly, the system in its current form cannot handle some specific scenarios. For example, the system cannot handle in a scenario with illumination requests with `EXACT` thresholds and devices that support only the `OnOffControl` interface. Such scenarios have to be studied further and analysed in order to produce more effective compromising algorithms that can deal with them.

Thirdly, a mechanism for *ad-hoc* selection of the compromising algorithm should be developed. In the current prototype system, the compromising algorithm is selected during the scenario set up. This of course is unacceptable for a system targeted for deployment in a live environment. Deciphering which algorithm would potentially yield promising results could be an interesting topic for future research.

Regarding other possible compromising algorithms, we would like to see some combined and hybrid algorithms being implemented. In a live environment the system has access to user information. Hybrid compromising techniques could potentially use that information to deliver even better compromising solutions.

Another part of the prototype system that needs further development is the service API. In its current form it allows nothing but the sending and cancelling of requests. Design of a system

based on the principles of the SI platform requires further research and a more comprehensive and versatile API for accurate control of services. Furthermore, security issues should be taken into consideration.

For a live system we must take into account *motion*. It is entirely possible to have moving anchor points. Introducing moving anchor points would add another level of complexity to the system because the ideal solutions in many cases change as soon as the anchor point moves.

Finally, we must consider the *user interaction with the system*. Such a system aims to support the user and automate the handling of the environment around the user. The user must be able to express its preferences and also take manual control whenever necessary.

Chapter 7

Conclusions

In this research we presented a prototype system capable of using the notion of the area of effect and compromising techniques to resolve conflicts between services in the home network system. We argue that the system offers a valuable solution to the feature interaction problem. Not dealing with the FI problem leads to conflicts between services and erratic, non-deterministic behaviour. Our system is not only able to avoid this behaviour but also able to produce device settings that satisfy the requests of services, as demonstrated by the experiments shown. Furthermore, compared to other conflict resolution techniques, the proposed system has the advantage that no conflicting services are needed to be paused or terminated.

At a theoretic level, we contributed the idea of using area of effect information and compromising algorithms and defined the problem as a maximization problem. We then propose two conflict detection methods, a per-device method and a method based on estimation of intensity at a given point. Furthermore, we demonstrated the idea of space-based and intensity-based compromising algorithms in action as well as their effectiveness by simulating a variety of different scenarios.

We believe that the proposed system augments the SI model and can be integrated into it. The combination of these two technologies could lead to a viable platform for delivery of services in the home network system with many advantages over conventional models of service delivery, thus becoming the base for a new dawn of complex services in the HNS.

Appendix A

XML configuration files

The following configuration files represent the information that the system has regarding the scene of the home environment. Each abstract scene element (a device, a room, a conduit, others) has position and size information. All objects are assumed to be box-shaped. Devices and their characteristics are declared inside the `Room` tags of the room they are in.

A.1 configuration file for illumination scenario: `7x5.xml`

This file contains only a single room definition. In it, there are thirty-five declarations of illumination devices such as those used in experiment 5.1.

```
<House>
<RoomList>
<Room>
        <UID>LivingRoom</UID>
        <Size><X>700</X><Y>200</Y><Z>500</Z></Size>
        <Pos><X>0</X><Y>0</Y><Z>0</Z></Pos>
        <Device>
                <Type>DirectionalIlluminationDevice</Type>
                <UID>Light11</UID>
                <Pos><X>50</X><Y>200</Y><Z>50</Z></Pos>
                <Lumen>1000</Lumen>
        </Device>
        <Device>
                <Type>DirectionalIlluminationDevice</Type>
                <UID>Light12</UID>
                <Pos><X>150</X><Y>200</Y><Z>50</Z></Pos>
                <Lumen>1000</Lumen>
        </Device>
        <Device>
                <Type>DirectionalIlluminationDevice</Type>
                <UID>Light13</UID>
                <Pos><X>250</X><Y>200</Y><Z>50</Z></Pos>
                <Lumen>1000</Lumen>
        </Device>
        <Device>
                <Type>DirectionalIlluminationDevice</Type>
```

```
        <UID>Light14</UID>
        <Pos><X>350</X><Y>200</Y><Z>50</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light15</UID>
        <Pos><X>450</X><Y>200</Y><Z>50</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light16</UID>
        <Pos><X>550</X><Y>200</Y><Z>50</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light17</UID>
        <Pos><X>650</X><Y>200</Y><Z>50</Z></Pos>
        <Lumen>1000</Lumen>
</Device>

<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light21</UID>
        <Pos><X>50</X><Y>200</Y><Z>150</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light22</UID>
        <Pos><X>150</X><Y>200</Y><Z>150</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light23</UID>
        <Pos><X>250</X><Y>200</Y><Z>150</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light24</UID>
        <Pos><X>350</X><Y>200</Y><Z>150</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light25</UID>
        <Pos><X>450</X><Y>200</Y><Z>150</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light26</UID>
        <Pos><X>650</X><Y>200</Y><Z>150</Z></Pos>
```

```
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light27</UID>
        <Pos><X>650</X><Y>200</Y><Z>150</Z></Pos>
        <Lumen>1000</Lumen>
</Device>

<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light31</UID>
        <Pos><X>50</X><Y>200</Y><Z>250</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light32</UID>
        <Pos><X>150</X><Y>200</Y><Z>250</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light33</UID>
        <Pos><X>250</X><Y>200</Y><Z>250</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light34</UID>
        <Pos><X>350</X><Y>200</Y><Z>250</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light35</UID>
        <Pos><X>450</X><Y>200</Y><Z>250</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light36</UID>
        <Pos><X>550</X><Y>200</Y><Z>250</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light37</UID>
        <Pos><X>650</X><Y>200</Y><Z>250</Z></Pos>
        <Lumen>1000</Lumen>
</Device>

<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light41</UID>
        <Pos><X>50</X><Y>200</Y><Z>350</Z></Pos>
        <Lumen>1000</Lumen>
```

```
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light42</UID>
        <Pos><X>150</X><Y>200</Y><Z>350</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light43</UID>
        <Pos><X>250</X><Y>200</Y><Z>350</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light44</UID>
        <Pos><X>350</X><Y>200</Y><Z>350</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light45</UID>
        <Pos><X>450</X><Y>200</Y><Z>350</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light46</UID>
        <Pos><X>550</X><Y>200</Y><Z>350</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light47</UID>
        <Pos><X>650</X><Y>200</Y><Z>350</Z></Pos>
        <Lumen>1000</Lumen>
</Device>

<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light51</UID>
        <Pos><X>50</X><Y>200</Y><Z>450</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light52</UID>
        <Pos><X>150</X><Y>200</Y><Z>450</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
        <Type>DirectionalIlluminationDevice</Type>
        <UID>Light53</UID>
        <Pos><X>250</X><Y>200</Y><Z>450</Z></Pos>
        <Lumen>1000</Lumen>
</Device>
<Device>
```

```
                <Type>DirectionalIlluminationDevice</Type>
                <UID>Light54</UID>
                <Pos><X>350</X><Y>200</Y><Z>450</Z></Pos>
                <Lumen>1000</Lumen>
        </Device>
        <Device>
                <Type>DirectionalIlluminationDevice</Type>
                <UID>Light55</UID>
                <Pos><X>450</X><Y>200</Y><Z>450</Z></Pos>
                <Lumen>1000</Lumen>
        </Device>
        <Device>
                <Type>DirectionalIlluminationDevice</Type>
                <UID>Light56</UID>
                <Pos><X>550</X><Y>200</Y><Z>450</Z></Pos>
                <Lumen>1000</Lumen>
        </Device>
        <Device>
                <Type>DirectionalIlluminationDevice</Type>
                <UID>Light57</UID>
                <Pos><X>650</X><Y>200</Y><Z>450</Z></Pos>
                <Lumen>1000</Lumen>
        </Device>

</Room>
</RoomList>
</House>
```

A.2 configuration file for temperature scenario: `TemperatureScenario.xml`

This file contains the description of a single room and the declaration of an air condition device.

```
<House>
<RoomList>
<Room>
        <UID>Kitchen</UID>
        <Size><X>300</X><Y>300</Y><Z>300</Z></Size>
        <Pos><X>0</X><Y>0</Y><Z>0</Z></Pos>
        <Device>
                <Type>AirCon</Type>
                <UID>Aircon1</UID>
                <Pos><X>250</X><Y>300</Y><Z>300</Z></Pos>
        </Device>
</Room>
</House>
```

A.3 configuration file for conflict descovery based on area of effect: `SoundScenario.xml`

In this configuration file we can see how conduits are declared. Each conduit has two endpoints that must point to rooms declared earlier in the configuration file. Finally, the position and size of the conduit express its actual position and orientation, since the conduit is considered to be just a two dimensional plane. Special care must be taken so that the coordinates will later overlap with a wall so that the conduit can be associated with.

```
<House>
<RoomList>
<Room>
        <UID>Bedroom</UID>
        <Size><X>300</X><Y>300</Y><Z>300</Z></Size>
        <Pos><X>0</X><Y>0</Y><Z>0</Z></Pos>
</Room>
<Room>
        <UID>Livingroom</UID>
        <Size><X>300</X><Y>300</Y><Z>300</Z></Size>
        <Pos><X>300</X><Y>0</Y><Z>0</Z></Pos>
        <Device>
                <Type>OmniSoundSourceDevice</Type>
                <UID>Sound1</UID>
                <Pos><X>250</X><Y>200</Y><Z>150</Z></Pos>
                <Db>80</Db>
        </Device>

</Room>
</RoomList>
<ConduitList>
        <Conduit>
                <UID>con1</UID>
                <Type>Door</Type>
                <Endpoint>Livingroom</Endpoint>
                <Endpoint>Bedroom</Endpoint>
                <Size><X>0</X><Y>200</Y><Z>100</Z></Size>
                <Pos><X>300</X><Y>0</Y><Z>100</Z></Pos>
        </Conduit>
</ConduitList>
</House>
```

Appendix B

Execution Logs

In this chapter we present the output results of the experiments. Each solving algorithm logs the final solution in a log file. In that log file we store the final scores of the services, the total score as interpreted by the solving algorithm and the final state of the devices. We proceed to list these files.

B.1 Space based resolution experiment results

The service scores of this section were evaluated using the `EightPointLightEvaluator` as described in section 4.6.2. The calculation formula is (4.1) which is essentially a decibel calculation. A score of 0 means that the final estimated intensity is exactly equal to the requested intensity, whereas a score of 3 means that the final estimated intensity value was roughly two times bigger.

The total score reported when the constrained programming end condition is used is actually equivalent to the ”badness” score as reported in the experiments of section 5.

B.1.1 Case 1: High score + Greedy

In this experiment, the final estimated intensity for services L_1 and L_2 is a little bit less and a little bit more than twice the original threshold. However, the threshold of service D_1 (that has a score of -8.4) was violated with an intensity that is close to 7 times brighter than the originally requested intensity.

```
Final solution:
_________________

device: Light35false
device: Light27false
device: Light14false
device: Light17false
device: Light25false
device: Light23true
device: Light32true
```

```
device: Light43true
device: Light47false
device: Light37false
device: Light45false
device: Light46false
device: Light15false
device: Light53true
device: Light54false
device: Light44false
device: Light24false
device: Light41true
device: Light42true
device: Light13true
device: Light34false
device: Light11true
device: Light21true
device: Light22true
device: Light31true
device: Light55false
device: Light52true
device: Light12true
device: Light57false
device: Light26false
device: Light36false
device: Light33true
device: Light56false
device: Light51true
device: Light16false
service: Dark1 Score: −8.406866232629483
service: Light1 Score: 2.579174857150577
service: Light2 Score: 3.347747929964837
Total Score: −2.479943445514069
```

B.1.2 Case 2: Constrained + Greedy

In this experiment, the results for services L_1 and L_2 stayed inside a ± 1.2 margin of the requested intensity. The score for service D_1 was also improved, since the final estimated intensity is only 2.7 times bigger , a vast improvement over the results of case 1.

```
Final solution:
_____________________
device: Light33false
device: Light25false
device: Light12true
device: Light15false
device: Light23false
device: Light21true
device: Light27false
device: Light41true
device: Light45false
device: Light35false
device: Light43false
device: Light13false
device: Light44false
device: Light51true
device: Light52true
```

```
device: Light42false
device: Light57false
device: Light22false
device: Light36false
device: Light37false
device: Light32true
device: Light16false
device: Light26false
device: Light17false
device: Light53false
device: Light47false
device: Light11true
device: Light56false
device: Light55false
device: Light24false
device: Light34false
device: Light31true
device: Light54false
device: Light46false
device: Light14false
service: Dark1 Score: −4.397132159713623
service: Light2 Score: −0.5719388815562672
service: Light1 Score: 0.775441771757775
Total Score: −4.96907104126989
```

B.1.3 Case 3: High score + Forfeit rights

The results here are the same with case 1.

```
Final Scores: ——————————————————
−8.406866232629483 3.347747929964837 2.5791748571505764
Final sloution:
——————————————————
device: Light31true
device: Light23true
device: Light56false
device: Light11true
device: Light13true
device: Light25false
device: Light21true
device: Light16false
device: Light36false
device: Light33true
device: Light43true
device: Light41true
device: Light42true
device: Light46false
device: Light47false
device: Light37false
device: Light55false
device: Light17false
device: Light34false
device: Light35false
device: Light27false
device: Light14false
device: Light24false
```

```
device: Light15false
device: Light51true
device: Light57false
device: Light45false
device: Light54false
device: Light53true
device: Light22true
device: Light32true
device: Light26false
device: Light52true
device: Light44false
device: Light12true
service: Dark1 Score: −8.406866232629483
service: Light2 Score: 3.347747929964837
service: Light1 Score: 2.5791748571505764
Total Score: −2.4799434455140688
```

B.1.4 Case 4: Constrained + Forfeit rights

In this experiment, the estimated intensity for services S_1 and S_2 was barely above that of the requested intensity, just 1.01 times more. The results for service D_1 further improved, resulting in an estimated final intensity roughly 2.5 times bigger that that requested.

```
Final Scores: ——————————————————
0.06466447241442536 0.06748716191621316 −4.185663037695831
Final sloution:
——————————————————
device: Light34false
device: Light26false
device: Light13false
device: Light16false
device: Light24false
device: Light22false
device: Light31false
device: Light42true
device: Light46false
device: Light36false
device: Light44false
device: Light11true
device: Light45false
device: Light14false
device: Light52true
device: Light53false
device: Light43true
device: Light23false
device: Light37false
device: Light41false
device: Light12true
device: Light33false
device: Light17false
device: Light21true
device: Light27false
device: Light54false
device: Light51false
device: Light57false
device: Light56false
```

```
device: Light25false
device: Light35false
device: Light32false
device: Light55false
device: Light47false
device: Light15false
service: Light1 Score: 0.06466447241442536
service: Light2 Score: 0.06748716191621316
service: Dark10 Score: −4.185663037695831
Total Score: −4.185663037695831
```

B.2 Intensity based resolution experiment results

In this experiment, the scores for services were calculated using the `SimpleTemperatureEvaluator`. The best total score was achieved for the setting of 27^oC.

```
Final Scores: ——————————
−2.0 −18.0 −8.0
Final sloution:
——————————
device: Aircon1 27.0
service: xmlparser.ServiceApi.TestService2 Score: −2.0
service: xmlparser.ServiceApi.TestService1 Score: −18.0
service: xmlparser.ServiceApi.TestService3 Score: −8.0
Total Score: −28.0
```

B.3 Conflict detection experiment results

This evaluator models the decaying of sound pressure. Furthermore, if the sound has to go through a wall or a closed conduit, we assume that it suffers a tenfold decrease. The result for service $Noisy$ is exactly zero, meaning that the estimated intensity is exactly the same as the requested intensity. Furthermore, we can see how the estimation of sound pressure changed for the anchor point of service $Quiet$ at position $(150, 200, 150)$. A tenfold drop in sound pressure translates into almost $20db$ less sound pressure at the anchor point, thus satisfying the request of service $Quiet$.

```
Area conflict detected: Device Sound1 ,Service: Quiet, Effect: 37.95880017344075
Intensity at: xmlparser.Vector3[x=150.0,y=200.0,z=150.0]  18.027737252919753
Intensity at: xmlparser.Vector3[x=450.0,y=200.0,z=150.0]  50.00004342923105
Final Scores: ——————————
−Infinity −Infinity
Final sloution:
——————————
device: Sound1 true 50.0
device: con1 false 0.0
service: Quiet Score: 1.419987888958309
service: Noise Score: −3.7722134409618327E−6
Total Score: 1.4199841167448681
```

Bibliography

[1] ″次世代ホームネットワークが描く新たな価値進化時代へ向けた挑戦″, pp.58-60, 次世代 IP ネットワーク推進フォーラム ホーム ネットワーク WG

[2] Y. Kiyoumi, ”A task allocation method considering resource availability for providing services to Home Network Environment”, Master thesis, 2010

[3] Pattara Leelaprute, ”Resolution of Feature Interactions in Integrated Services of Home Network System”, Proceedings of Asia-Pacific Conference on Communications, 2007

[4] Pattara Leelaprute , Takafumi Matsuo , Tatsuhiro Tsuchiya and Tohru Kikuno, ”Detecting Feature Interactions in Home Appliance Networks”, Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008

[5] Marios Sioutis Takashi Okada Junya Nakata Junsoo Kim Azman Osman Lim and Yasuo Tan, ”Area of effect and compromising techniques to improve availability of services in a home network environment”, IEICE Technical Report, Japan, 2010

[6] L. du Bousquet. “Feature Interaction Detection using Testing and Model-checking – Experience Report”, World Congress in Formal Methods, France, 1999

[7] M. Calder, A. Miller, “Using SPIN for Feature Interaction Analysis – A Case Study”, Model Checking Software 8th International SPIN Workshop. Toronto, Canada, 2001

[8] A. Metzger and C. Webel, ”Feature interaction detection in building control systems by means of a formal product model”, Feature Interactions in Telecommunications and Software Systems VII, pp.105–122, 2003

[9] ISO, Information Processing Systems, Open Systems Interconnection, ”LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour”, Switzerland, 1989

[10] D. Amyot, L. Charfi, N. Gorse et al, “Feature Description and Feature Interaction Analysis with Use Case Maps and LOTOS”, Sixth International Workshop on Feature Interactions in Telecommunications and Software Systems FIW ’00, Scotland. 2000

[11] H. Selig, The visual discrimination of intensity and the Weber-Fechner Law, USA, 1924

[12] Theodore L. Turcoy, Bernhard von Stengel, Game Theory, CDAM Research Report LSE-CDAM-2001-09 , 2001

[13] F. Rossi, P. Van Beek, T.Walsh, Handbook of Constraint Programming, Elsevier, 2006

[14] OSGi Alliance, http://www.osgi.org

[15] Netflix streaming services, http://www.netflix.com

[16] Digital Living Network Alliance (DLNA), http://www.dlna.org

[17] Universal Plug and Play, http://www.upnp.org

[18] ECHONET consortium, http://www.echonet.gr.jp/english/index.htm

[19] Apache Felix, an OSGi R4 Service Platform implementation, http://felix.apache.org

[20] Eclipse Equinox, an OSGi R4 Service Platform implementation, http://www.eclipse.org/equinox/

[21] Makewave’s Knopflerfish, an OSGi Service Platform implementation, http://www.knopflerfish.org