Integration of an intelligent home control system to a central bus architecture

Anders Jensen

Kongens Lyngby 2012 07-September
IMM-BSc

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Resumé

Denne rapport beskriver integrationen af et intelligent hjemme styrings system til en central publish/subscriber baseret message bus arkitektur. Projektet er motiveret af DTU's deltagelse i 2012 Europe Solar Decathlon konkurrence hvor målet er at bygge det mest energirigtige og innovative hus. I nyere tider er det blevet meget vigtigt at udvikle og bygge energi rigtige og intelligente huse. Forskellige hjemme styrings systemer har været på markedet i lang tid men indtil videre har de ikke kunne samarbejde ordentligt. Målet med vores gruppes system er at muliggøre integrationen af flere forskellige hjemme styrings systemer til et komplet system. Denne rapport fokuserer på integrationen af LK Schneider's IHC system til en central message bus. Bedømmelses kriterierne for systemet er bl.a at systemet skal kunne udvides sådan så det ikke blot kan genbruges når DTU deltager i fremtidige Solar Decathlon konkurrencer men også hvis andre interreserede parter ønsker at integrere et IHC system til en central bus arkitektur. Systemet skal være meget robust sådan så run-time fejl kun sker meget sjældent. Desuden er en rimelig respons tid på at fx tænde og slukke et lys også et krav. Det udviklede system integrerer IHC'en fra LK Schneider Electrics til en bus med et interface af forskellige forespørgsler der kan gives til IHC'en fra de andre systemer på bussen. System henter og processerer automatisk en hvilken som helst projektfil på IHC'en og klargører den til brug med bussen. Få tests er blevet udført i et tidligt stadie af systemet som viste at systemet møder de kriterier som der blev sat.

Keywords: IHC, Lauritz Knudsen, LK, Schneider Electrics, IHC OpenAPI, Event-based communication, IHC integration.

Abstract

This paper describes the integration of an Intelligent Home Control (IHC) system to a centralized publish/subscriber based message bus architecture. The project is motivated by DTU participation in the 2012 Europe Solar Decathlon contest on building the most energy efficient and innovative house. In modern times developing energy efficient and intelligent houses have become increasingly important. Several different home control systems have been around for a while on the market but they cannot cooperate. The goal of our groups system it to integrate a number of different home control systems into one complete system. The scope of this report is the integration of LK Schneider's IHC system to a centralized message bus. The criteria of the system is that the solution is extendable so that it not only can be reused in the next iteration of the system when DTU will participate in the next Solar Decathlon contest but also if other interested parties wish to integrate an IHC System from LK Schneider Electrics to a centralized bus architecture. The system should be very robust so very few run-time errors occur. Additionaly a reasonable response time on fx turning a light on/off is also required. The proposed system integrates the IHC from LK Schneider to a bus with an interface of available requests for the other subsystems on the bus. The system automatically fetches and processes any projectfile on the IHC and readies it for use with the bus. Few early-stage tests have been made that proved that the system meets the criteria set out.

Keywords: IHC, Lauritz Knudsen, LK, Schneider Electrics, IHC OpenAPI, Event-based communication, IHC integration.

Acknowledgements

I would like to thank LK Schneider Electrics for being very generous with their support and equipment to our group - both for the testing and the actual installment. Also for aiding our group with training sessions. Thanks goes out to Jesper Plass, Liza Lindbjerg, Pelle Fischer Nielsen all from Schneider Electrics. A special thanks goes out to Claus Jørgensen from LK Schneider Electrics for precious know-how support regarding how to program the projectfile on the IHC Control module and for being so helpful.

I would like to thank my fellow students working on the project, without them this system would not have been possible to create. It has been a valuable experience working with these people and one that i have learnt a lot about teamwork from. A thanks goes out to my friend Patrick Dennis Kassow and my colleague Morten Schnack for reading the report through and commenting on it. A special thanks goes out to my supervisor Christian Damsgaard Jensen for his great supervision of the project and his aid to this report.

Contents

Chapter 1

Introduction

DTU is participating in a competition called Solar Decathlon 2012 in which the goal is to create a house that is as energy efficient and innovative as possible. The house is called "FOLD" and this project is part of the intelligent element of FOLD and therefore i will start by describing the project and how my project relates to it.

The team building FOLD primarily consists of students. To provide us with knowledge, know-how and materials there are also professors and industry sponsors connected to the team. Together this makes up Team DTU. In the team we are 8 students working on making FOLD an intelligent by building a control system for it. Each of us have different responsibilities and tasks. The overall architecture of the system is a decentralized, decoupled system with many subsystems communicating through a message bus. This architecture will be described further in section 2.2. Some of us are working on the bus and some of us are working on integrating subsystems to the bus. The innovative element of this system is that we will have subsystems from different companies all integrated into the same system which has not bee done before. This will give us the advantage of having one CCU (Central Control Unit) that

can make decisions for the house based on information from many different subsystems from different companies. One of the company sponsors to FOLD is Schneider Electric who has sponsored their IHC (Intelligent Home Control) system to our software control group. The IHC system is a home control system that is specially good at controlling lighting, power and alarms. This project focuses on integrating the IHC system to the bus so that the CCU receives information from the IHC system through the bus and is able to send requests to it fx turn on a light. The integration of this system means that some middleware layer is needed between the IHC and the bus system in order to offer a common API for the bus to communicate with the IHC system through.

When creating the middleware layer between the bus and the IHC system my main goals are:

- Include as many features as possible making almost all actions the IHC system can possibly do, available for the CCU through the bus.
- Giving the CCU as much control over the IHC system as possible while still encapsulating unwanted behaviour.
- Automating as much of the IHC communication as possible, in order to minimize coding in the future when other programmers are going to develop the next iteration of this control system.

In this project i will need information from other student groups working on FOLD. Interior furniture designers working on FOLD might have special wishes for how the lighting will be, and electricians who are going to install the actual lights or power might have some restrictions regarding where it can be placed. Therefore i need to gather information from different parties working on FOLD in order to make decisions on how to design the IHC system which means that i will partly act as a communications person between our software group and the rest of Team DTU.

The IHC system is going to have many controllable elements connected to it (lamps, power outlets, alarms and more). These elements are called resources. We want these resources to function both like in a non-intelligent standard house where you can turn on lights via buttons in the house fx

but we also want the CCU to be able to control these resources connected to the IHC system. This means that the CCU must be able to change the state of the resources but it also means that the CCU must be informed when the resources state is changed by a button in the house. Additionally we would also like other subsystems (not only the CCU) to be able to know when a light has been turned on/off. Based on these demands an event-based control system is well suited as a solution. When a resource is switched on in the house with a button an event will be fired that tells the CCU that now the light has been switched on. Similarly if the CCU sends a request to the IHC system to turn on a light, then an event will be sent to the other subsystems informing them that this light has been turned on. This resource-event system is part of the middleware layer and will be described in detail in the design chapter 3.2.

The resources connected to the IHC are created inside the IHC system. This means that they cannot be communicated with directly by the event system in the middleware because they have not been translated yet. Therefore the middleware layer must also contain a translation layer that reads the resources inside the IHC system and process them so they can be used by the event system. This processing of the resources must preferably be done generically so if new resources are added or deleted then no new coding is needed. This process is described in detail in the implementation chapter 4.

Most of us in the group were very familiar with C# or at least Java which resembles C# very closely therefore C# was chosen as the programming language used. Programming languages could have been decided by the specific subsystem designer, but to minimize additional translation layers and making the system easy to maintain we chose to use C# all over the system. Since a service oriented architecture is needed WCF in .Net was chosen with Visual Studio as the work environment. We had all worked with this before so not much attention was given to this decision. For programming the IHC system there are two technologies developed by Schneider Electrics that must be used in order to communicate with the IHC. These are LK Visual and OpenAPI. LK Visual is used to program the resources inside the IHC system described further in section 2.3.1. OpenAPI is used by the event system to communicate with the IHC after translation of resources has been done.

The evaluation criteria for the integration layer are that it should be able to control all resources in the IHC system meaning that it should be able to receive requests on the resources to change their state. Vice versa it should also send out events when resources change their state in the IHC. The code generating the resources from inside the IHC system, should be as generic and automated as possible encapsulating the underlying IHC mechanics from the bus and making it easy to maintain the system. The layer must try to facilitate as many wishes as possible from the different student groups.

At this point in the project almost all wishes from the different student groups at DTU have been met. Some of the wishes were not possible to install due to space limitations in the physical house. A generic resource-event system has been implemented and some functional tests have been performed that show that a lamp can be turned on by a request from the bus. The same goes for dimmable lamps and power outlets. Many resource-types have however not been tested at all, since instalment of many electrical appliances in the house were delayed for unknown reasons. So some resource-types have been tested while others haven't. However we do know that the general framework of the resource-event system works since some of the resources have been functionally tested. And since all resources use the same pattern and framework getting them to function when they have been installed shouldn't prove too hard.

1.0.1 Organization of report

The report is organized into five chapters apart from the introduction.

The analysis chapter analyses the problem by describing the domain and what solutions already exist. After this a description of the technologies used are given and the chapter ends with a list of requirements for the system.

The design chapter describes what the overall thinking pattern was when the system was created. In this chapter the design of the virtual division of the rooms are described along with the placement of the resources in the house. After this the event-based communication system is described.

The implementation chapter describes in detail how the projectfile is fetched and processed so it can be used by the event system. This chapter is more focused on the detailed implementation of the system.

The Evaluation chapter evaluates the tests carried out and the issues that are unsolved. After this the possible future extensions are described and some initial conclusions are made.

The conclusion concludes and wraps up the project describing briefly what the problem was and how it was solved and then a section with test conclusions and experiences learnt are included.

Chapter 2

Analysis

2.1 Motivation

During the last 20-30 years home control systems have been gaining increasing attention from not only private home owners but also corporations and industries. Old fashioned simple relays and simple light bulbs are getting replaced by smart relays that fx turn off when leaving the house and dimmable lights that can be adjusted according to the amount of light needed. There are numerous companies worldwide working with home automation and in Denmark some of the bigger players are Z-Wave, X10, Insteon, Zigbee and IHC. Z-wave and Zigbee are both wireless communication protocols utilizing mesh networks for exchanging data. X10 on the other hand is a communication standard that uses the already built in power lines in the house to communicate through. Insteon is a mix between the mesh networks and x10 since both communication forms are used. IHC requires specially installed wires in the house to communicate through.

Home automation is very relevant in these times where saving energy is

of great importance. In an automated house lots of energy can be saved through lighting- and heating- control or programmed scenarios for when people are present in the house and much more. Home automation also offers comfort and convenient control of the house. pre-heating of a house is fx highly usable during the winter, where you could completely turn of all heating in the house when no residents are present and then start pre-heating the house as soon as you leave work. Home automation also offers an array of alarm systems available that serve to monitor fx fire/smoke, intrusion or leakage.

Comparing the IHC solution from Schneider Electrics to some of the other solutions we see that it is rather complex compared to fx Z-wave or x10. The IHC solution takes much longer time to configure but has a wider span of possibilities. The Z-wave and x10 solutions are both rather simple-to-use autonomous systems but neither currently has an official API to communicate with the device through. Therefore a completely new command library and communication protocol would have to be built in order to use these systems. The IHC system comes with the OpenAPI Beta developed by Schneider themselves to communicate with their IHC saving us this part of the programming. In the end we didn't have a choice in what technology to use since Schneider Electrics was a sponsor already picked out for us when we started the project but it could have been exciting to create your own API for Z-wave.

2.1.1 LK Schneider background

Since 1993 when the first generation of IHC systems were marketed, through 2001 when a new generation was released and finally in 2006 when the currently working generation was developed, the IHC system has always been an autonomously controlled unit that could only be interacted with through buttons and sensors. It allowed the user to program it and then only use it via the buttons installed in the house or the sensors used. In January 2012 LK Schneider created the OpenAPI and opened it up for 3rd party users. This API allows the use of 3rd party programs to interact with the IHC using an ethernet or USB connection. This means that the IHC is no longer an autonomous system and that one can have complete control over it through a 3rd party program. In our case this

means we can design our own message bus and connect it to the IHC, provided that the proper middleware between the bus and the IHC controller is created. It is this exact middleware that is the biggest part of my project.

2.2 Overall system Architecture

As mentioned in the introduction the overall goal of our system is to integrate many subsystems onto a bus allowing the different subsystems to communicate with each other not regarding which company or technology the specific subsystem uses. We want one of the subsystems to be the intelligent subsystem (the CCU). This subsystem must be informed of almost all activity in the house because it needs to be able make intelligent decisions and suggestions for the user. To facilitate this purpose we decided to make the communication protocol event-based. This means that a subsystem subscribes to a certain event from a specific subsystem if information from this subsystem is desired. Subsystems publish events when actions occur inside them and the subsystems subscribed to this event will receive it and can do whatever they want to do with the information contained in the event. All this communication is done via a custom-made message bus developed in our group.

In order to enable proper and easy communication between the subsystems we have created a common interface that all subsystems need to be integrated with.

One of the subsystems on the bus is the IHC system which this project revolves around. The IHC system will need to be integrated with the common event interface that is created. Along with this interface there will be a common data model with some strong typing in it that will also need to be integrated with. To do this a software layer between the bus and the IHC is needed that facilitates communication from the bus to the IHC.

2.2.1 Quality parameters of IHC integration layer

The quality of the IHC integration layer can be measured by some parameters that will be described here.

Speed of the IHC system. This is the time between when a button in the house is pushed and the action linked to that button is executed and the bus is informed of the action. Alternatively it is the time between when a request is sent from the bus to the middleware and the request is carried out on the IHC and the bus is informed of the action.

Features of the IHC system. This is the amount of resources and features supported by the system. It is a measurement of how many of the desired features for the system that has been achieved.

Extensibility and simplicity of the IHC system. This is a measurement of how easy the system is to maintain and expand. Expansion of a system is made easier for the programmer if code is written generically and with easily recognized patterns.

Robustness of the IHC system. This is how smooth the system runs. Ideally there shouldn't be any run time errors since this will immediately be felt by the user.

2.3 IHC Hardware and firmware

Here i will give an analysis of how the IHC hardware and firmware works in order to get an overview of what possibilities and limitations we have.

Before the mechanics of the hardware and software is described i will describe the input/output principle of the IHC system. The IHC hardware

and software revolves around input and output resources. These serve different purposes and are used in different ways. Generally output resources are the resources that are controlled and input resources provide information which is used to control the output resources. A nice example is a motion sensor and a lamp. The motion sensor is the input resource and the lamp is the output resource. The motion sensor provides information to the system about whether or not motion has been detected and with this information the system decides whether to turn on or off the lamp.

2.3.1 Hardware

The IHC home control system from LK Schneider Electric is the system we have utilized for controlling many of the elements in Team DTU's house in Solar Decathlon. The hardware of the IHC consists of three elements: output-, input- and control modules. All output resources in the house are connected to the output module via wires or wirelessly. Likewise the input resources are connected to the input module. Both of these modules are connected to the control module that contains the logical mapping that controls the house (this mapping is described in the next section). All resources have a control state. Either the resource is a toggle resource that can be toggled on or off or else it is a set-point resource with a value from 1-100. When an input resources changes its state the control module is informed of it and can make decisions on what to do with the output resources.

2.3.2 Firmware

The control module of the IHC is the module that contains the "programming" of the IHC (it is not real programming, more like advanced mapping between input and output resources). It can only be mapped using Schneiders own firmware called LK Visual which gives the programmer a variety of options.

LK Visual operates with resources on the left side of the screen and

function blocks on the right side of the screen. In order to connect any resource with another resource, you need at least one function block to dictate which kind of control there should be between the resources. This means you can have a lamp and a PIR-sensor connected in the system with one kind of control, and another lamp and PIR-sensor connected with a completely different kind of control because different function blocks are being used between them. Resources must always be connected to other resources through function blocks, however a function block can be connected to another function block if any properties from another function block is desired.

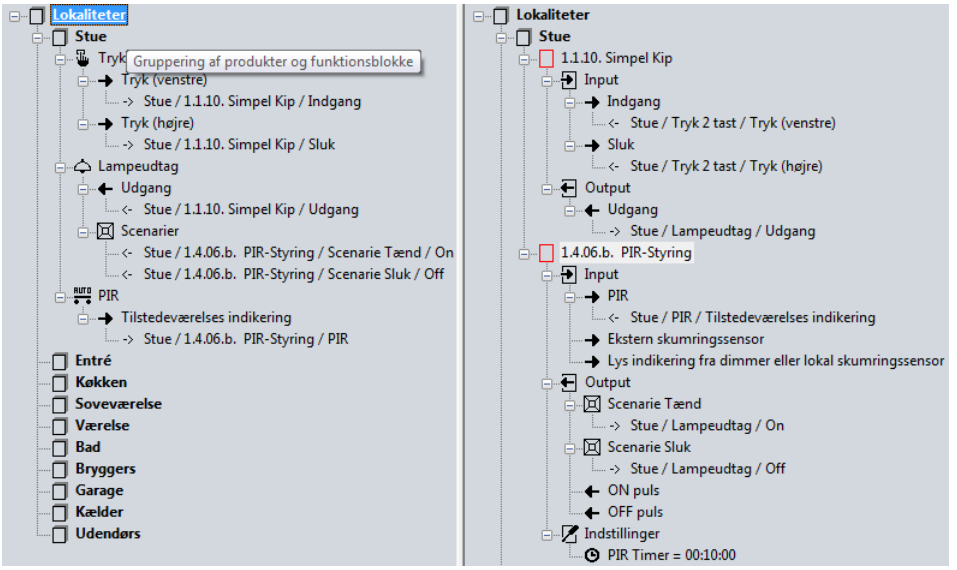

Figure 2.1: LK Visual programming of IHC Control Module

Figure 2.1 on page 12 shows a little display configuration with a lamp, a button and a PIR-sensor is shown. The left part of the button is mapped to one of the Input ports on the function block "simpel kip" called "indgang" and the right part of the button is mapped to the other input port on "simpel kip". Both of these input ports affect the output port of the lamp turning it off or on. In conjunction with this a PIR sensor is connected to the lamp, turning it on if it detects any movement, and off if no movement has been detected for a configurable amount of time. So this means that the lamp is both controlled by a physical button but also by a PIR sensor.

If you look closer at the input ports on the function block called "PIR-Styring" you will notice an input port called "Ekstern skumringssensor". This is an input port on the "PIR-Styring" function block, for a twilight sensor that is placed outside and detects if it is bright or dark. This input could be used as an additional input condition to decide if it is necessary to turn on the light when motion is detected if it is already bright outside.

When the mapping is done the it can be saved as a project file and transferred to the IHC control module. The project file will contain information regarding all resources, their placement and how they are controlled.

2.3.3 IHC OpenAPI

The IHC OpenAPI allows a 3rd party application to interact with the IHC through a webservice containing an array of webservice methods available. I will not give a thorough explanation of the API but only introduce the most important methods.

The OpenAPI is built around the LK Visual programming tool. This means that it is centered around resources and the changing of the state of resources. Each input and outport port on each input/output resource has its own unique ID. By calling the API method **Setvalues** with an argument being the desired state, you can instruct the IHC to change the state of a resource to the argument. If you want to be informed about the change of the state of a resource you can subscribe to that resource using the **Enable subscription** method. When you are subscribed to a resource you call the **Waitforevents** method which places a "hook" on the IHC returning when an event occurs on the IHC. An event occurs when any resource changes its state. This means that resource-related interaction with the IHC through the OpenAPI will be event-based through asynchronous calls. How this API is used with the IHC is described extensively in the design section 3.2.

The OpenAPI also has some additional methods that provides valuable information. In order to extract the projectfile on the IHC Control module (Created in LK Visual) you call **getprojectsegment** any number of times depending on how big your projectfile is which is further described

in the implementation section 4. Finally the OpenAPI also contains methods for creating a session/connection with the IHC using TCP/IP.

2.3.4 IHC Enabled systems

Here i will give an overview of a typical setup in an IHC controlled house.

Somewhere in the house the IHC hardware modules will be placed (the input-, output- and control modules). Throughout the house there will be a number of input and output resources connected to the modules. Input resources typically include button pads, motion-, lux-, smoke, door and twilight sensors. These provide input information via the input module to the control module. The control module controls the output resources based on this information. Output resources typically include lamps (dimmable and non-dimmable) and special power switches that can be turned off to save standby power.

Either the user of the house has "programmed" the IHC using Schneiders firmware or the electrician does it when he installs it. When the house is programmed the file is transferred to the control module and now the house is controlled as it is programmed. This means that one button might turn on kitchen lighting, another one might dim up/down on a bedroom light and another one might turn off all lights in the house and set the alarm.

What is described above is a standard usage of an IHC system. When the system is programmed you can't control the house in other ways than the way it is mapped. There is no flexibility in this system. What we in the control systems group is doing is adding a software layer on top of this IHC system that grants a bus complete control over the IHC system. It allows the bus to "push the buttons in the house" so to speak. But not only does it allow the bus to push the buttons in the house, it grants independent control over all output resources in the house. What is meant by independent is that the bus is not limited to pushing the button, it can just directly by-pass the button and select the exact output resources that it wants to control meaning that it is not limited by what the button is programmed to do. Additionally the bus is also fed all information

from the input resources. This information can then be used by other 3rd party apps which another team member from the software control group is creating.

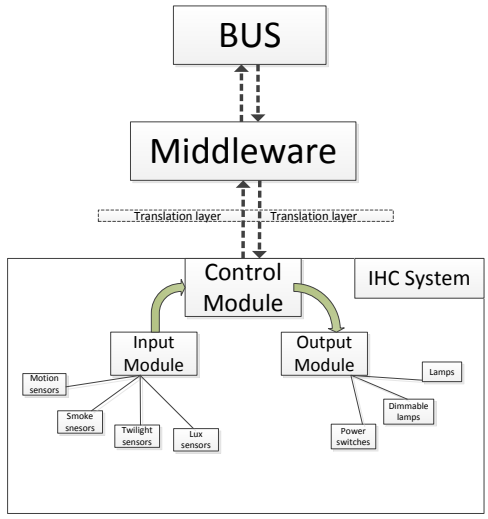

Figure 2.2: Diagram of IHC system with middleware layer built on top of it

So as can be seen on figure 2.2 on page 15 our system is not preventing or doing any changes to how the IHC system works. You can still have an "old-fashioned" IHC system installed in your house where only the physical buttons and sensors are used to control the house. Our system comes on top of the regular system allowing 3rd party programs to interact with the house, via fx an app, which will also give the user suggestions for better power usage, weather forecast etc. based on information from the IHC and other subsystems.

2.4 Problem definition

Here the two main challenges will be described and a short summation is given.

2.4.1 Middleware Layer

The middleware should essentially be a wrapper over the IHC control module providing a common interface to all parties on the bus. It should allow toggling and adjusting of all output resources (both on/off resources and set-point resources), so that lights can be turned on from the bus, power switches can be switched off and on and the alarm can armed or disarmed. It should inform the bus whenever any input resource changes its state e.g. a motion sensor is triggered. Note that everything is possible with the OpenAPI, even things that intuitively should not be possible, such as changing the state of the twilight sensor so it will inform the house that its dark outside even though its bright daylight. Or changing the state of the PIR sensor so the house will think motion is detected. Therefore its important to encapsulate invalid actions from the bus so other programmers wont think fx that they need to manually control the state of sensors.

The middleware layer needs to communicate with the resources on the IHC control module. This requires a translation layer that can fetch the information from the projectfile and translate it so the middleware knows what resources are placed where in the house and how they are controlled.

2.4.2 Firmware programming

The house needs to be modelled and designed with Schneider's firmware. The resources should be intelligently placed in the house which means that the house needs to be divided into rooms in a smart way. The resource placement and room division should not only make it more comfortable to live in the house but also to a large degree be systematically placed to allow for future development of the house, since FOLD is purely built as a display house.

The resources programmed in LK Visual should be done so in a way that allows for easy translation.

2.4.3 Work tasks summarized

Summing up the work tasks will consist of:

- Connecting the physical components in the IHC through wires or wirelessly (an electrician will help with this part).
- Build the LK visual program with all the resources in it and map the correct outputs to the correct inputs via function blocks etc as described in IHC Firmware section.
- Create a translation layer in the middleware that can translate the information from the project file on the IHC control module.
- Create the middleware layer integrating it with the bus.

2.4.4 What is out of scope

This report will not focus on how the bus works or how any of the other subsystems besides IHC is connected to the bus. I am assuming that there is a functioning, fast, reliable and secure bus to send messages through. There will be no focus on the other entities connected to the bus e.g. the CCU-system, the App-Service, PLC etc. since these simply subscribe to/send request through the bus to the IHC system if they want to interact with it. Security will not be discussed in this report because we in the group made the choice to trust the subsystems connected to the bus. If a subsystems is connected to the bus we assume that it is programmed and integrated by a trusted developer. In order to make this assumption we have instead put all security in the bus meaning that communication between subsystems on the bus is secure.

2.5 Resource-type List

The comprehensive list of resource types connected to the IHC are the backbone of the entire IHC system and needs to be considered when

programming the control module in LK Visual and when designing the middleware. It is both an overview of what possibilities you have when programming the system but also the limitations to the system since this is all the resources there is. Here is a list of all the physical resource-types available that is connected to the IHC with descriptions of what their properties are.

2.5.1 Input resources

PIR-Sensor Passive Infrared Sensor is a motion sensor typically placed near the ceiling in a house pointing downwards. It is mainly used for turning off the lights in a room if no motion in it has been detected for a certain amount of time.

Magnet-Sensor is a small pair of magnets typically installed in windows and doors. When the door/window is closed the magnets will face each other creating attraction and generating a current. When this current is broken it means that the door/window is open. It is mainly used with the alarm system to detect intrusion but is also used to ensure all doors/windows are closed when leaving the house.

Lux-Sensor is a sensor that detects the light level at a certain spot. It can be placed anywhere one wants to measure the light level at. As stated in the Solar Decathlon Competition rules[1] we must uphold a minimum of 500 lux at a workstation in the house. A lux sensor will therefore be placed near the workstation to cooperate with a dimmable lamp.

Smoke-Sensor is typically placed near the ceiling in the kitchen of the house and detects smoke/fire which is used to trigger the alarm system.

[1] http://www.scribd.com/doc/47730341/SDE-2012-RULES-V-1-0 Rule 19, Page 52 of the document

Twilight-Sensor is a kind of lux sensor but it is placed outside and detects whether it's dark or bright. It is used to decide whether the PIR sensors inside should turn on lights or not when motion is detected.

2.5.2 Output resources

Power Outlet is a plug in the wall from which you draw power. Approximately half of all the power outlets in the house can be controlled to switch on/off. This means that you could have many appliances that use standby power plugged in to these power outlets. Then when you leave the house all these power outlets are switched off so no standby power is consumed while nobody is in the house.

Dimmable Lamp outlet is a lamp outlet that has a setpoint value between 1-100 with 100 being the max load. This value determines how much light the dimmable lamp outlet will produce.

Lamp outlet is a standard lamp outlet where the state is on/off.

Sound generator is a small device for creating loud alarm sounds. It can either produce an 80 dB warning sound or a 102 dB sound. The 102 dB sound is used in the case of intrusion or smoke.

2.6 Requirements

The system should be intelligent in the way that it intelligently turns off and regulates lights in the house. It should offer both physical control of the house through buttons in the house but also facilitate all functions for the bus. The system should be able to communicate properly with the bus. This means giving users on the bus the ability to change the state of all output resources e.g. switching lamps on/off, switching power outlets on/off and activating the sound generator. Likewise it should inform

the bus whenever any resource changes it state. The system should also provide an alarm system that is both controllable through the bus but also through a code pad in the house. The system should also inform the bus when the alarm system has been activated. The resources necessary for the alarm subsystem might not be present at the moment and therefore we might need some additional virtual resources (More on this in the design chapter 3.1.4).

2.6.1 System users

Here the users of the system will be briefly described

2.6.1.1 End-user

The end user is a user that lives in the house. He/she is in inhabitant in the house and has no knowledge of the mechanics behind the scenes but solely sees the functions as they appear in the house. An end-user will care about how many features the system includes. He will want the system to be as reliable and simple as possible so he can get way with as little as possible maintenance.

2.6.1.2 Programmer

A programmer that is going to maintain, expand on, or use the IHC middleware system will want it to be coded as simple as possible with easily recognized patterns. He will want the system to be transparent regarding information so nothing is hidden.

2.6.2 Functional requirements

Functional requirements from the end-user.

1. The system should automatically turn on lights in rooms where this is beneficial.
2. The system should offer lighting and dimmable lighting to the users where this is appropriate.
3. There should be alarms in the house, both for intrusion and also for fire/smoke.
4. Some power switches (ca 50%) in the house should be controllable through the bus.
5. The system should be able to uphold a certain lux level at a certain spot for a longer time.
6. The user should never feel that the system has more power than them, therefore all functions that are available to the bus should also be able to be manually overridden.

Functional requirements from the programmer.

1. The bus must be able to control all relevant output resources in the house.
2. At least all functions that are offered to the user in the house should also be offered to the bus.
3. When an input or output resource changes its state the bus should be informed.

2.6.3 Non-Functional requirements

1. The system should be robust so very little or no run-time errors occur since this will be felt immediately by the user in the house.
2. When a user turns on a light either through a physical button in the house or through the bus, the response time shouldn't be more than 1 second.

3. It should be easy for future programmers to add additional resources to the system and maintain it, meaning that a certain level of extensibility and simplicity is desired.

Chapter 3

Design

In this chapter i will walk through how the house is designed in terms of room division and resource placement in LK Visual. I will also discuss the placement of the resources, what their functions are and why they are placed where they are. This mainly involves the programming of the IHC through LK Visual.

After this the design of the middleware layer will be described in detail including the event system between the middleware and the IHC and the event system between the middleware and the bus.

3.1 Room design in LK Visual

In order to make the house smart and innovative when using an IHC system, a good room design is important. It will not only ease the task of mapping input resources to output resources but it will also give a nice overview of the entire house making the maintenance of the system

easier. What is meant by a room design is to divide the house up into areas that are treated as virtual rooms. The big issue often is how many virtual rooms it is lucrative to divide the house into.

Designing the house one has to find a balance between conceptuality and practicality. In a practical design emphasis is put on the actual living in the house. Only the necessary buttons will be placed in the house and all resources will be placed where they are actually going to be used if people lived in the house. This means that a practical design often will be a lot cheaper than a conceptual design since only the necessary resources will be used. The way the resources are used will also be as simple as possible in a practical design since it is made for living in the house. A practical design can however lead to limited extension possibilities of the system and might be hard to maintain since no systematical approach is used.

In a conceptual design resources will be above all be placed systematically in the house. If there is a button pad in one room that controls lighting in a certain way then there will also be a button in other rooms controlling lights in the same way. A conceptual design tries to use the same resource placement pattern and the same control patterns for those resources. This might lead to some more expensive solutions since excessive resources might be necessary to uphold the patterns. On the other hand extending a system conceptually designed is easy once the designer recognizes the patterns used in the house for the resource placement and for the control of the resources.

3.1.1 Conceptual design vs. Practical design

The biggest part of the process of programming the house in LK Visual was to decide a way to divide the house into sections. Which is better, a practical or conceptual approach ? The advantages of a practical approach is that it makes the actual living in the house easier. There will be fewer buttons and they will have a more intuitive control pattern. There will be no redundant or unnecessary lights in the house. A conceptual approach allows for greater design possibilities. It creates advanced and somewhat harder to understand control patterns but with a greater potential for expansion of the house. The conceptual approach can feel

a bit like "overkill" in some situations. This especially becomes true if the bedroom is placed right next to the work area in the same room 1,5 meters from each other. Then turning on the lights in the bedroom will also illuminate the work area room thereby devaluating the concept of "turning on the lights in the bedroom" when this clearly also illuminates the work area. However in a standard house the bedroom and the work area would normally be separated, at least enough so the lights wont effect each other, and therefore separating the bedroom and work area into two rooms makes sense conceptually.

In our design of the house we put emphasis on the conceptual side because with this project we think it's important that the design choices regarding the house can be used in a wider format. fx if the house was to be expanded or some of the rooms made bigger, then a conceptual design is more useful than a practical one. Also if the design pattern were to be used in another iteration of DTU's solar decathlon house. Another reason for emphasizing conceptuality is that this house in essence is a display house. It is built to show people what can be done with technology today, not built for practicality. Therefore we think a conceptual design is more important than a practical one.

3.1.2 Room division

Based on a conceptual design pattern we want to divide the house into a number of virtual rooms.

Figure 3.1 on page 26 shows a drawing of the house seen from above with some of the furniture in it. Our group did not have any say in where the furniture was placed so the room division had to be done based on where the furniture was placed, not the other way around. Also the names and locations of some of the rooms on the drawing were already decided before our group started working on the project.

If you study the drawing you will notice that the house essentially can be divided into two subsections. One section is the big living space with all the furniture in it, and the other section contains the four rooms behind the wall partition. The section behind the wall are called the

Figure 3.1: Ground floor drawing with furniture

Technical Core rooms and the big room we will call the living space (this is later sub-divided into more virtual rooms). As mentioned earlier when dividing the house into sections or rooms one has to consider how practical vs. conceptual you want to design the house. This is often expressed in how detailed the house is divided into virtual rooms. The more room-divison the more conceptually designed the house is and also the more systematically designed it is the more conceptually designed it is.

Regarding the Technical Core rooms we have the choice between merging some of them to make fewer virtual rooms, further subdividing them, or keeping them the way they are. Merging two of the virtual rooms makes sense if we want to treat these rooms as a single room whilst further subdividing them makes sense if the functionality in the room affects too large a part of the room and hence needs to be focused on a smaller part of that room.

As the drawing shows the technical room is separated from the storage room and the toilet by walls and likewise the toilet is separated from the bath area. Therefore it does not make any sense to for these rooms to share motion sensors or lighting since motion cant be detected through walls and lights cant pass through walls. Therefore merging them makes no sense. The rooms themselves are not bigger than approximately 2x2 meters therefore subdividing them further also doesn't make sense. This leads to the first natural step which is to leave the rooms as they are and create four virtual rooms in LK Visual corresponding to these rooms. This means a storage room, a technical room, a toilet and a bath room is created.

Regarding the living space the balance between conceptuality and practicality again comes in. The living space actually consists of one big room, but the room serves a number of different purposes. In the upper left corner of the living space the kitchen in the house will be placed. On the right side of the kitchen there is a dining table and below the dining table is a work area followed by a bed. Next to the bed is an open space and above the open space is a cushion arrangement with a TV on the wall next to the toilet. The living space actually serves many of the tasks that in a normal house would have been separated to rooms for themselves. As mentioned earlier emphasis in this house is put on the conceptual side,

we want the work we do on the house to be reusable and extendable. Therefore we have divided the living space into six virtual rooms even though it is one big room which is seen on figure 3.2 page 29.

This division has its advantages and its disadvantages. It provides for a very nice division where all the rooms that one would find in a normal house is present. On the other hand some apparent inconveniences arise, which include that when you turn on the light at the dining table it may also illuminate the rest of the entire living space which means that the room division is rendered unnecessary. The advantage of this division is that the IHC integration system that we have developed more easily can be transferred to other houses since many of the same rooms with the same functional needs will be present. It also allows us from now on to consider each rooms needs individually without considering the other rooms. As mentioned earlier the rooms might affect each other since there are no walls between them, but this is ignored to preserve the conceptuality.

3.1.3 Resource placement

Here i will describe what resources will be placed in each room in the house and what their function is, which will result in a list of all resources used by the IHC. It has not been easy compiling this list since it is constantly changing when new ideas and demands arise from the architects and construction engineers.

Common for all rooms in the house except the bath area, storage room and outdoors is that they all have power outlets. This is because nowadays most people have a lot of devices that use power and you want to have access to power no matter where in the house you are. All rooms in the house also have lighting in them though the kind of light differs which will be further elaborated in the next section. In order for the user to not feel dependant on controlling the house only through software, there are buttons in most rooms to allow for manual control of the resources in the house.

Figure 3.2: Ground floor drawing with furniture and room division

3.1.3.1 Living space

As discussed earlier the living space is one big room which means that we have to address some issues. We would like to use motion sensors as often as possible in the house since they can turn on lighting when people aren't present saving energy. However if motion sensors are used in the living space to turn on the lights in the different rooms it will be almost impossible to only turn on lighting in one room since the sensitivity of the sensors are too great to be limited to certain areas of the same room (This would require actual walls between the rooms). Furthermore it would also be a nuisance for the inhabitant when the lights keep on turning off and he has to wave his hands to turn it on again.

Therefore we have decided not to use any motion sensors to control lighting in the living space since the sensitivity of the sensors are too great and cannot be limited to one section of the living space. The use of motion sensors in the living space would destroy the division of the rooms and be a nuisance for the inhabitant.

Regarding lights in the living space dimmable and non dimmable lamps were both considered and weighed for a long time. The advantages of dimmable lights are that it saves power when it is turned on since the power consumed is reduced when the light level is reduced. Further more it is also a nice option for the inhabitant that he/she can dim the lighting to his/her desired level in every single of the six rooms in the living space. The cons are that dimmable lighting is a tiny bit harder to program and if the dimmable lights is used very rarely it actually consumes more power than standard lights since a tiny portion of standby power is consumed by it. Considering these facts dimmable lights were chosen over standard lights for the main reason that inhabitants will spend a great amount of time in the living space which means that the light will be turned on a lot of the time making dimmable lights the more energy efficient solution.

To control the functions in the living space each of the six rooms has its own button-pad placed on the wall to control the dimmable lights in that room along with possible extra features. This button-pad can be perceived as an interface for the user over what options he has with that specific room. The same functions and much more will be provided

through the bus to the user. The button pads in each room will offer the user the option of turning on/off lights in that room, dim up/down the lights and in some rooms activate certain scenarios.

Here is an overview of the resources in the six rooms in the living space, with a description of the functions of the specific rooms. A total overview of all resources in the house can be seen on figure 3.3 page 40. It should be noted that some resources in the living space are not connected to any room but is instead placed in another virtual room called "Common" which contains shared functionality for all the six rooms in the living space. This is described further in section 3.1.3.3.

Dining Room

The dining room is where the inhabitants eat, entertain guests etc. It is a room where inhabitants will often sit for a longer period of time meaning that dimmable lights will save energy. Dimmable lights will also be great for dinner parties when guests are entertained which is actually relevant since a dinner party must be hosted for the official Solar Decathlon Jury [1]. The resources required are therefore some dimmable lights along with power outlets and a button pad to control the lights i.e turn them on/off and dim up/down.

- Dimmable lamp outlet is placed directly above dining table
- Power outlet is placed at ground level near eastern wall
- Button-pad is placed 110 cm above floor level

Work Area

The work area serves as an office for the inhabitants. This requires power outlets for laptops and additional electric equipment. A dimmable lamp is required to set just the right lighting level along with a lux sensor. This is to be able to maintain a light level of 500 lux at all times when the light in the work area is turned on which is one of the requirements from the official Solar Decathlon Jury [2]. Along with this a button pad is needed

[1] http://www.scribd.com/doc/47730341/SDE-2012-RULES-V-1-0 Rule 42, Page 97 of the document

to control lights in the work area.

- Dimmable lamp outlet is placed directly above work desk
- 2x Power outlet is placed at ground level near eastern wall
- Button-pad is placed 110 cm above floor level at eastern wall
- Lux sensor is placed at a yet unknown location in work area

Bedroom

The bedroom is supposed to accommodate two people, this means that power outlets in both sides of the bed is required. Dimmable ceiling light is required in the bedroom along with a night stand light in each side of the bed. We want all lights in the bedroom to be controllable from both sides of the bed. Therefore button pads are placed at both sides of the bed. Both of these button pads contain a toilet scenario button that creates a path of light from the bed to the toilet.

- Dimmable lamp outlet is placed directly above bed
- 2x Power outlet is placed at ground level at each side of bed
- 2x Button-pad is placed 70 cm above floor level at each side of bed

Kitchen

The kitchen is where cooking and cleaning is done along with various other things. Like most of the other rooms we want to provide lights, power and a button pad for the user in this room. In addition to this we want a smoke sensor placed in this room since this is where a fire is most likely to break out. The smoke sensor will trigger a sound generator creating a load sound of 102 dB. The button pad controls the light in the kitchen but can also disable the smoke alarm for a short period because you may know there will be smoke the next couple of minutes. Likewise there is also a button to stop the sound generator from making noise when the smoke generator has been triggered.

[2] http://www.scribd.com/doc/47730341/SDE-2012-RULES-V-1-0 Rule 19, Page 52 of the document

- Dimmable lamp outlet is placed directly above kitchen table
- 2x Power outlet is placed just above kitchen counter in kitchen
- Button-pad is placed 110 cm above floor at yet unknown location in kitchen
- Smoke sensor is placed at ceiling in the kitchen

Entertainment

The entertainment area is for watching TV, playing games and general relaxing. Inhabitants will often spend a lot of time in this area and therefore dimmable lights are attractive. Also as a nice feature to when you are watching TV. Along with this some power outlets are placed here to support various Console/TV equipment and a button pad is installed to control the lights.

- Dimmable lamp outlet is placed directly above square-shaped furniture
- 2x Power outlet is placed at ground level near wall partition
- Button-pad is placed 110 cm above floor level at wall partition

Open Space

The open space can be treated a bit like the entry hallway of a house. This room does not have any special requirements except for lighting and a button pad to control it. A power outlet is added to the room for convenience.

- Dimmable lamp outlet is placed directly above open space area
- Power outlet is placed at ground level near wall partition
- Button-pad is placed 110 cm above floor level at wall partition

The nice thing about the resource layout of these rooms in the living space is that all resource types that are usually desired in the particular rooms are present, meaning that if one wanted to expand the rooms, more of the same resources could simply be added. If a new resource type is

wanted in a room that doesn't already contain it this is no problem either since this can simply be added and configured in LK Visual. All these resources in the living space can be seen in appendix 2 and 3 where the entire programming of the house in LK Visual is shown.

3.1.3.2 Technical Core Rooms

As mentioned earlier the Technical Core Rooms are divided into four virtual rooms each being separated by walls. This means that motion sensors is an option since motion in one room won't trigger from motion in the other rooms. The disadvantages of motion sensors are if they either turn on the lights when it isn't supposed to turn on thereby consuming unnecessary power and possibly annoying the inhabitant. It is also an annoyance if lights are turned off because the inhabitant is simply still without moving.

Considering these facts we decided that lights controlled by motion sensors was still quite important, since the Technical Core Rooms are rooms that one typically enters for a short time and leaves again many times during the day. The motion sensors will help turn off the lights for you so you don't have to remember thereby saving a lot of energy when the inhabitants accidentally forget to turn off the lights. You are also saved the annoyance of pressing a button to turn on the lights every time you enter the rooms.

As for whether or not to use dimmable or non dimmable lights the same cons/pros as mentioned in the living space section 3.1.3.1 applies. Non-dimmable lights were chosen since people often won't spend a lot of time in these rooms making the stand by power from dimmable lights a factor. It was however considered to provide dimmable lights for the bath area and toilet since this would be comfortable if the inhabitant were visiting the bath room at night but in order to save more energy this wasn't chosen.

I will whenever a motion sensor is used comment on the motion timer. If the motion timer is set to 180 seconds this means that lights are turned off if no motion is detected within 180 seconds. The motion timer counts

down all the time from its setpoint to 0, and when motion is detected it is reset to its setpoint again. All motion timers can be changed by the user through LK Visual.

Storage Room

The storage room is where home appliances are located. This includes a washing and drying machine, a refrigerator and storage space. Inhabitants will mostly come here for short periods of time and then leave again. Therefore lighting here will be motion sensor controlled and non-dimmable. The motion timer is set to 60 sec because inhabitants will visit this room very frequently for the refrigerator and leave again. The storage room is placed right next to a glass facade through which natural outside light will shine. Therefore the motion controlled lights are connected to a twilight sensor that prevents the light being turned on in the storage room when it is bright daylight outside (it can still be switched on manually with a button pad).

- Lamp outlet is placed at ceiling in middle of room
- Button-pad is placed 110 cm above floor level at north end of wall partition
- PIR-Sensor is placed at ceiling in middle storage room

Technical Room

The technical room is a very important part of the house. In here a lot of electrical hardware will be placed. Some of this includes pipes and valves for heating/cooling of the house controlled by PLC and Uponor hardware, IHC hardware, Windowmaster system controlling windows and a server hosting our software groups entire software system. Because the technical room is accessed from outside, magnet sensors are placed in the door to this room that triggers the alarm system when the door is opened. This is to prevent that a robber simply breaks down the door and destroys the security system before entering the house through the main door. This is a room that (Hopefully) is visited rarely since this means things are running as they should. Lights here are therefore motion sensor controlled but can also be activated with a button pad. The motion timer is set to 10 min, since when this room is visited it will often be for longer periods

of time. Note that numerous power outlets are present in this room but the one we refer to here is a free power outlet.

- Lamp outlet is placed at ceiling in middle of room
- Button-pad is placed 110 cm above floor level right side of entrance
- PIR-Sensor is placed at ceiling in middle technical room
- Magnet sensor is placed inside door
- Power outlet is placed at unknown location in technical room

Toilet

The toilet is used for obvious reasons. It is visited very often during a day and often at short intervals. This means that motion sensors are very well suited to control the lights for the reasons mentioned earlier in this section. Again a button pad is also provided so manual control is possible. The motion timer is set to 7 min since it can be very annoying if the light is turning off all the time and you have to wave your hands. The lights in here are not connected to the twilight sensor since only a small portion of the daylight reaches this room through the glass facade.

- Lamp outlet is placed at ceiling in middle of room
- Button-pad is placed 110 cm above floor level at right side of entrance
- PIR-Sensor is placed at ceiling at right side of entrance
- Power outlet is placed at 110 cm above floor level at unknown location
- Power outlet is placed at unknown location in technical room

Bath Area The bath area is used for obvious reasons. Because of the architecture of the house inhabitants have to pass through the bath area when visiting the toilet. Therefore inhabitants will visit the bath area at least as often as the toilet making it suitable for motion sensor controlled lights. The problem is that inhabitants typically use the bath area for longer periods than the toilet, leaving us with the choice of how long the motion timer should be. If the same motion timer as the toilet is

used inhabitants will be annoyed that lights may shut off suddenly while taking a bath. If a long motion timer is used unnecessary power might be consumed when merely passing through the bath area to the toilet. We decided that it was too annoying for the inhabitant if lights suddenly turned off while bathing and therefore set the motion timer to 15 min. Lights in here are connected to the twilight sensor since a large portion of the daylight reaches this room making lighting unnecessary during broad daylight.

- Lamp outlet is placed at ceiling in middle of room
- Button-pad is placed 110 cm above floor level end of wall partition
- PIR-Sensor is placed at wall partition near ceiling

3.1.3.3 Other "Rooms"

To make the programming of the IHC easier, some resources are not assigned to any room but instead to a "room" called Common. This is essentially resources that are located in the living space but doesn't belong to any specific room.

We wanted to make the house comfortable providing intelligent ways of controlling it fx by having certain pre-programmed setups that could be activated. This can either be done through the bus from the app or from a button directly in the house or both options. Since our goal is to provide as much functionality as possible to the bus while also making sure that the user doesn't feel that an app is an absolute requirement to control the house we decided to do both things. Therefore we created two button pads in the living space that contain six pre-programmed scenarios (these can be changed by the user with LK Visual) that activate a set of resources. These scenarios include a cooking-, TV-, welcome- and leave house scenarios just to mention some of them. The cooking scenario fx turns on the dimmable lights in the kitchen at 100 %, dining room lights at 35 % while turning off all other lights. The welcome scenario will create a nice lighting, with lights spread out through the living space at 65 %.

Since saving energy is a major focus point with this house we wanted to

offer functions for the user that save power while the house is still easy to use. Therefore one of the button on the the 6-scenario button pad is dedicated to this. When this button is pushed for more than 0.7 seconds all lights in the house will turn off. This is to save the inhabitant the hassle of having to manually turn off all lights when he wants to. This makes it very easy to save energy on sunny day or whenever you simply don't need any lights turned on in the house. Furthermore when this button is pushed for more than 5.0 seconds all power switches are turned off, all lights turned off and the alarm system is activated. This function is ideal when the inhabitant is leaving the house since it turns off the IHC controlled part of the house and saves energy.

Common

There are three extra dimmable lamp outlets in the living space. This is created so future customization of the lighting is possible. Also so electricians installing the lights have some freedom to use different outlets if desired. The two button-pads here are the scenario button pads mentioned earlier. Each button pad has six buttons that activates scenarios. The alarm system consists of special motion sensors installed in the living space that only trigger the alarm when the alarm is activated. In conjunction with this magnet sensors are installed in all the doors in the house that trigger if the alarm is activated and a door is somehow opened.

- 3x extra dimmable Lamp outlet is placed at ceiling in uppermiddle, middlemiddle and lower middle of living space
- Button-pad is placed 110 cm above floor level between the 2 doors at north facade
- Button-pad is placed 110 cm above floor level just right of southern door
- 2x Alarm PIR-Sensor is placed at ceiling in corners of living space
- 2x Alarm Code Pad is placed at north and south end of wall partition
- Sound generator is placed at yet unknown location in living space
- Magnet sensors are installed inside all 4 doors (2 in double door, 1 in other doors)

Outdoor

The outdoor area is typically used whenever an inhabitant leaves or arrives home or for grilling and various other activities outside. We want to provide motion controlled lighting for the inhabitants when its dark since this is convenient and nice to have. Therefore lamp outlets and motion sensors are installed outside. We wanted the possibility of creating a light show for by-walkers at the street to see and therefore not just one lamp outlet is used for all the lights but six in total (with only one lamp outlet all the lamps connected to it are either turned on or off simultaneously). As mentioned in the description of some of the other rooms the house utilizes a twilight sensor placed outside to detect when its dark or broad daylight. A sound generator is also installed that makes a loud noise when the alarm system is triggered.

- 6x Lamp outlet is placed at ceiling outdoors, 3 south, 3 north
- 2x PIR-Sensor is placed at ceiling outdoors, 1 north, 1 south
- Sound Generator is placed outside at unknown location
- Twilight sensor is placed outside at unknown location

Figure 3.3 on page 40 shows a drawing of the entire house and the locations of the resources. This accompanied by the descriptions above should give a complete overview of the exact location of all the resources in the house.

Figure 3.3 on page 40 gives us a complete picture of all resources in the system and where they are going to be placed (some of them only approximately but that is fine). we can program the IHC controller in LK Visual precisely as this is shown. The complete and final projectfile in visual studio can be seen in appendix [3]and the projectfile will also be available on the CD accompanying the report.

[3]Appendix der peger på LK Visual program

Figure 3.3: Ground floor drawing with furniture and room division

3.1.4 Virtual resources

Having the physical resources will get you a long way of programming the IHC controller. However some functionality might not be available unless you introduce some additional virtual resources that does not exist physically but will make it easier for the middleware.

Concerning the alarm system in the house we in the software group wanted a way to activate the alarm from the bus. In a standard IHC installation the alarm is activated through the Code Pad (which can also be done in our house). This means that the Code Pad is created in LK Visual as a resource on the IHC Controller. However this resource can programmatically in LK Visual not be interacted with in a simple way, due to limitations from Schneider. Therefore a work around has been done in which a virtual resource called "MainAlarm" is created as if it was a real physical resource and mapped to the alarm system. This resource will then act as a codepad which is an output resource that when set to `On` activates the alarm system and when set to `Off` deactivates the alarm system which can be done from the bus.

A small extra feature was wanted when we designed the house in LK Visual. We wanted the lights connected to PIR-Sensors to be able to be switched off from the motion sensor it was connected to i.e. the light is no longer automatically turned off after a certain time period. This means that we somehow must prevent that lights are turned off when the motion timer reaches 0. To facilitate this an additional virtual resource connected to all motion sensors were introduced and called "ConstantLight". When this resource is set to `On` the lamp outlet connected to the motion sensor will no longer switch off when the motion sensor no longer detects motion. When the ConstantLight is set to `Off` again the lamp outlet will again react to the motion sensor.

3.2 Middleware

When we in the IMM-group started this project together we decided that we were going to develop a communication system with a message bus in

the center as the communication platform. All communication was going to be through subscribing and publishing on this bus. Therefore this IHC middleware that must integrate into this system. The middleware is split up into two parts. One part is designed to handle all communication between the bus and the IHC through an event based system and the other part is designed to handle the booting of the IHC system. The Layer has been designed to be as simple and segmented as possible in order to facilitate easy bug fixing of the code.

3.2.1 Startup component

The startup component will be described in detail in the implementation section 4, since it is rather simple in terms of what it does and what its responsibilities are, but the implementation of it is interesting. So only a brief overview will be given here.

The startup component consists of two sub components, one being the IHC Connection component and the other the IHC Load Project component. Both are run when the system boots up. The connection component has two responsibilities. Establish the initial HTTP or HTTPS session with the IHC and act as a simple security layer with a user/password combo required.

The Load Project component has four responsibilites:

1. It synchronizes the projectfile in the middleware with the current one on the IHC. If the projectfile has not changed since the last boot up this task is unnecessary, if it has, the current projectfile from the IHC will be loaded into the middleware.
2. It generates the resources in the projectfile programmatically for the middleware.
3. It subscribes to all request types in the common-space that concerns the IHC system.
4. It starts the event thread that polls the IHC for events. This is described further in section 3.2.2.4.

3.2.2 Communication components

The communication component of the middleware consists of four sub-components wherein two of them handle in- and outgoing communication with the IHC and the other two in- and outgoing communication with the bus. This can be seen on figure 3.4 on page 44.

Communication between the bus and the middleware consists of a lists of requests that the middleware is subscribed to. When the middleware receives a request, an action is wanted by the sending subsystem along with a response from the middleware when the given resource has changed its state. This response will be given by publishing an event on the bus when the resource has changed its state.

Communication between the middleware and the IHC is done by having the middleware subscribe to all the resources on the IHC it is interested in. When the middleware wants to change the state of a resource, which it may want when a request comes from the bus, the middleware sends a request to the IHC about changing the state of some resources. When the resources have changed their state the IHC responds by publishing an event to the middleware about the new state of the resource.

3.2.2.1 Definitions and types

Before describing the four communication components some definitions and communication types needs an explanation.

Resource state A resource can be in different states and will frequently change its state. However three instances of a resources state exist in the different layers of the IHC integration system. One in the common space, middleware and IHC. The first one is referred to as the "common space resource" and this is the state of the resource known to other subsystems on the bus. The next is called the "internal resource state" which is the one known by the middleware. The last one is the "IHC resource state" which is the actual state of the resource only known by the IHC. Whenever an update to

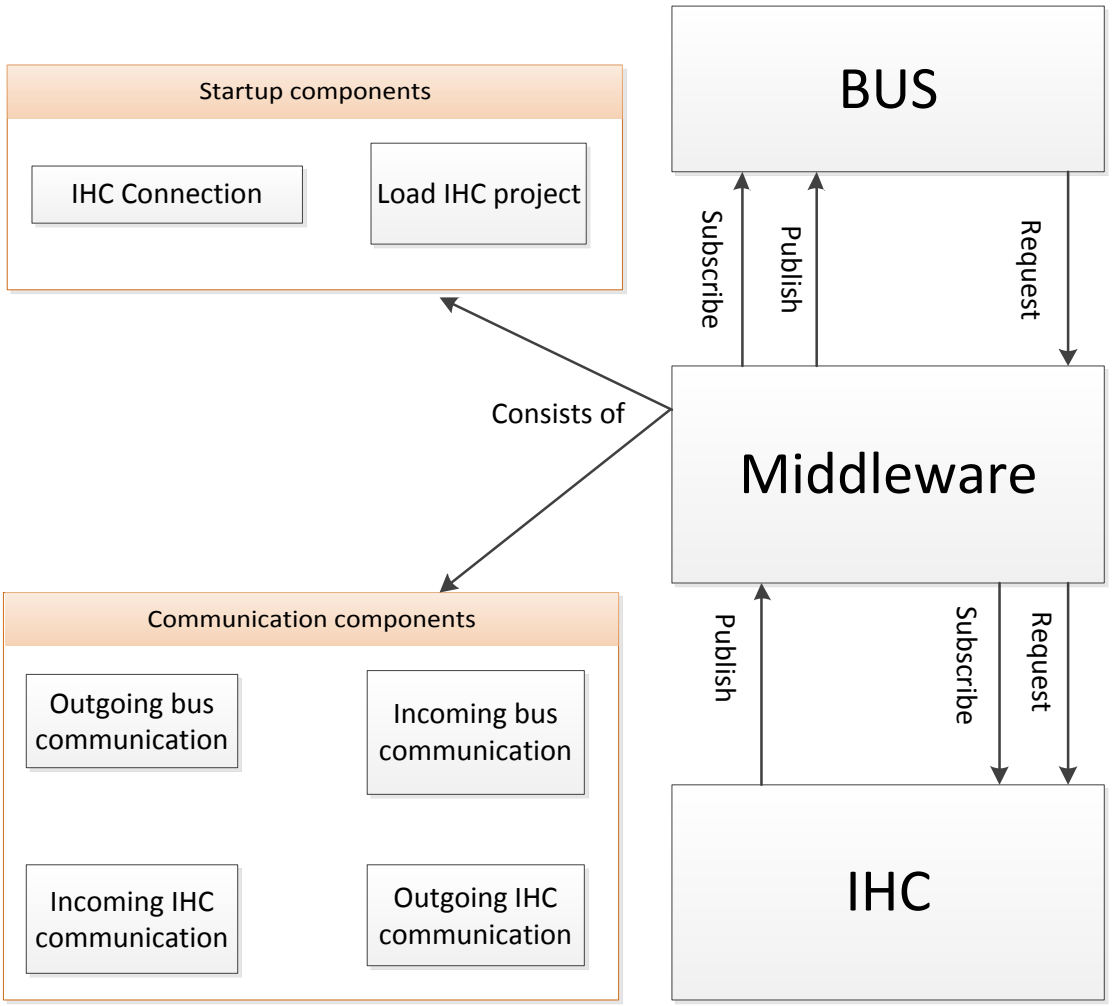

Figure 3.4: Component diagram of how the middleware communicates with the bus and the IHC and also illustrating the middlewares subcomponents

the state of given resource is made the internal and common space resource states are sought to be synchronized with the IHC resource

state. These resource states and their interaction with the rest of the system is depicted in figure 3.5 on page 49.

Hook What is meant by a hook is simply that the middleware has called the "WaitForEvents" method and is now awaiting the event threads return.

Request route A request route is a tracepath through the system of where the request started from. It can start from two places in this system. Either from the bus or directly from the IHC (Via a button in the house).

Internal event An internal event is an event that is only used inside the middleware layer when an internal resource state is updated by the event thread. The internal event will be picked up by the Resource-ValueChangeEventHandler which is described in section 3.2.2.5.

After this brief introduction to the communication components and some definitions and explanations the communication components will be described in detail in the following subsections. After the detailed descriptions a summarizing component-communication diagram 3.5 will be shown on page 49 along with a sequence diagram 3.6 on page 50 of a request coming from the bus. I recommend viewing these while reading the following four sections about the communication components.

3.2.2.2 Incoming bus communication (RequestEventHandlers)

This component takes care of all incoming requests to the IHC from the bus. These are requests to change the state of output resources in the system e.g. switch a lamp On/Off. The middleware is subscribed to a list of requests in the common space. This list of requests defines what the bus is capable of doing with the IHC. If the middleware isn't subscribed to a certain request, that request won't be received by the middleware when this request is published on the bus. Therefore when the system

starts up, the middleware subscribes to all request types in the common space regarding the IHC system.

The logic within each RequestEventHandler (there is one for each request subscribed to) differs depending on what the goal for that handler is. The "ChangeAlarmRequestHandler" fx, implements some logic to confirm that the user knows the old password before changing it to the new password, therefore it receives both an old and a new password within the request. Common for all handlers are that they after having performed some logic, do two things.

1. Update the state of the given resource in the common space. All subsystems can potentially access this resource, so one might consider making sure this is thread safe. But since there will be no other subsystems on the bus with access to resources connected to the IHC this is not an issue.
2. Call their own update method (there is one for each RequestEventHandler) that tells the given resource to update its internal state. This will automatically trigger a call to the IHC Event Interface which is described in the next section.

3.2.2.3 Outgoing IHC communication (IHC Event Interface)

The IHC Event Interface is the component that handles outgoing requests from the middleware to the IHC. It is here that requests from the middleware to the IHC is given i.e. change a resource state, wait for an event to occur or subscribe to a resource. This component is essentially a wrapper to the part of the OpenAPI that handles event-driven communication with the IHC.

In the previous section we saw that the middleware on startup subscribes to a list of requests that the bus is able to execute. In the same way the middleware also on startup subscribes to all the resources on the IHC that it wishes to be alerted of when they change state. When a resource on the IHC subscribed to by the middleware changes its state the middleware is alerted through the Event Thread described in the next section. When

the middleware wishes to change the state of a resource it gives a requests to the Event Interface to change the state of the resource. This results in an asynchronous event based system between the middleware and the IHC. Synchronous communication is not available through the OpenAPI provided by Schneider and nor is it needed since all calls to the IHC is treated in sequence.

3.2.2.4 Incoming IHC communicaiton (Event Thread)

The Event Thread handles incoming events from the IHC. It is a component that acts as a channel for changes of resources subscribed to by the middleware. It is a thread that is started at bootup of the system and runs all the time in the background polling the IHC for events. This is done by constantly placing "hooks" on the IHC that returns when an event occurs i.e. a resource changes its state (A hook equals a "WaitForEvents" call through the IHC Event Interface). If no resource state changes happen, the hook returns after a configurable amount of time. If the client is "gone" i.e does not have a hook on the IHC then the events will be queued on the IHC and returned (in sequence) when the next WaitForEvents method is called.

When an event occurs and is picked up by the event thread it will inform the internal state to update itself which will trigger the ResourceValueChangeEventHandler described in the next section. Now depending on where the request to change the given resource value came from, the internal resource state might already have been updated. If the "request-route" started at the bus, the internal state will have been updated by the Request event handler. If the "request-route" started at the IHC from a physical button in the house then the internal state will not have been updated first and hence the Event Thread will need to update it. See figure 3.5 on page 49 for an overview of the request routes.

When the internal state has been updated by the event thread an internal event is fired that is picked up by the ResourceValueChangeEventHandler described in the next section. If the internal state was updated by the Request Handler it will also be updated by the event thread (though simply to the same value). This is to ensure that an event to the bus is

fired no matter if the request to change the resource came from the bus or from a physical button in the house.

3.2.2.5 Outgoing bus communication (ResourceValueChangeEventHandler)

The ResourceValueChangeEventHandler's responsibility is to handle outgoing communication from the Middleware to the bus. The handler is called when the internal resource updates its own state due to an update call from the event thread. It is important to notice that the internal state has two different update methods. One for updates coming from the event thread and one for updates from the request handler. The ResourceValueChangeEventHandler will not be called if the internal state is updated by the request handler only when updated by event thread.

Note that it will not be called when the request handler updates the internal resource state, but since this update by the request handler will afterwards go through the event interface, the IHC and then the event thread then the ResourceValueChangeEventHandler will eventually be called.

At startup all resources in the IHC system is registered to the ResourceValueChangeEventHandler. The handler reacts when the internal state of a resource is updated by the event thread. When an event is picked up it first determines which kind of resource the event is on. When this is done the common space resource state is updated to match the internal state of the resource. Then a new event is created corresponding to the resource-type detected. The necessary data desired by the subscribing subsystems on the bus is populated to this new event and then it is published on the bus.

3.2.3 Communication diagrams

Figure 3.5 on page 49 compactly illustrates the communication pattern between the bus, the middleware components and the IHC. If a request is

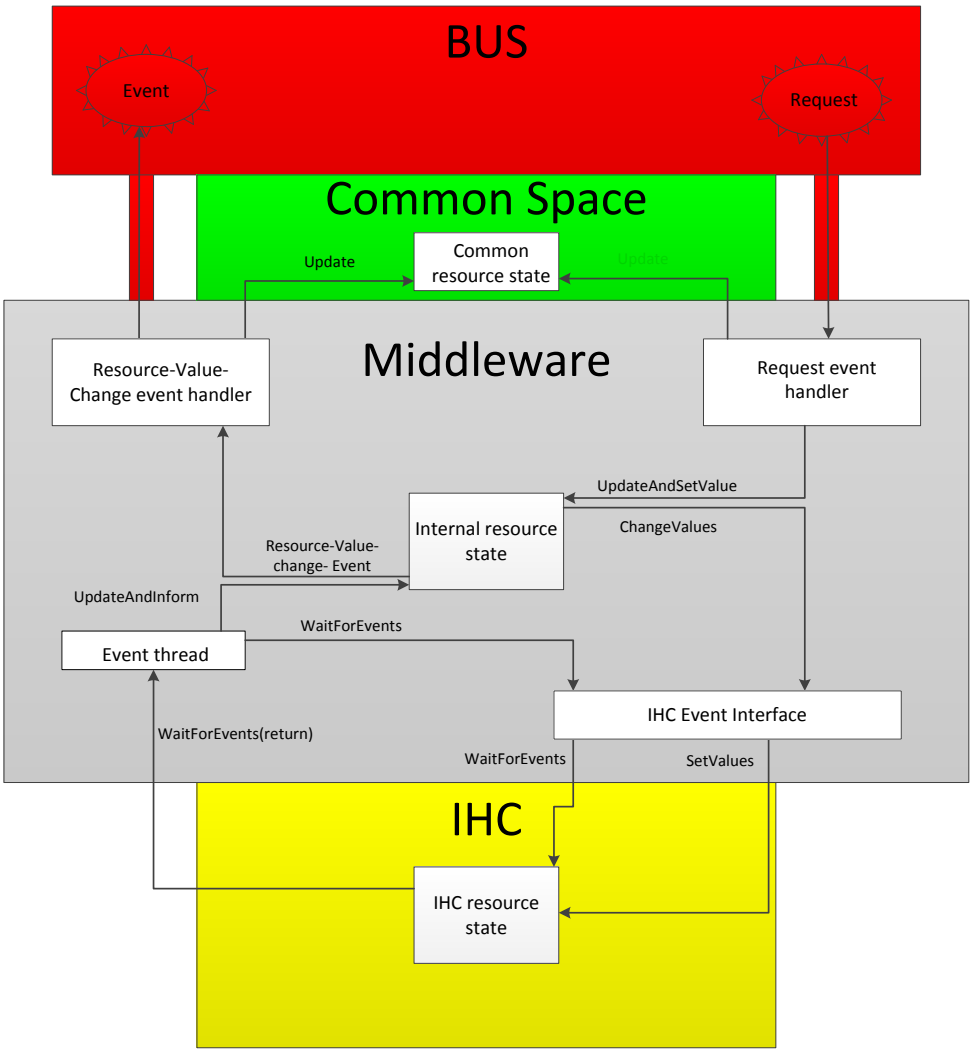

Figure 3.5: Component diagram showing the outgoing and ingoing communication from the middleware

received from the bus the long request route around the system is taken. The request is first handled by the matching Request event handler. Hereafter the Internal resource state is updated to the new value triggering a call to the IHC event interface. The interface calls the OpenAPI method SetValues which sets the actual resource value on the IHC. This will trigger the event thread that is constantly polling the IHC for events and when the event thread picks up the event it will update the internal value if it has not already been updated. The resource updates itself and then fires a resource-value-change-event which is picked up by the ResourceValueChangeEventHandler. Here an event matching the resource type is created to inform subsystems on the bus that the given resource has changed its state. The events data fields are populated and the event is published on the bus.

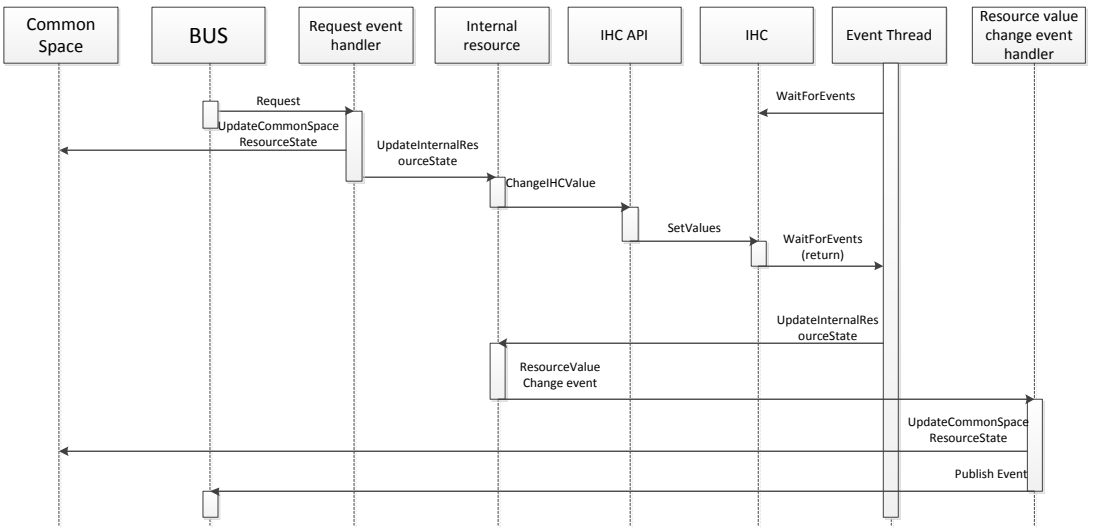

Figure 3.6: Sequence diagram showing the communication route between the different processes when a request is sent from the bus

This sequence diagram shows the same communication pattern as the component diagram figure 3.5. Note that if the request to change the resource state comes from the IHC, the calls before the IHC being active can be ignored i.e. the SetValues call and all calls before it.

3.3 Common space Integration

The architecture of our groups entire system involves a common interface that all subsystems must integrate into. This was done to make it easier for new subsystems to integrate into the system. One issue that we discussed in the group was to what degree we wanted to normalize data before it was sent over the bus. It was a possibility to simply send resource data in its raw form over the bus or we could try and normalize it as much as possible before sending it over the bus. Since we want the system to easily interact with new subsystems and to be easy to work with for future programmers the latter approach was chosen. Therefore each subsystem has a number of service contracts in the common space that data sent from the subsystem needs to be converted to before sending it over the bus. The service contracts concerning the IHC system are:

1. A class for each request type to the IHC. There is a request type for each output resource type in the IHC plus one for requesting the state of the entire IHC system.
2. A class for each resource type in the IHC.
3. A class for each event type in the IHC.

For the IHC middleware integration that we are creating this means that it must subscribe to each of these request types in the common space. Events that are published from the IHC middleware must also contain information on which event-type it is and what resource-type it is affecting (in addition to containing the resource ID).

Facilitating these requirements makes for a lot similar code but is necessary in order to provide the required information. The similar code occurs three times in the source code (mathcing to the three service contracts that must be met).

In this system a requesthandler for each requesttype is necessary. It is possible to have more than one requesttype registered to the same handler, but then you wont know which requesttype invoked the handler.

Part of the requesthandlers responsibility in this system is to update the common space resource state which has a resourcetype. Therefore each requesttype must contain information on which resourcetype it wishes to interact with. The alternative would be to have all requesttypes registered to one single requesthandler losing the information on which resourcetype it wants interact to with. This would require that the rest of the system is designed without resourcetypes.

As we have just discussed the resource-type is required. Therefore when creating the resources from the projectfile on startup it is necessary to check which resource type it is since the type is required when creating the resource. The alternative would be to only differ between boolean and set-point resources but this merely moves the problem of finding out the resource type to the other subsystems.

The same challenge is seen when the IHC publishes events on the bus. An event type is required. The alternative would be to send out either boolean events or set-point events no matter what happens on the IHC. This is possible since the event still contains the resource id and the state it has changed to, but again the problem of finding out the type of resource will simply be moved to the subscribing subsystem.

It would be simpler for the IHC subsystems if it didn't have to care about resource types since as can be seen on the figure above, only the control type of the resource would have to be known. In this way the only information that would have to be stored for each resource was its ID and how to control it i.e. setpoint or boolean. However as discussed earlier this would only push the problem higher up in the layers to the other subsystems.

3.3.1 Extensibility and simplicity

Providing these extra informations hurts the extensibility and the simplicity a bit. If the user suddenly decides to add an extra lamp or even a new room with resources to the system no coding at all to the middleware is needed. The only thing required would be to add it to the projectfile in LK Visual and the auto generation of resources in the middleware takes

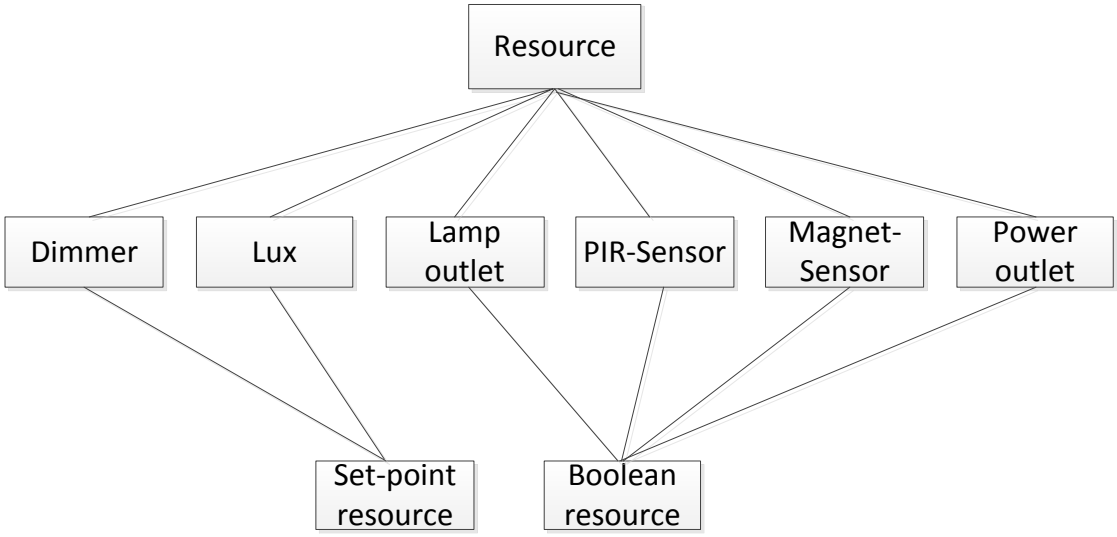

Figure 3.7: Shows what subtypes some resources are. Not all resources in the system are included

care of the rest. This is because a lamp is a resource-type that already exists. If however the user decides to install a controllable radiator system in the IHC (a new resource-type), some additional code would need to be added to the middleware. This code includes a check for this new type of resource in the LoadProject component, a new event type for events concerning this resource and a new request type so the bus can give requests to the new resource-type. These are the changes to the IHC system that needs to be done. The new resource-type would also need to be added to the common space and the other subsystems on the bus might also need some updates.

It is not possible to completely avoid having to add some code in the middleware when new resourcetypes are introduced, but the code work is easy to do since the template used by the other resources can be used with very few changes needed.

Chapter 4

Implementation

In this chapter i will walk through the boot-up components describing which coding patterns they use and how they work. A description and explanation of how the resource base is implemented will also be given.

4.1 Load Projectfile component

When the IHC integration system was initially implemented we used a manual setup where we had a resource-settings file wherein all resources were manually typed in. The resources ID's had to be fetched by manually inspecting the projectfile for the resource and its ID. Also if any additional resource information e.g. "location" was wanted you had to further inspect the file for that too. This quickly became very cumbersome and hard to overview when we added more resources with additional information attached since the slightest change to the projectfile would mean that the settings file needed to be updated. This was even if the change was only and additional resource of a type that already existed.

Therefore an automatic way of importing to and creating the resources in the middleware is needed. When the IHC system is booted we want the resources in the projectfile to be automatically created in the middleware even if new resources has been added since the last boot-up.

4.1.1 Importing the projectfile to middleware

Importing the projectfile basicly consists of four steps:

1. Check if a new there is a new projectfile on the IHC. If yes -> go to step 2. If no -> go to step 4.
2. Retrieve project segments from IHC and write them to a binary file.
3. Decompress the binary file to an xml file and store it locally on the server.
4. Deserialize the locally stored xml file on the server to xml class trees.

On bootup the first thing to happen (after establishing connection to the IHC) is the check to see if there is a new projectfile on the IHC. This check is simply done by comparing the version numbers of the projectfile on the client-side and the one on the IHC. These version numbers are fetched via the OpenAPI method `getProjectInfo`. If the versions differ the new projectfile on the IHC is loaded, if not the old projectfile is used.

If a new projectfile needs to be loaded the `getIHCProjectSegment` method is called to the OpenAPI. This returns a single segment of the projetfile. Therefore this call is put inside a for loop that runs as many times as there are segments. The amount of project segments is retrieved by calling the API method `getIHCProjectNumberOfSegments`. In each iteration of the loop a segment is loaded and written to a binary file. When the entire projectfile is written to this binary file we end up with a compressed file which is unzipped programmatically using Gzip. The result is an unzipped XML-file describing the entire projectfile from the IHC. This

file is saved in the clients folder appconfig (Which is created if the folder does not exist already).

The projectfile is now imported to the middleware and can be read directly as an XML file. This means that now we can programmatically access the resources on the IHC through this XML. This can be done through simple xmlnode querys. However we want to avoid fetching data directly from an XML file. The reason to not read values directly from the XML file is that if/when changes occur to the projectfile it may change the structure of the XML file, thereby affecting the way you access the data in the XML projectfile. This would mean that the programmer would have to change the xmlnode querys every time the projectfile was changed. So a generic solution is wanted that saves the programmer this hassle.

We decided to normalize all loaded projectfiles from the IHC using a solution as described by the figure below:

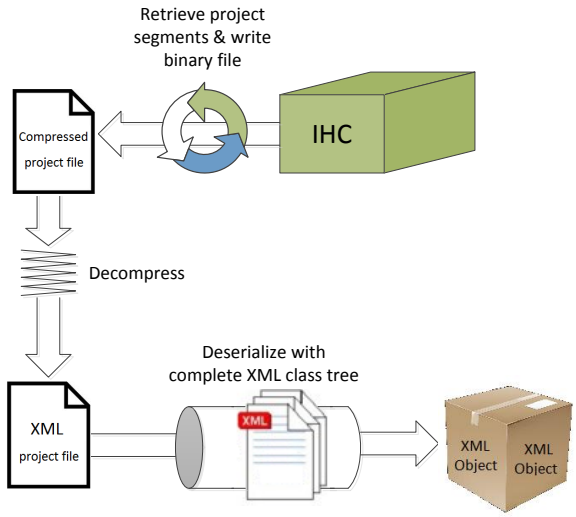

Figure 4.1: Diagram of the entire process of loading the IHC project into the middleware

The first three steps are described above, where the project segments are retrieved from the IHC, written to a binary file and decompressed with gzip resulting in an XML project file. However instead of retrieving resource data directly from this XML file we choose to deserialize the file first using a large almost comprehensive XML Class tree as the target type. This ensures that the project file will always have the same structure making the solution generic and the access to its data structured.

To do this i contacted Schneider Electrics to get an IHC projectfile from them that theoretically contains all resource types and control scenarios possible. From this file we used Visual Studio's built in XSD tool to generate an XML schema definition from the XML file. This is done by running XSD.exe tool from the visual studio command prompt typing: `xsd projectfile.xml`, resulting in an XSD file generated from the xml file describing the structure of the XML file. From this XSD file visual studio again has built in tool to generate XML class trees describing the structure of the XML file.

This means that we can generate XML class trees that theoretically should be comprehensive in terms of structural possibilities of the projectfile retrieved from the IHC. This XML class tree is used to deserialize the projectfile resulting in a uniformed projectfile that always has the same structure. This solution gives the most generic approach to the problem of uniforming the projectfile relieving the programmer of constantly having to maintain this part of the software. The disadvantage of this solution is if you haven't used a comprehensive file to generate the class tree i.e. there is a node in the XML file that is not covered by the XML class tree.

4.1.2 Generating and mapping resources from projectfile

One of the responsibilities of the Load Project component is to generate resources from the projectfile (From now on when "projectfile" is written we simply refer to the projectfile after it has been completely deserialized to an XML data object). When generating resources in the middleware we need to fetch three pieces of information from the projectfile. The ID of the input- or output resource, the type of the resource and the location

of the resource. To facilitate this, a massive for-loop is constructed that iterates through the entire projectfile. It does so in two levels, the first level being the resource location (a location equals fx "Entertainment Area"). For each location in the projectfile we have a for-loop iterating over each resource within that location. Whenever a resource is encountered we enter a switch statement that checks the name of the resource and maps it to the corresponding resource generation code. Here is an example of a PowerOutlet resource in the Entertainment area that has been mapped to the PowerOutlet resource generation code:

```
case 'S': //Power outlet

  if (String.IsNullOrEmpty(data.groups[0].group[group].
      product_dataline[dataline].dataline_output[0].id))
        {
            Console.WriteLine("Skipped Resource in location: " +
                  data.groups[0].group[group].name + "with
                                resourcetype: " + ResourceType.
                                    PowerOutlet);
       continue;
    }

    DecimalID =  UInt32.Parse(data.groups[0].group[group].
        product_dataline[dataline].dataline_output[0].
                                    id.Substring(3),
    System.Globalization.NumberStyles.AllowHexSpecifier);

    guid = Common.Helpers.GuidMapper.GetGuid(DecimalID.
        ToString());

    resourceBoolean = new ResourceBoolean(DecimalID,
        ResourceType.PowerOutlet, location, guid);
    resourceBoolean.Valid = true;

    SubsystemItem item = new PowerOutlet();
    item.Id = guid;
    item.LocalId = DecimalID.ToString();
    item.Name = location;

    break;
```

The code creates a new boolean resource containing four parameters. The 1st parameter `DecimalID` is the exact ID that must be parsed to the IHC

when interaction with the resource is wanted. The DecimalID is fetched from the XML Data object. The 2nd parameter is the resource type. This type does not need to be fetched from the XML but can be directly set since we know that the resource is a powerOutlet when executing this code. The 3rd parameter is the location which is fetched during the outer for-loop and is used for all resources in each iteration of the outer for-loop. The last parameter is another ID for the resource used globally by all subsystems on the BUS.

After the creation of the resource for the middleware the resource also needs to be created for the common space. This is done by creating an object of the type `SubsystemItem`. This item is the common space resource that was described in the design section which is updated by the request- and eventhandlers.

When the big for-loop is finished the entire XML object has been iterated and each resource in it has been generated for the middleware and the other subsystems. While the for-loop was running the middleware also subscribed to all the resources on the IHC corresponding to the resources created in the middleware. When all resources have been created they are also registered to the ResourceValueChangeEventHandler so when their internal state is updated the handler will be informed.

4.2 Resource Base

The resource base is the classes that are instantiated when new resources are created. Currently it consists of three classes one of them being the abstract class `ResourceBase` from which the others inherit from. The two resource-classes that can be instantiated are the `ResourceBoolean-` and `ResourceSetpoint` classes. These resource classes represent the different types of resource-control present in the system, which at the moment only consists of on/off and setpoint resources. If the need for a new type of resource-control arises one simply creates a new class inheriting from `ResourceBase` with a new set of control methods. To illustrate what the shared logic is between the resource types a class diagram of the resource base is shown here:

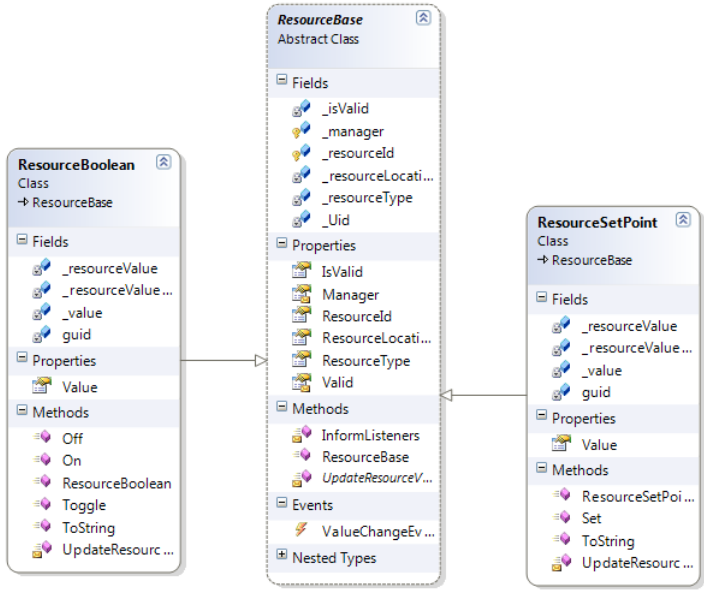

Figure 4.2: Class diagram of the resource base

The diagram shows that all resource-types contain the three pieces of information: ID, type and location mentioned in the Load Projectfile section 4.1.2 on page 58 and therefore these are inherited from `ResourceBase`. Where the classes differ is in the control of the resource state. In the `ResourceBoolean` class we see that it has the three methods: On(), Off() and Toggle() to control it, while the textttResourceSetpoint class has only one method called Set(). These control methods directly change the state of the internal resource in the middleware triggering setValue() a call to the event interface as described in section 3.2.2.4 on page 58.

You will also notice that each resource has a valueChangeEvent that is called when the UpdateResourceValue() method is called by the event thread. The valueChangeEvent is picked up by the ResourceValueChangeEventHandler as described in the section 3.2.2.5 on page 48.

Chapter 5

Evaluation

In this chapter i will evaluate the project reflecting on the experiences made throughout it as well as discussing possible additions and future plans for the Solar Decathlon house.

5.1 Testing and unsolved issues

Testing of the IHC integration system has been very limited since neither the IHC resources or the server operating our entire software control system for the house was installed in the house before around the middle of June. Therefore testing with the IHC system has been done on a test-IHC connected to a single lamp and a single motion sensor. This however proved that the IHC integration system works perfectly fine with a response time around 300 ms. A lamp connected to the IHC (both wirelessly and wired) could be turned on both through a bus request coming from a 3rd party app or through a physical button or a motion sensor. Both requests resulted in the lamp being turned on and an event fired onto the bus containing information about the affected resource and its

state after the request. Booting of the system works as intended. The IHC middleware layer correctly connects to the IHC and generates all resources from the projectfile and informs the other subsystems of them.

It should be noted that not all resource-types utilized in the system have been tested since they were not installed early enough to test. But all resources in the middleware are implemented using the same coding pattern so if some of them functions properly the chance that the other do too are high. A table of tests conducted and some that should be performed in the future are listed in table 5.1 on page 65.

There has also not been performed any stress tests on the system e.g. sending 1000 switch on/off requests from the bus to a lamp without any delay in between. This is not something that will come up at all when the system is implemented in the house since a user cannot generate requests fast enough for this to be a problem (even if he clicks the button in the house as fast as he can), since there is a built in delay timer in the IHC for how fast input-resources (e.g. Buttons and motion sensors)can affect an output resource. This is not the case when communicating through the bus though, but since this is being encapsulated by providing our own app developed by ourselves for the user to interact with the system this should not be a problem. Still this is something that should be tested further not only in order to further stabilize the system making it more robust but also for when other programmers are going to work with the system and play around with sending requests to the IHC.

Related to stress testing the IHC system, thread safety is another issue. Fixing this issue is however very hard due to the nature of how the IHC works. The problem briefly described is that when a request is sent from the bus then if the state of the resource is requested for some reason whilst the request is being processed in the IHC integration layer the wrong state might be read leading to race conditions and deadlocks. The same problem could also occur if a request is sent from the bus and in the small time frame it is being processed a physical button in the house is pressed affecting the same resource. Granted this will practically never come up since this would require extreme timing from the inhabitant in the house, but still this is something that should be looked into.

Scenario	Tested	Evaluation/prediction
A lamp is turned on via a button in the house or from the bus.	Yes	Succes. The light turns on within 300 ms
A lamp is dimmed up/down from the bus or by a button in the house	Yes.	Succes. The lamp is dimmed 20 % down as is configured by standard in the projectfile.
A lamp is turned on when a motion sensor is triggered.	Yes	Succes. The light turns on within 300 ms
A power outlet is switched off/on	Yes	Succes. The source connected to the power outlet does not receive any power after approximately 300 ms.
A lux sensor automatically adjusts the light level on a lamp	No	The lamp should at all times maintain a lux around 500. It is expected the lux level will swing up and down around this level.
The alarm system is activated	No	Within a second the alarm system should be activated and the user will have 20 seconds to leave the house.
The alarm system is triggered through motion sensors	No	The alarm system should be triggered and the sound generator resources should be activated that produces a 102 dB sound.
The alarm system is triggered through magnet sensors.	No	The alarm system should be triggered and the sound generator resources should be activated that produces a 102 dB sound.
The projectfile on the IHC is fetched and auto processed.	Yes	The resources in the projectfile are generated programmatically with properties matching exactly with the projectfile.
The auto generation of the resources from projectfile is generic.	To some extent	We have tested with a lot of different kinds of projectfiles on the IHC which all worked fine, but since the possible number of projectfiles are very big it cannot be a 100 % guaranteed that ANY projectfiles is processed correctly.

Table 5.1: Test table showing test scenarios, whether they were tested and the evaluation of the test or prediction if not tested.

5.2 Future development and extensions

Here possible extensions to the IHC integration system will be described as well as what the future of the Solar Decathlon project looks like.

Easy adding of resources is a feature that future users as well as programmers could benefit from. By this i mean developing an app that does two things. First it should serve as encapsulation from the LK Visual program saving the user the hassle of having to learn how to use LK Visual in order to add new resources to the system. Additionally the app could even support adding of new resource-types by auto generating the necessary source-code to the system.

Automatic adjusting of light level is something that is only done at the workstation at the moment. By installing Lux-Sensors throughout the house automatic adjustment of indoor light level compared to the sunlight could be done. This could further reduce unnecessary lights in the house. Of course this needs to be able to be manually overridden like all other functions in the house.

Robustness through further testing is very important and is definitely one of the key points to work on in future iterations of this software system. This has already been achieved somewhat by striving to make the startup component and communication component as simple as possible in order to easily debug them. But actual testing still lacks since the equipment to do so was only recently installed in the house.

On the 1st of September 2012 instalment of our groups software system in the official house began. This takes place in Madrid where the official Solar Decathlon 2012 Europe competition will be held. Here further testing of the system will be done in order to improve the robustness of it and to make sure everything runs smoothly for when the jury will be judging and measuring the house. [1]

[1] http://www.solardecathlon.gov/sd_europe.html

Current status on the project as of the 7th of september 2012 is that the competition is already underway and the system proposed in this report is already being installed in the house. It feels like we have achieved an overall well designed system that will serve the house well by saving lots of energy and making it intelligent which also makes for a good potential for future usage.

5.3 Initial conclusions

Evaluating the project some initial conclusions can be made.

Communication is very important when up to around 140 people from DTU are working together on the same project. People whose only job is to communicate are needed so developers don't need to attend meetings all the time but can focus more on the technical side of their project.

The IHC integration system works as intended by creating all the resources from the projectfile and offering the bus almost complete control over them. Further work on the IHC system should be focused on testing since this will increase the robustness of the system which is a very important quality.

The future of the project looks very promising with the competition in Madrid starting in September. We will need to test the system further and this is being carried out by our group as this thesis is being written, but i am positive that we will have a functioning system that will impress crowds, jury members and student groups as well as the administration working on the project.

Chapter 6

Conclusion

This report is motivated by DTU's participation in Solar Decathlon Europe contest 2012 on building the most energy efficient and innovative house. Part of this task was installing an intelligent home control system in the house which was carried about by myself and the other members of the Control Systems group. The specific task this report covers is the integration of the IHC system to the central message bus. To support us we had five different sponsors that provided us with system hardware/-software and know-how to install it.

The proposed solution consists of an event-based integration between the IHC and the bus. This is facilitated by creating an event-based communication both between the middleware and the IHC and between the middleware and the bus. Regarding the projectfile on the IHC we have made a solution that always synchronizes with the current projectfile on the IHC so the latest projectfile is used in the middleware. Furthermore resources are automaticly generated from the newest projectfile and registered with the event system. The reading of the projectfile is done in a generic way that should be able to read any projectfile created in LK Visual.

6.1 Evaluation

The basic resources that are used most often such as a lamp, dimmable lamp, motion sensor and power switch have been tested and proved to function properly. Conluding on the small amount of tests made on the system described it looks promising. The basic resources in the system have been tested and proved to function properly. The more advanced resources are implemented with the exact same coding pattern and integration with the bus so these should theoretically also function properly.

As mentioned in the evaluation the system has been hard to test since the physical resources in the house were not installed until around mid-july. Therefore testing has not been the main priority of this system but rather the extendibility and robustness of it. We can conclude that the system is very extendable since new resources very easily can be added/removed and new resource types can be added/removed with a small amount of work based on coding patterns already implemented. Regarding the robustness very little can be concluded since the system has not been installed and tried out in a real house yet.

6.2 Communication role

During the creation of our groups control software system for the house, i have acted partly as a communications person acquiring needed information for the group from other student groups working on the house, company sponsors and project administrative information. This job consists of two parts. One is to gather administrative information for the group e.g. meetings everyone from our group needs to attend and general information regarding the competition. The other part is to gather information from other student groups and sponsors regarding the actual resources that are going to be placed in the house.

It started out when i was the only one in the group who was able to attend the weekly Monday meetings that was held from around February all the way through to somewhere in the middle of May. These meetings

provided administrative information for our group. During this period it became a natural transition that i also had contact with the companies and other student groups who attended these meetings since i needed information from them regarding which resources we were going to use in the house and how they were best utilized. One needs to consider that this information is relevant for everyone in our software group but since i am dealing with the lowest software-layer it was me who needed the information first since i could not disregard from it like the people creating the higher-layers could. In order to program the IHC Control module in LK Visual i needed to know exactly which resources we were going to have in the house and where they were going to be placed. Therefore it was natural that i was the one to have the contact with the students and other student groups which i did not mind at all.

Contact with the other student groups mainly concerned what their wishes for the house were regarding lighting. The interior designers had a long list of wishes for where and how lighting in the house should work. Some of these wishes included an outside preprogrammed light show and special placement of lamps and when they would turn on. When wishes from the student groups was gathered i contacted the sponsors delivering the materials and discussed to which extent these wishes could be met with their products. Also if and how their products was compatible with the IHC system.

After working on this project for more than half a year now i have realised that communication is a huge part of a project this size and can take up a lot of time. It's very important that the different groups working on the project talk to each other about their wishes and to what extent they can be met. This has been somewhat of a big problem on this project when the different student groups keep on changing what they want and adding new things last minute. When future iterations of this project is undertaken i would greatly recommend communication is emphasized more than it has been on this project.

Wrap up

Overall the system seems to meet the overall requirements set out which were to integrate an IHC system to a central bus architecture. The parts

of the system that have been tested have been tested have a reasonable response and function properly. The system correctly and generically fetches the projectfile from the IHC and generates resources programmatically ready for integration with the bus through an event-bases system.

In the future further testing of resource functioning and the general robustness of the system should be carried out. It would also be very interesting to extend the system so new resource types easily can be added via some auto generation of code.

Appendix A

Appendix 1 - Classdiagram

This appendix contains the class diagram for the big single class that contains all the main functionality.

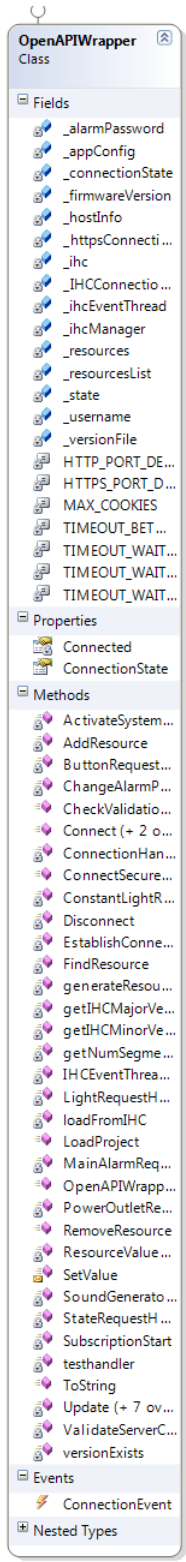

Figure A.1: Class Diagram of the big central class with most function-

Appendix B

Appendix 2 - LK Visual file (top)

Top half of Resources in LK Visual shown.

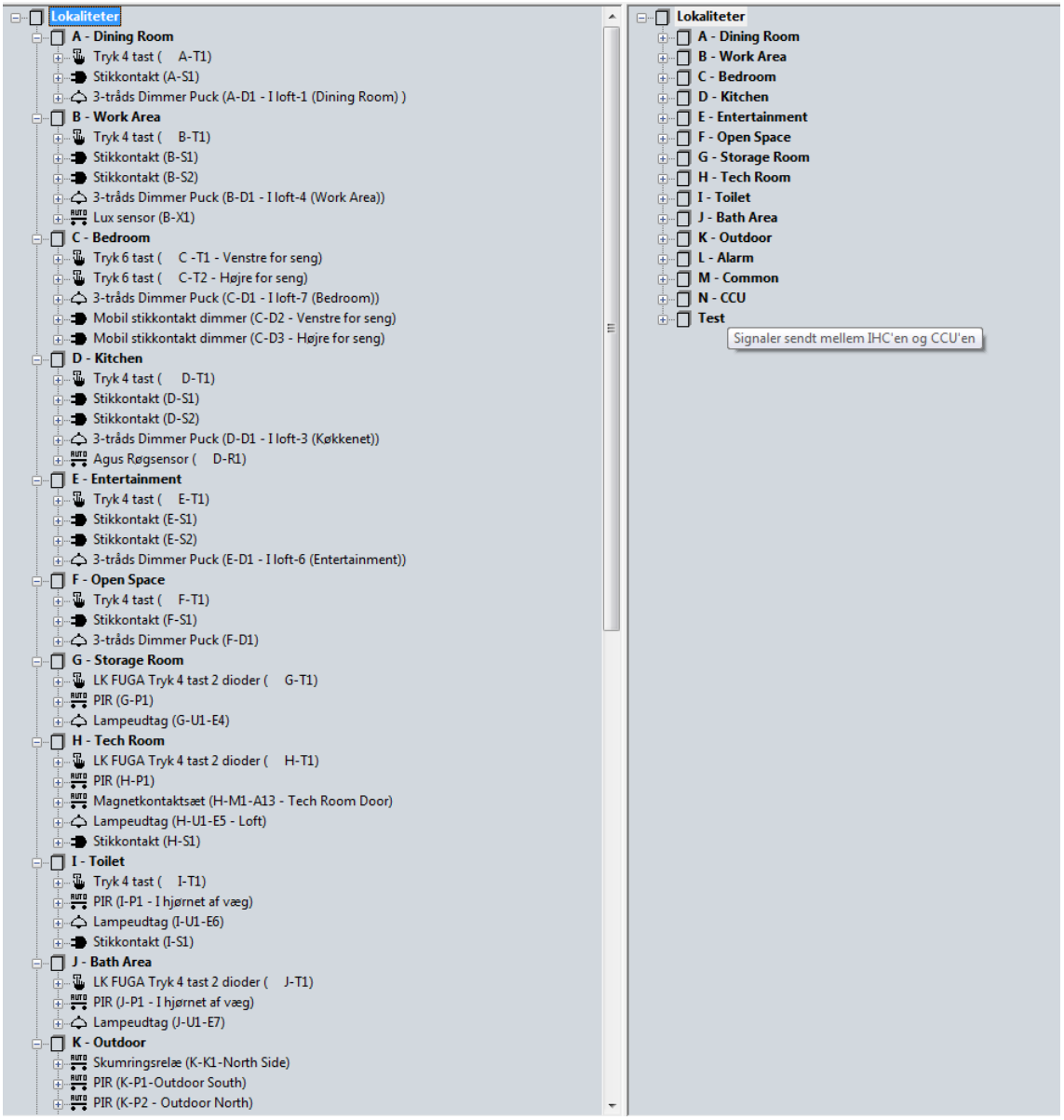

Figure B.1: The top half of all the resources in the house

Appendix C

Appendix 3 - LK Visual file(bottom)

Bottom half of Resources in LK Visual shown.

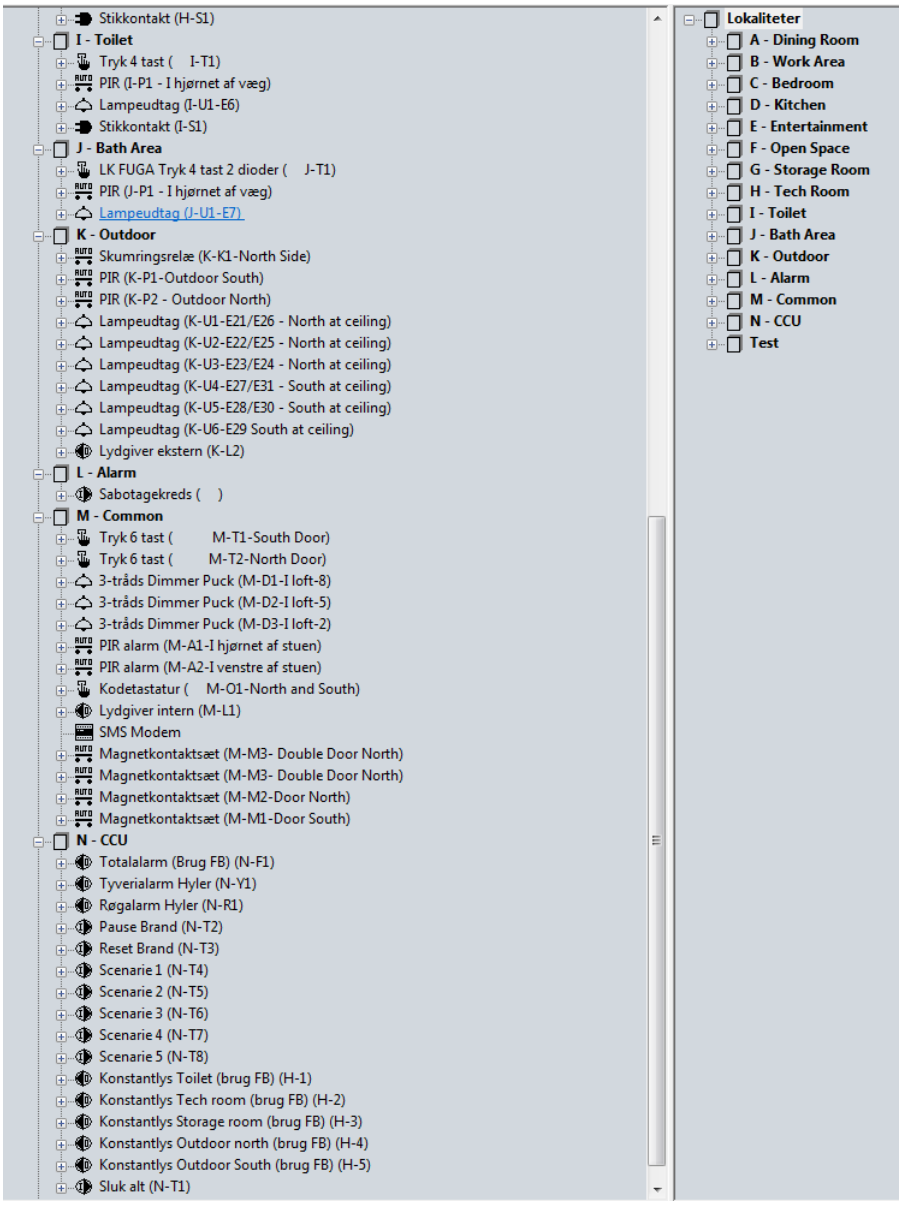

Figure C.1: The bottom half of all the resources in the house