MASTER'S THESIS | LUND UNIVERSITY 2015

Overview browser for PalCom assemblies

Vatan Bytyqi, Jonas Jinbäck

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2015-43

Overview browser for PalCom assemblies

(A Master's Thesis at Lund University)

Vatan Bytyqi
ada10vby@student.lu.se

Jonas Jinbäck
ada10jji@student.lu.se

September 10, 2015

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisor: Boris Magnusson, Boris.Magnusson@cs.lth.se

Examiner: Görel Hedin, Gorel.Hedin@cs.lth.se

Abstract

Internet of Things (IoT) has become a frequent subject mentioned worldwide. PalCom is a system for handling the problems that occurs in IoT applications and is focused on the user groups that are developers and skilled PalCom participants. This forecloses a higher level user group for managing their own PalCom network, e.g. end users for a smart home where they build their own smart home manager with PalCom.

In this Master's Thesis we came up with a solution for a tool that shows an overview of an assembly and that is able to edit such an assembly. The assembly is used to create and manage the devices in the network. The evaluation was done by letting people try our solution and the old one and then compare the results.

Keywords: IoT, PalCom, overview browser, assembly editor, smart home

Acknowledgements

We would like to thank Boris Magnusson for his guidance throughout the work and for providing us with material that we weren't able to get. We would also like to thank Mattias Nordahl and Björn A. Johnsson for their help in working and making changes in PalCom. Lastly we would like to thank all the persons that were willingly to participate in our experiment. Thank you all for your contributions that made this work possible.

Contents

Chapter 1
Introduction

This chapter provides a brief introduction to our Master's Thesis at Lund University. We will describe the problem and the background to our work. This chapter also presents how the responsibilities were divided between the two authors to complete this Master's Thesis.

Internet of Things (IoT) is a very hot topic worldwide and there are many and increasing devices that has enabled connectivity in all areas - industry, healthcare, private homes etc. (more information about this can be found in section 2.1). A system that enables all these connected devices to communicate is required and currently there are some solutions developed for accomplishing these tasks. One of them is the PalCom system developed by universities for over 10 years. The PalCom project was initiated in 2004 and funded by the European Commission, within the Society Technologies priority in the Sixth Framework Programme, for four years [3]. After these four years PalCom has been funded and developed mainly by Lund University consisting of professors, PhD students, Master's Thesis students etc. The PalCom system has been designed with the end-user in mind, but the focus of the implementation work has not yet been on these aspects and in particular a more intuitive GUI for end-user-composition was necessary. This has led to a Graphical User Interface (GUI) that is difficult to handle for people outside the PalCom development team.

1.1 Problem statement

PalCom is mainly handled by the professors and students developing the system. The implementation is in most cases not carried out by the people using it but rather by a PalCom developer. To open up the opportunity for other people, not the PalCom developers, to connect the devices and make them communicate with each other, one need to have a system that is easy to handle and to interact with.

There is also a problem for the PalCom developers when they make connections. The problem is the lack of an overview of what the connections look like when there is a system

consisting of many devices. The developers often need to draw on paper how the system works since it is difficult to determine it by the current GUI.

The problem can be divided into two separate problems which require the same solution. Therefore, it is convenient in this case to do one master thesis for solving two problems.

1. It is difficult for non-technical users to create an assembly without a proper introduction.
2. It is impossible to get an overview of how devices are connected in a PalCom assembly.

Transforming these problems into questions that we will be answering with our thesis will be:

- How can non-technical users create an assembly without a proper introduction?
- How can PalCom users get an overview of how devices are connected by applying the same solution as for the first question?

1.2 Method

In this thesis similar editing tools and design approaches that can solve the stated problems will be evaluated, see chapter 3. The implementation of the new developed editing tool during this thesis will then be based on the result of our previous evaluation. To finally evaluate the new editor and compare it to the old solution an experiment of a smart home scenario (see section 1.4.2) will be conducted. In the scenario experiment the time efficiency of the new editor will be measured, i.e. if the new editor is more time efficient regarding users understanding how to create an assembly. In simple form the method will be:

1. Evaluate similar tools
2. Implement a solution
3. Do an experiment on real test subjects with the two editors to see if problem 1 is solved
4. Evaluate and compare our editor with the old assembly editor to see if problem 2 is solved

1.3 Related work

There has been a previous overview browser [13] for PalCom but it is not compatible with the current PalCom version, v3.1.12. This overview browser only solves one of the problems, namely the problem for palcom developers to get an overview of the assembly. It is nevertheless possible to use this overview browser as a starting point in this thesis for solving the problems. The thesis is highly dependent on the work in the doctoral dissertation

by David Svensson Fors[4]. His thesis dissertation provides all the required information regarding the assemblies and all the functions that the GUI needs to support to be fully functional for PalCom.

1.4 Approach

Initially the authors will commence by stripping down the current PalCom editor to get all the parts that the new editor needs to support. To solve the problems the previous browser solution, that enabled an overview, will be studied and on similar solutions used in other systems that solves the stated problems. To get an understanding if the developed solution in this thesis is better for non-technical users an experiment will be performed, see section 1.4.1, on some random test persons.

1.4.1 Scenario experiment

One way to verify the work in this thesis and its improvement over the previous solution is by applying some kind of measurement. The selected measurement that fulfills the requirements of this thesis is the amount of time it takes for the test subjects to accomplish a task, see section 1.4.2. The task will be conducted by the random test persons by using both the previous editor and the new solution, followed by comparing the time results and lastly extracting an evaluation based on that. This will provide the conclusion if there are any improvements and simplifications of the new editor. In comparison, by only conducting observations and interviews of random persons to receive data for evaluation is difficult, due to the reasons that measuring feelings and thoughts can result in skewed and ambiguous results.

1.4.2 Smart home scenario

Internet of Things is wide spread also in home appliances. There are Smart TV's, connected refrigerators, connected light bulbs and much more. This is not what every home has installed today, but the trend is going towards that every home will be connected. In the experiment conducted in this thesis, it will be shown how PalCom can be used in a real home scenario with some connected devices. The scenario consists of:

- 40" Philips Smart TV [14]
- 2 Tellstick on/off switches [11]
- Linocell Bluetooth camera shutter [9]
- Regular lamp for the living room.

All the devices are going to be used according to their manufactured purpose, i.e. the TV is going to display a channel, the switch is going to turn on/off some lights etc. One exception is the Linocell camera shutter, which is a bluetooth device that is connected to a smart phone to take pictures remotely. This device will be used as a button to trigger some event in your home.

The scenario is as follows:
You come home from work and have an hour to rest before you need to cook dinner. This is something that you do everyday. When you enter your home you have a physical button (Linocell device) on the wall near your kitchen and living room. When you push button number 1 on the Linocell device a reaction is started in your home

- *The light in the living room is turned off (if it was off from the beginning nothing will happen).*
- *The TV in your living room is turned on and set to your favorite channel which is SVT2 in channel 2 (SVT1 is in channel 1 and TV4 is on channel 3).*

After an hour you feel it is time to start making dinner. Then you leave the living room and on your way to the kitchen you pass by that button that initialized the reaction earlier (started the TV, etc.). But this time you push button number 2, which causes another reaction in your home

- *The TV is turned off.*
- *The light in the living room is turned on.*

Now you have the energy to make dinner.

1.5 Work responsibilities

The two authors have been involved on all aspects within the scope of this thesis. Nevertheless, a division of the main responsibilities and the parts where each has contributed the most can be made following.

Vatan was responsible for the following parts

- Development
  - Data structure for the new GUI
  - Function for creating Synthesized Services
  - Function for creating variables
  - Creating a service for the Bluetooth device
- Report
  - Overall structure
  - Chapter 1 - Introduction
  - PalCom overview browser in 3.2
  - ABB Editor in 3.3
  - IFTTT in 3.4
  - Solution of services in 4.2

Jonas was responsible for the following parts

- Development
  - Creating Graph manager, which can be divided in a backend and frontend part.
  - Connecting graph to the new assembly script
  - Creating a new service for TellStick
  - Creating services for the Philips TV
- Report
  - Chapter 2 - Background
  - Assembly editor in 3.1
  - Graph implementation in 4.3

The following parts are where both have contributed equally and are equally responsible of:

- Development
  - Handling the scenario experiment
- Report
  - Solution of PalCom editor 4.1
  - Evaluation
  - Discussion
  - Future work
  - Conclusion

Chapter 2
Background

This chapter will describe two major subjects in order to help the reader understand the contents of this master's thesis. The first subject to describe is Internet of Things (IoT) and what problems have emerged regarding this. The next subject is PalCom, which solves the problems emerged with IoT. Two PalCom programs will be used in this thesis, TheThing and BrowserGUI. Therefore, a description of these two programs will be embraced in this chapter as well.

2.1 Internet of Things (IoT)

The Internet of Things (IoT) is an expression that emerged during the time when RFID was growing as a technical field and RFID devices began to connect to the Internet in 1999 [16]. IoT means that physical devices create a network with other devices so that they can exchange data with each other. The possibility for many devices and sensors to send data over the Internet opens up opportunities to create advanced distributed systems with many different sensors/devices from different manufacturers. These networks of physical devices can react when things happens in the physical world and notify each other of the changes to perform different tasks in tandem.

The problem, however, is that even though the devices can be connected to Internet and other networks, they are not constructed to communicate with each other. There must be a mutual protocol so that everyone understands how to communicate with the devices.

The applications of IoT can be in almost every field. When you have a network connected device with low CPU and power it can be made cheaply and portable to put anywhere. The devices can be used to send data for monitoring natural ecosystems or even buildings and factories to help improve resource usage. IoT is used for environmental monitoring to assist in protection of water and air pollution by collecting data and sending it to a main controller. It is also used for infrastructure management in large cities to assist in traffic or warn about dangers in the area. Another growing area is in healthcare

where IoT is used for monitoring a patient's health remotely. The doctors simultaneously follow a large sample of patients' health conditions without forcing the patients to leave their homes.

Today we think of IoT also as Ambient Intelligence (AmI), even though AmI doesn't require any Internet connection. But we expect that our connected devices should be smart enough and together with other connected devices learn patterns and help us in our everyday life. This is a vision of the future in consumers electronics, telecommunications and computing. But this future is present, considering the vast amount of smart smoke detectors and alarm systems by NEST and others.

2.2 PalCom

PalCom is a system that tackles the problem that has emerged since IoT has become a reality. PalCom is an abbreviation of Palpable Computing which is a concept for computing that will make technologies a lot easier to understand, use and to construct on the fly. Palpable computing is about constructing IT that is easy to grasp, modify and understand also for users that are not software developers.

PalCom began as a project from the European Union and was funded by the commission for four years [3]. After that it was continued by Lund University and is now applied in the areas within the academic world and in hospitals where different devices are connected to monitor a patient's health remotely. The remote monitoring in healthcare is an area growing rapidly within PalCom. Therefore, a separate project named itACiH[6] emerged to be able to fulfill the demands required within this area.

PalCom is now an open source architecture developed in Java for making different devices on different datatransferprotocols (bluetooth, IP etc.) to be able to understand each other and communicate. The PalCom architecture creates an application layer for the users so that all devices are represented equally even though they run in different protocols. This enables the users to only see the relevant part and make seamless integration of devices to build up their distributed system.

The core components in the PalCom architecture consists of:

Devices corresponds to physical hardware or software that are thought of as objects. A device has one or more services. They have a unique ID in the PalCom network and the device will automatically be discovered when it is connected through its default network connection (bluetooth, LAN, etc.).

Services have commands that can either receive or send. These are the commands that are presented to the user so they can easily interact with the device.

Connections is the function that allows communication between services. If there is no connection established between services on different devices there can't be any communication.

Assemblies are user defined rules and actions that tells which services are connected and how they should behave together. These are created as a script by using an XML editor or by using the Assembly Editor, see section 3.1. When you have created an

assembly you need to load it into PalCom, this can be done with TheThing or with the BrowserGUI.

2.2.1 TheThing

TheThing is a middleware where users can load assemblies and stand-alone services. These are loaded as class files or jars and will then be available for users to interact with through the BrowserGUI 2.2.2. TheThing is a Java program with a GUI, see figure 2.1, and it will show up as a device in the PalCom network for other services to interact with it.

A very neat feature in PalCom is that it is possible to make two PalCom devices in different networks located in the opposite corner of the planet to be visible as if they were on the same network. By creating an IP tunnel between TheThing and another TheThing or BrowserGUI it will be possible to call the services running on the different places seamlessly.

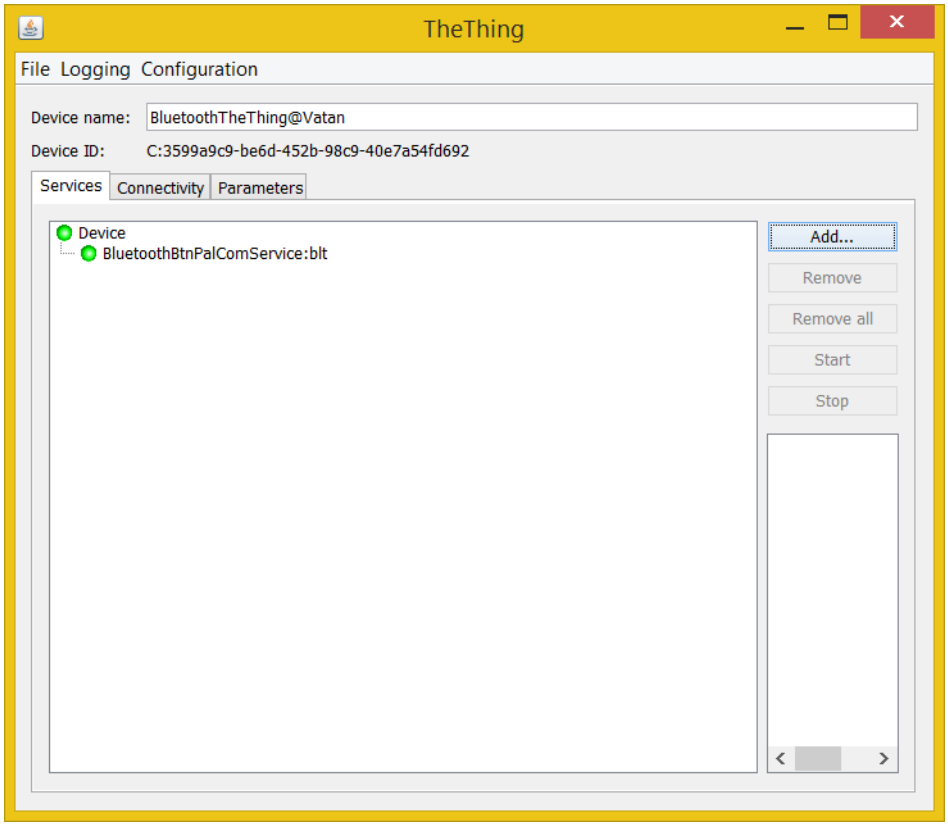

Figure 2.1: TheThing running a service called "BluetoothBtnPalComService"

2.2.2 BrowserGUI

The BrowserGUI is a manager for creating a complete distributed system that is easy to grasp and overview. It is a java program that uses the Palcom system to find connected

devices, run assemblies, create IP tunnels to other Palcom networks etc. The GUI can be divided into three major parts:

Devices are shown as number 1 in figure 2.2. This area displays the current devices visible in the PalCom network and their status. It is also possible to see what services the devices are running and which commands that can be used for each service.

Assemblies are located at number 2 in figure 2.2. This area holds the scripts created with this Browser and whether they are running or not. There can be multiple Browser-GUI's in the PalCom network but it is not possible to see what assemblies the other browser has created. The assemblies are stored locally and will show up every time when the BrowserGUI starts but an assembly can run on other PalCom devices as long as the device has support for executing assemblies even if it hasn't created it or can't create assemblies.

Assembly editor is located at number 3 in figure 2.2. This is where you will edit an assembly that you have created. Furthermore, this is the area which causes the defined problems in our thesis. We will thoroughly explain the editor in the next chapter.

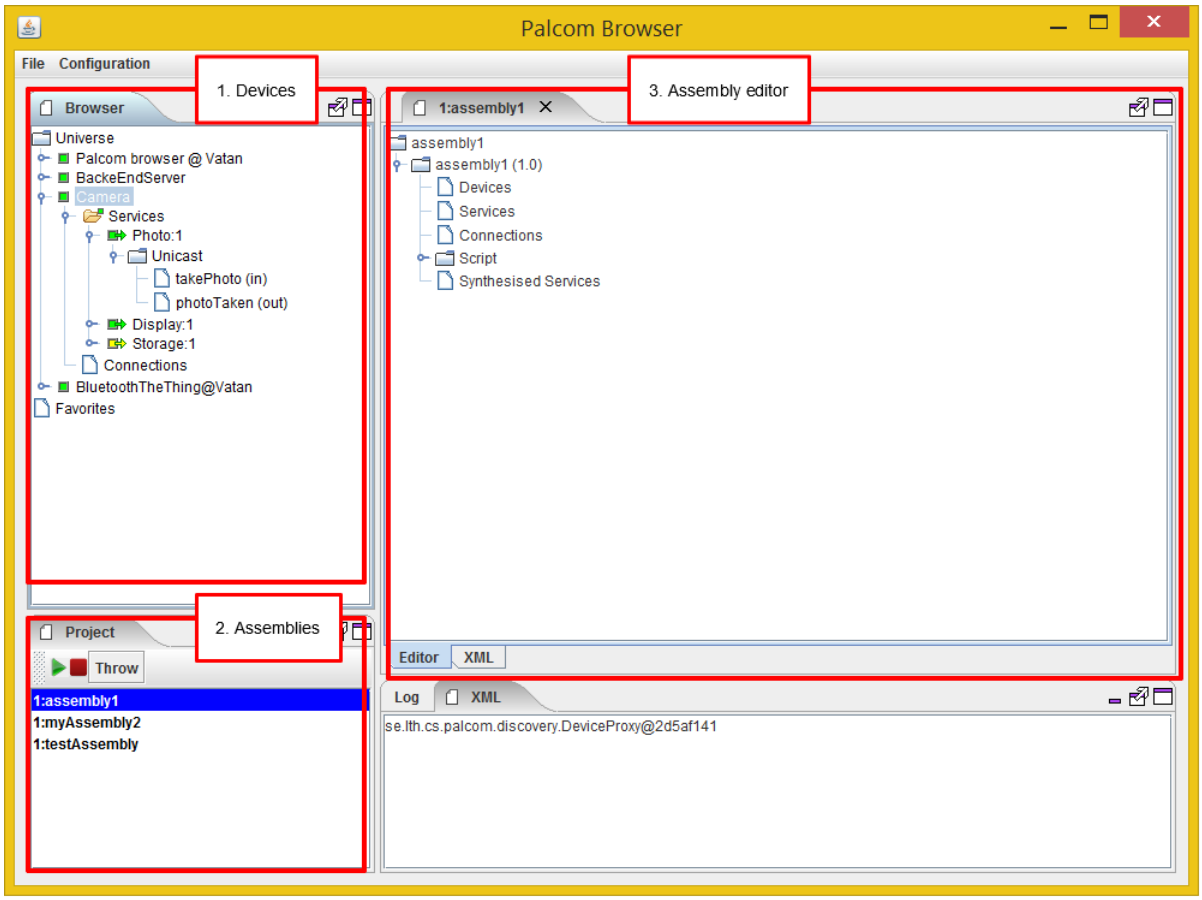

Figure 2.2: The BrowserGUI with three major parts; 1. Devices, 2. Assemblies and 3. Assembly editor.

Chapter 3
Previous work

In this chapter some existing tools will be described and in what way their design approach can be used to solve the defined problems from chapter 1. The mentioned work here will set the base of the solution in the next chapter.

A definition of the main parts in the current assembly editor that the new solution needs to support will initially be presented. Followed by examining the old PalCom overview Browser and an ABB editor to see what can be useful to the new solution for solving the stated problems. Furthermore, the design approach (IfThisThenThat) will also be examined in order to extract what can be useful for this thesis.

3.1 Assembly editor

The current PalCom editor will be stripped down in order to find some core parts that the new editor also must support to be useful in real situations.

A Palcom Assembly is represented in XML format and is also managed and saved in that format. The existing Assembly editor have two different ways to view an assembly. The assembly can be viewed as a tree, as seen in 3.1 , or directly as the underlying XML code as seen in fig 3.2. The tree view of the xml provides a better overview than the xml view but the xml view provides more flexibility to the user. However, with the downside of increased risk of error.

The assembly consists of five main parts; Devices, Services, Connections, Script and Synthesized Services as seen in fig 3.1.

3.1.1 Devices

The device section of an assembly consists of every Palcom device that are to be used in that assembly. It contains the name of the device as well as its device id. The underlying XML for an example device has the following layout.

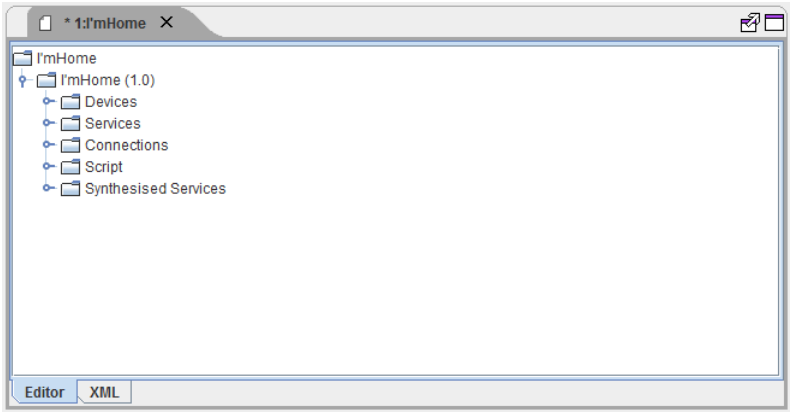

Figure 3.1: An overview of an assembly's tree view in the current PalCom assembly editor

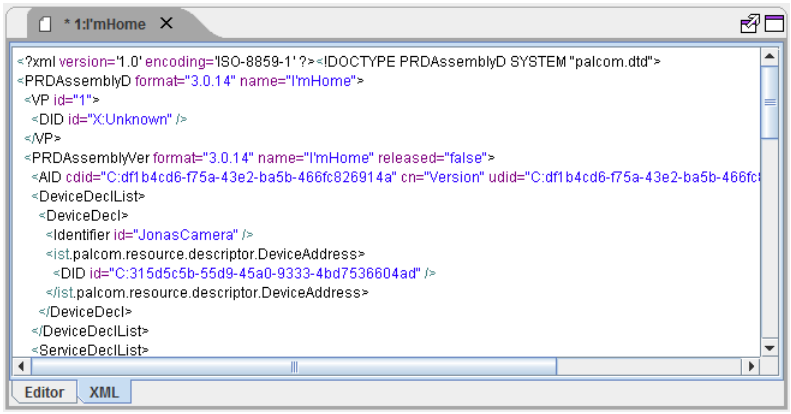

Figure 3.2: An overview of an assembly's xml representation in the current PalCom assembly editor

```
<DeviceDecl>
  <Identifier id="GeotaggerCamera" />
  <ist.palcom.resource.descriptor.DeviceAddress>
    <DID id="C:43749ba6-6495-4e9b-b028-120374182845" />
  </ist.palcom.resource.descriptor.DeviceAddress>
</DeviceDecl>
```

3.1.2 Services

The services node contains all services that are used in the assembly. Each used service is assigned a unique id within the assembly for references of the service in the connections and eventhandler described later. The underlying XML layout for a service has the following format:

```
<ServiceDecl>
  <Identifier id="s9" />
  <SingleServiceDecl ruid="false">
    <Identifier id="Photo" />
    <DeviceUse>
```

```
      <Identifier id="GeotaggerCamera" />
    </DeviceUse>
    <SIID in="1">
      <SID cdid="X:1" cn="TF37" udid="X:1" un="TF37" />
    </SIID>
  </SingleServiceDecl>
</ServiceDecl>
```

3.1.3 Connections

In the connections node all connections to the used services are set. The assembly need the connections to enable communication to the corresponding service. A connection is also assigned a unique id within the assembly like the services were. A connection has the following XML layout.

```
<ConnectionDecl cid="conn-2">
  <ServiceUse>
    <Identifier id="s9" />
  </ServiceUse>
  <ThisService />
</ConnectionDecl>
```

3.1.4 Script

The Script node contains variables that are used within the assembly as well as eventhandlers.

Script - Variable

The variable node inside script contains all the local variables which are to be used within this assembly. A variable holds a name and a type. The XML layout for a variable has the following format:

```
<VariableDecl type="image/jpeg" identifier="imageVar" />
```

Script - Eventhandlers

In the handler node all event triggers are handled and it is defined what corresponding actions that should be triggered.

An event can be triggered by either a service from a device or a synthesized service (described later) within the assembly. The triggering command can contain zero or one parameter. A trigger command has the following format:

```
<CommandEvent commandName="photoTaken">
  <ServiceUse>
    <Identifier id="s9" />
  </ServiceUse>
  <CmdI id="photoTaken" help="Signals␣that␣a␣photo␣was␣taken" direction
      ="out" commandNumber="2" />
</CommandEvent>
```

In this case it's a trigger without a parameter from a service in a device. The other types have a slightly different appearance.

There are three types of actions that can be triggered from a commandevent. These are the following:

- **AssignAction** that assigns a value on a variable from a parameter in the commandevent. The AssignAction contains a variablename to be used as well as a parameter name which refers to the parameter name from the triggering command.

```
<AssignAction variableUse="imageVar" paramUse="img" />
```

- **MessagAction** which triggers a command on a service. MessageAction contains a commandname, a serviceid and optionally a ParamUse or VariableUse. Commandname refers to the command which should be triggered in the used service which in turn is refered by the internal service id in serviceUse. ParamUse or VariableUse is added if the command takes a parameter. ParamUse refers to the param from the triggering command and VariableUse refers to a variable in the current assembly where the value will be fetched.

```
<SendMessageAction command="storePicture">
  <ServiceUse>
    <Identifier id="s12" />
  </ServiceUse>
  <ParamUse name="img" />
</SendMessageAction>
```

- **InvokeAction** triggers a command on a synthesized service. InvokeAction is much like the MessageAction but is used for synthesized services instead of a service from a device. It contains a SynthesizedServiceUse which refers to a synthesized service and a command that refers to a command in that specific service. InvokeAction also contains an optional ParamUse or VariableUse that works exactly the same as above in the MessageAction.

```
<InvokeAction command="img" addressingType="">
  <SynthesizedServiceUse>
    <Identifier id="synth2" />
  </SynthesizedServiceUse>
  <VariableUse name="imageVar" />
</InvokeAction>
```

3.1.5 Synthesized Services

This section contains all synthesized services in this assembly. A synthesized service is a virtual service with its own commands which are defined by the assembly. The synthesized service contains an id, commands and groups which contains other groups and/or commands. The xml layout for a synthesized service has the following layout:

```
<SynthesizedService distribution="1" RemoteConnect="false">
  <SD id="synth2" help="">
    <CmdI id="img" help="" direction="in" commandNumber="1">
```

```
      <PI id="image" help="" type="image/jpeg" dataRef="0" />
    </CmdI>
  </SD>
</SynthesizedService>
```

3.2 PalCom overview browser (OVB)

A similar attempt, to obtain an overview of how the assemblies are set up and the connection between the devices, has been made previously. But it has not been made as an editor for the assemblies but as an overview when all the assemblies are running [13]. This browser has not been continued and is unfortunately not compatible with the latest PalCom version, v3.1.12. However, a summary of it will be made in order to examine and select the parts that will be useful for the new editor.

The OVB focuses on bringing a subset of the overall system and the developing tools for PalCom to a user group that is non-technical. It allows users to inspect the topology and some internal states of devices that are connected to the PalCom network. The browser itself is a PalCom device with an assembly manager that maintains one assembly. This assembly contains two services; `Swing Display Service` and a `Visual Browser Accomplice Service`. These services will then provide the solution for updating the graphics in OVB in real time. Figure 3.3 shows the graphical user interface of OVB and the relationship between the services and the assembly that makes it possible for the OVB to work properly and update all the connections in real time.

Figure 3.4 shows the overview of one example where three devices are used; `My server`, `My GPS`, `My camera`. These device are then connected to `My laptop` through an assembly. The connected services on each device are shown by lines going from one end to another. If one thinks of a workspace where there are objects on the table and requires to connect these to each other, they could be wired together. This metaphor [12] can be used on the OVB as well where there are devices that are connected to each other with some lines.

3.3 Control Builder Diagram Editor - ABB editor

The Control Builder Diagram Editor[10], CBDE, is a tool used by ABB [1] for their own product, AC 800M, which is a Programmable Automation Controller, PAC. A PAC is a controller used in the process industry to handle digital, serial and analog signals from different industrial applications and then process that data into some PID algorithm to control some process in the manufacturing line. The same hardware can also be used in other fields and not just for process control, but for collecting data, remote monitoring etc.

A PAC uses standard open protocols such as Ethernet to communicate with the different applications. Ethernet is something most enterprise computer have support for. This makes it easy for PACs to be integrated in already running systems.

The editor that ABB has for their PAC is a GUI running on Windows. This GUI is used to create and monitor control systems for the process industry. Moreover, the editor

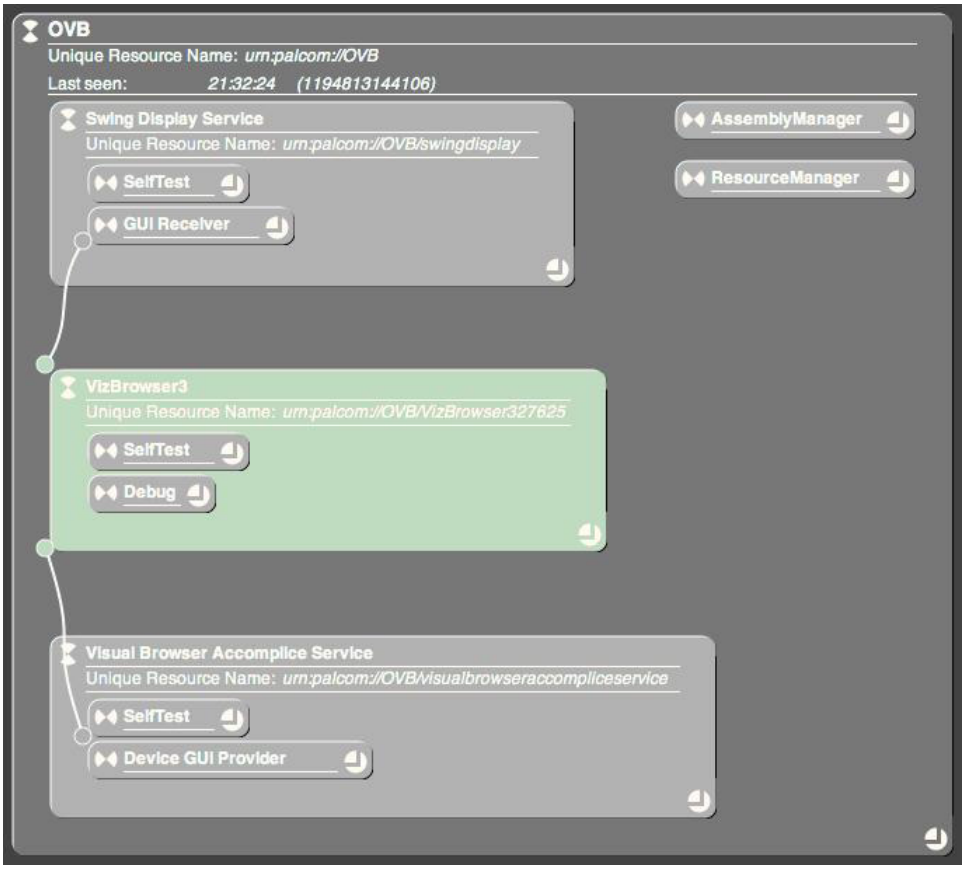

Figure 3.3: The overview browser showing the topology of the OVB device. It shows how the Swing Display Service and Browser Accomplice Service are connected through an assembly, VizBrowser3

uses Block Diagram[17] that represents the different functions and algorithms as blocks and the relationship to other blocks are represented by lines through the ports that can go in or out, see figure 3.5.

The diagram editor has a workspace on which users can drag and drop objects. Users can drop the same object multiple times on the workspace to arrange interconnectivity among objects and each object on the workspace can show different ports going in or out to save space and make it easier to understand. See figure 3.6 that shows how the same device is displayed on two blocks and with different ports visible.

3.4 If This Then That (IFTTT)

If This Then That (IFTTT) is a design approach that simplifies for non-programmers to do chains of conditional statements. This is similar to an IF-statement in all common programming languages. Today IFTTT is widely used in mobile applications were there can be a web services that connect other services on your smart phone to do tasks when some event occurs, e.g. If I post a picture on Instagram, save the photo on Dropbox as seen in fig 3.7.

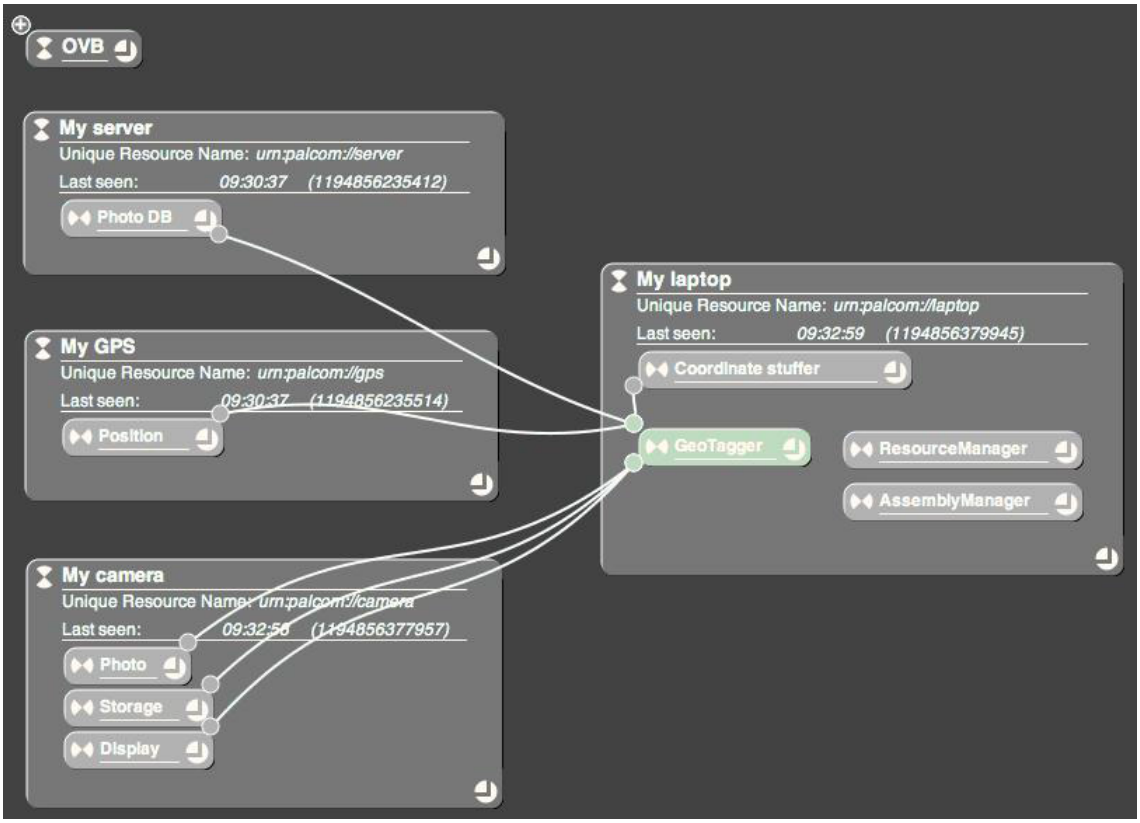

Figure 3.4: The overview browser showing an example of three devices (My server, My GPS, My camera) in the PalCom network connected through one assembly running on another computer (My laptop).

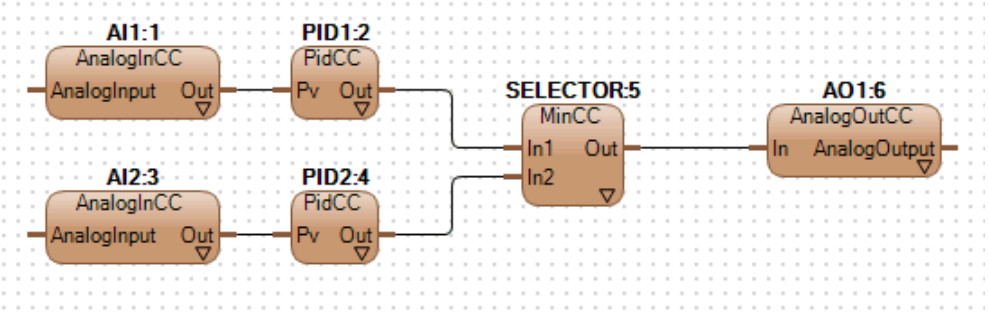

Figure 3.5: ABB editor with Block Diagram where relationship between blocks are represented by lines.

In the current PalCom editor we have the same conceptual model where services and commands on services are triggered when some condition is met in a state. Instead of using an IF-statement, which tells us that it may or it may not happen, the PalCom assembly editor uses a When-statement, which tells us when the condition is met to do something and as result invokes some action. These actions are managed by the assembly editor and some parallels can be drawn between IFTTT and the contents of the assemblies. The *THIS* part in IFTTT is the same as the triggering event in the assembly and *THAT* corresponds

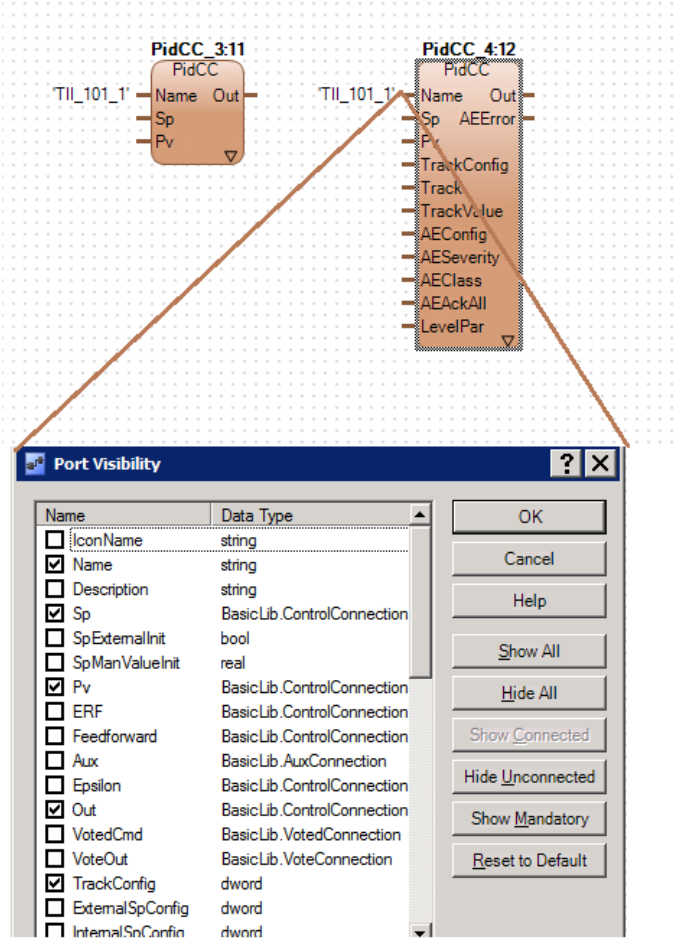

Figure 3.6: Two blocks for the same device but with different ports that are visible.

Figure 3.7: If This Then That example where instagram photos gets saved to dropbox.

to the actions which in Palcom can contain multiple actions instead of single one as in IFTTT.

Chapter 4

New assembly editor with implementation

In this chapter the emerged new solution will be presented, as a result from the observations in the previous chapter. A description of the features of the final solution is set out, together with the reason behind the made design choices.

Furthermore, a detailed description is provided on how the converting from a regular XML description of an assembly to a graph representation in our editor is done and vice versa. This detailed description will be helpful when future research is conducted on improvements of the new solution.

4.1 Assembly editor

The new solution must solve both of the defined problems, laid out in the beginning of this thesis:

1. It is difficult for non-technical users to create an assembly without a proper introduction.

2. It is impossible to derive an overview of how devices are connected in a PalCom assembly.

To make it easier to create an overview of how the services are connected with each other, i.e. solving the second problem, a solution that behaved like the OVB is intended. The OVB has been used in previous PalCom versions and has fulfilled its purpose.

To solve the first problem a conceptual model of how people would think when connecting different devices with each other had to be created. The conceptual model that was used in the OVB and ABB editor follows the natural description. By using a workspace

with devices as objects and allow users to drag lines between commands, thus creating an experience close to the real world.

The functionality that the previous assembly editor had was taken in consideration whilst the new solution emerged. This resulted in a GUI, see figure 4.1, consisting of three major parts:

- The workspace
- Variable handler
- Synthesized service handler

These will be explained in detail further on.

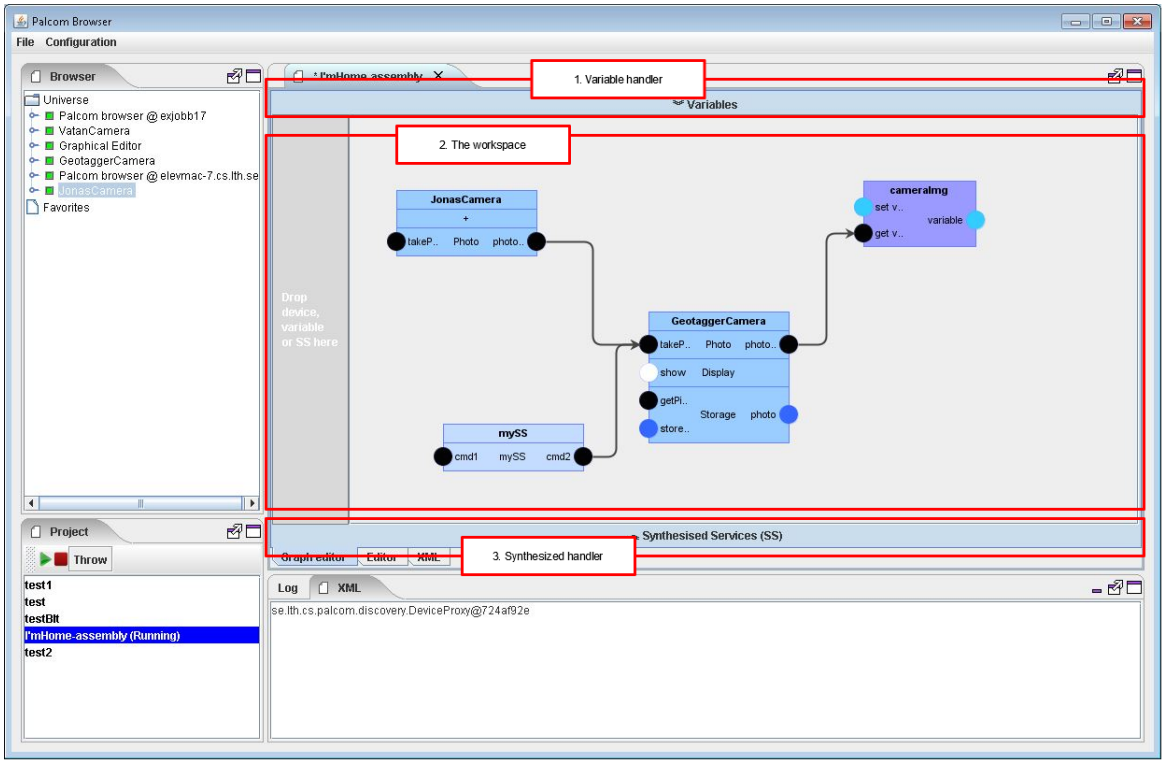

Figure 4.1: Our solution for editing assemblies and obtaining an overview of the connections. Section 1 contains the Variable handler. Section 2 is the workspace to make connections. Section 3 is the handler for synthesized services

4.1.1 The workspace

The whole solution was thought of as a workspace area where the user grabs devices and drops them on the workspace to establish interconnectivity. This enables a natural mapping to a real physical workspace, where you have your physical devices and connect cords in between to make some sort of connection. In this new solution the connection between a service inside a device to another service on anther device was made by simply dragging

a symbol from one command to another. This results in a better and simplified overview of the complete system, compared to the existing solution where it is difficult to see how the system is connected when there are many devices.

By dragging the devices, variables or synthesized services on the drop area a graph object will be created and displayed on the workspace. Because of limitations in the software and libraries a specific drop area had to be used for the objects, see figure 4.2. This is less intuitive compared to having the whole workspace as a drop area. All the devices will begin by not showing any service and the users will have to press the + sign to see all the available services. The users then choose which services they need to use in this assembly and all the commands will then be displayed in that graph object.

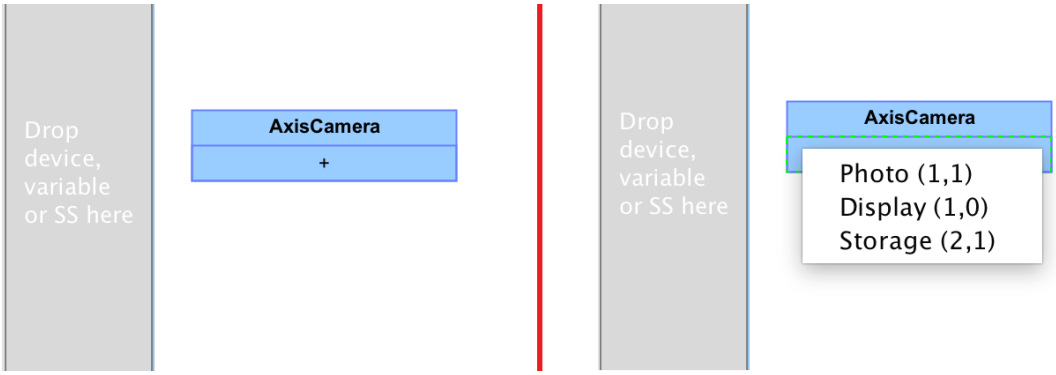

Figure 4.2: Drag device and drop them on the grey marked area where it says drop device, variable or SS here. All the devices have their services hidden and the user needs to press the + sign to choose which services to display.

All the commands that go in to a service are on the left side of the object and all the commands that go out are on the right side of the object. Each command is displayed with a name and a circle with a specific color of what type of parameter that command can obtain or send out (the color description can be found in 4.1.2). To connect a command that goes out from a service to another command that goes in, the user drags the circle from out command to in command. See figure 4.3 of how two devices are connected to each other. If the connection is supported (the parameters are of the same type) a line will be fixed between those commands, otherwise no line will be fixed and the circle will be marked with a red border to indicate that this connection cannot be made.

We have three different objects that can be on the workspace; Devices, Synthesized Services and Variables. To identify an object on the workspace, their name is displayed on top in bold and each object type has also their specific color. This helps the user to be able perceive a quicker overview and separate the devices from synthesized services and variables. If a device is not available in the network, but it is used in the assembly, it will be displayed as a dark graph object whilst as all the commands and connections still remain on the workspace. See figure 4.4 for an overview of these types.

A variable has three different commands, see figure 4.5. Two of them that goes in are `set variable` and `get variable` and one that goes out is `variable`. The `set variable` is used to set a value and the `get variable` is used to trigger when something happens get this variable and send it to some other command. The command

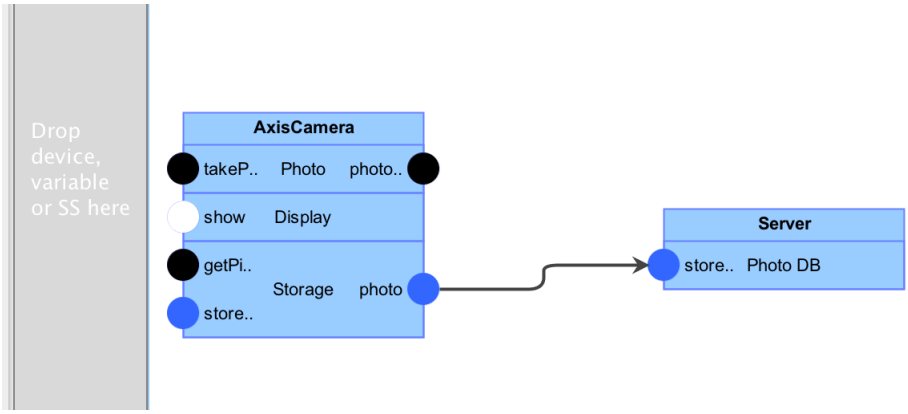

Figure 4.3: This shows how the AxisCamera has three different services visible and that it sends a photo from its photo command to the store command on the Server, which only has one service available, Photo DB

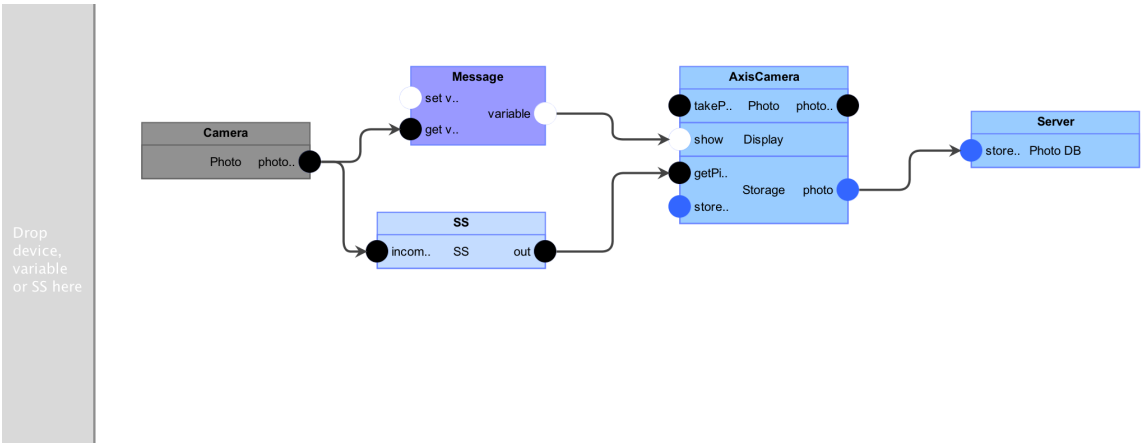

Figure 4.4: This shows how graph objects display different colors depending of their type. From left is a Camera device that is not available on the network but used in the assembly. On top, is an object that is a variable with the name message. On the bottom, is an object that is a synthesized service named SS. In addition there is an AxisCamera device and last a Server device.

`variable` that goes out will only get the value of the variable and pass it over to a command.

JgraphX

The workspace could be created by either developing a new graph tool or by using an existing tool. The latter was chosen due to the time to create a custom graph tool would be to immense to fit within this thesis. After trying out several different graph visualisation libraries (JGraphT, JGraphX), the JGraphX[5] was chosen since it provides the best support and documentation and is the one which was updated most recently. JGraphX is a Java Swing graph visualisation library which is platform independent as well as open source

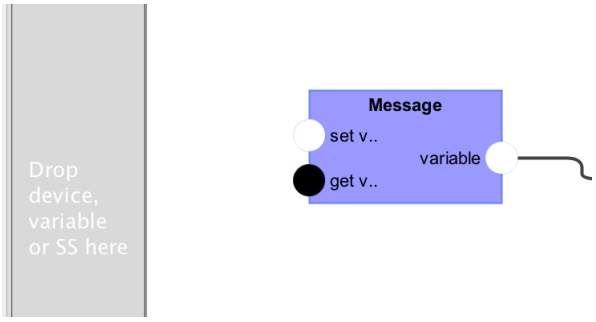

Figure 4.5: This shows what a variable looks like when it is displayed on the workspace. The three commands can be used to set and get the variable.

licensed under the BSD license. It is a library for visualizing and interacting node-edge graphs by creating graph objects and adding coordinates for them to place them on the workspace. It supports many default functions for visualizing different blocks and connections. Our solution is highly dependent on this library and we have had the ability to customize it for our purposes.

4.1.2 Variable handler

This area of the GUI, see figure 4.6, is required to make the variables that are used in an assembly. A variable consists of a name and a type (like in a programming language). We have defined some standard mime types that are used regularly in services and colorized these in different groups.

- Image types are colored from the blue spectrum
- text types are from black/white area
- audio types are in green
- video types are in red/purple.

When a type, which is not defined in the standard list is used, a random color will be picked. The predefined color description is always available in the variable area to help users understand what the colors are used for. When a new variable is created it will get assigned a color that is defined in the mentioned list. To use a variable (to assign it a value and read it) the user drags it from the list and drops it on the workspace.

To assign a value to a variable users will only drag the symbol from a service to the variable. Then that service output will be assigned to that variable.

4.1.3 Synthesized service handler

The Synthesized Service handler, figure 4.7, is very similar to the Variable handler regarding the design. The SS handler is also hidden initially not to confuse the user with to much functions and buttons. These synthesized services are also more likely to be used by the more advanced user and therefore does not need to be visible initially.

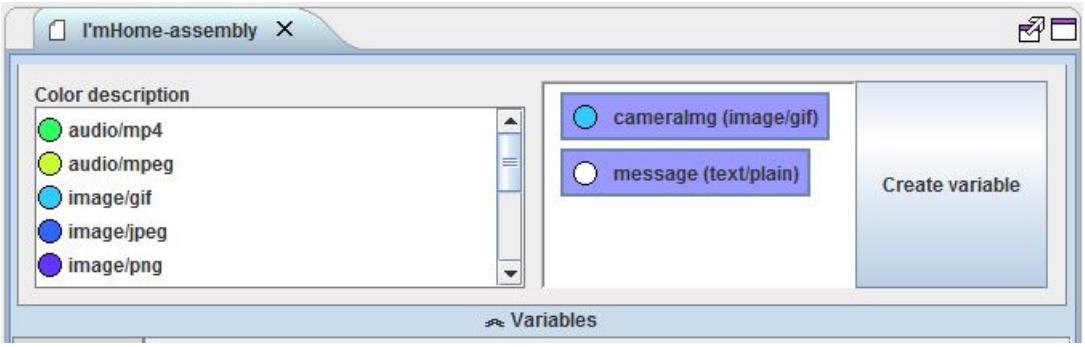

Figure 4.6: The variables are hidden and displayed by clicking the big variable bar. A description of what the colors mean and the ability to create a variable for later use in the workspace can be found here.

The handler consists of a list where all the created SS are available for that assembly and a button where the user can create new synthesized services. An SS can be very complex since it can have many commands in a deep tree structure and this is solved by creating an object of an SS that contains the same structure as the previous browser so that the more advance users will feel familiar with the new design.

To use an synthesized service the same approach as in the previous areas has been followed, where the user drags and drops things to the workspace to make the connections. This is also done with the SS and when a user drops the service on the workspace they will see the available commands for that service and initiate the connections.

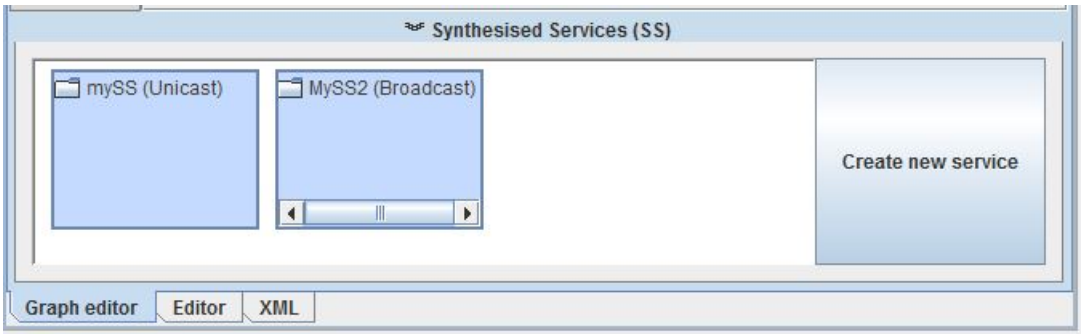

Figure 4.7: The services are hidden and displayed by clicking the Synthesized Service bar. Here it is possible to create new services. The layout of a service is similar to the previous assembly editor where a tree structure is used.

4.2 Services

To use the devices that were required according to the scenario, they were initially introduced to the Palcom network since they are not directly built for this. A service controller for Philips TVs and a service for the Linocell shutter device was created. The services for

Tellstick were imported from a previous project, with a minor modification to simplify it for the users when conducting the experiment.

4.2.1 Philips TV service

This PalCom service can handle multiple Philips TVs, but to add a Philips TV one has to do this manually with the controller since the service does not support automatic discovery of Philips TVs. The controller has three methods that can be invoked:

Add TV takes in an IP as a parameter and then adds that Philips TV that is referred by the IP.

Remove TV takes in an IP as a parameter and removes that Philips TV that is referred by that IP.

List all TVs lists all the Philips TVs that has been added.

All the added TVs will then be displayed on the network as separate devices with unique DeviceIDs consisting of their IP. To use the Philips controller service one has to load the jar-file into TheThing that will run the service. All the added TVs will be stored in the file system so that every time TheThing is started all the TV devices will appear on the network.

The Philips devices that are automatically created by the controller have the following commands that can be invoked:

- **Favorite channel 1**
- **Favorite channel 2**
- **Favorite channel 3**
- **Favorite channel 4**

These commands will set the TV on the specific favorite channel.

4.2.2 Tellstick service

This service consists of a controller service that will find different tellstick devices (just like the Philips service). It is loaded by adding the jar into the TheThing that will run the service. To add devices the tellstick manual[2] has to be followed.

The difference from the previous tellstick service is that when a tellstick device is visible it will not show up as a service in TheThing but a separate device will be displayed on the network just like the Philips TV. This will simplify it for the users in the scenario experiment since they will see every tellstick device as a separate device in the network and not as services under TheThing. In figure 4.8 the difference of the result that our changes made is visualized. The left view is the old tellstick service and the right view is the new one.

The tellstick devices have many commands but the ones of interest are:

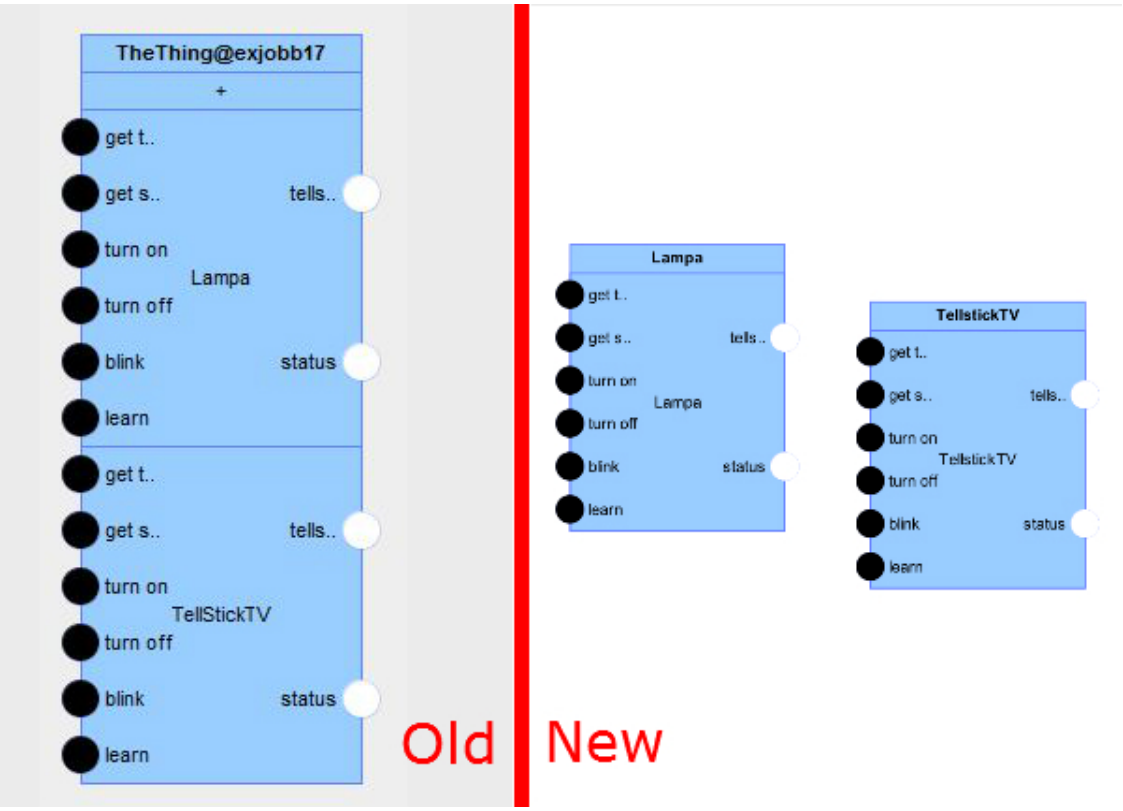

Figure 4.8: The old tellstick service added each tellstick device as a service in TheThing. The new service creates each device as a separate PalCom device

- **turn on**
- **turn off**

These are the only commands that will be used in the scenario experiment for the two tellstick switches.

4.2.3 Linocell bluetooth camera shutter service

This is a single Bluetooth service that will interact with a Linocell Bluetooth camera shutter. It is loaded by inserting the jar-file into TheThing that will run the service. The camera shutter is then displayed as a service under TheThing that is running its service. Due to problems with paring bluetooth devices in the service, the service had to be made to listen on key presses on the keyboard since the operating system running TheThing did connect the camera shutter as a bluetooth keyboard. In other words the camera shutter service was listening for the key presses `Return` and `Volume_up` and when the camera shutter was not connected to bluetooth the service would be unavailable in PalCom.

The service has two commands that go outwards:

- **button1pressed**
- **button2pressed**

These are the event triggering commands that will be listened in our scenario.

4.3 Graph implementation

In this section it will be described how the editor is built and how the different java-classes are dependent of each other to help any future developer that is going to make any improvements. Figure 4.9 shows an overview of the classes in this solution.

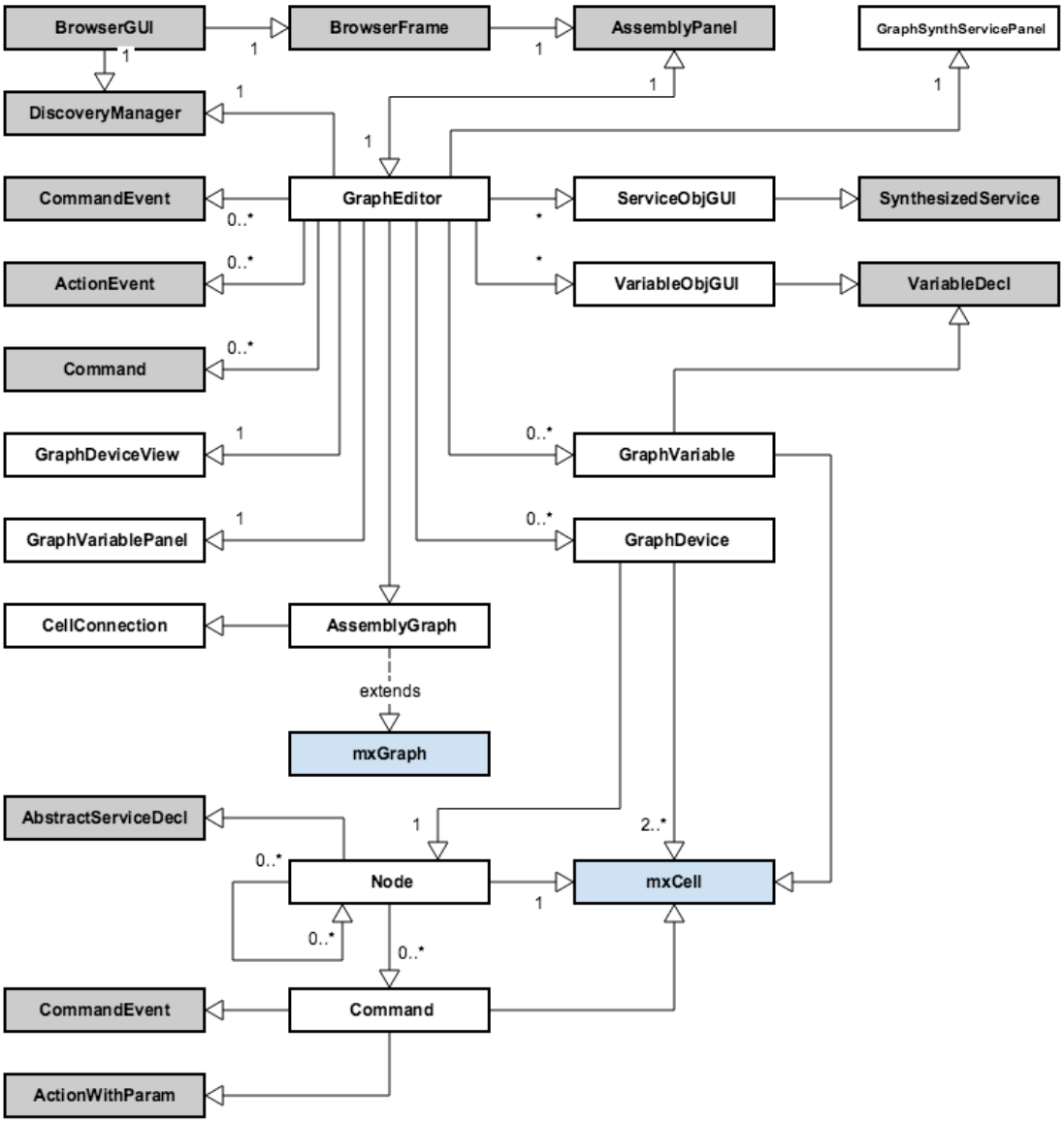

Figure 4.9: An overview of the java classes used for the graph implementation. The gray ones already existed in Palcom, blue ones come from the Jgraphx library and the white ones were created for this solution.

4.3.1 Java classes used from Jgraphx

mxGraph The graph which AssemblyGraph extends. It represents the graph that is rendered in the workspace and manages all interactions with the graph.

mxCell Represents a single object in the mxGraph. A device, service, command and command-connection are all represented in the GUI by a mxCell.

4.3.2 Java classes from Palcom

The following existing classes in Palcom were used or modified to support the graph implementation.

AddSynthServiceDialog This class was modified so it has support for opening the dialog window for adding synthesized services in the graph editor as well.

AssemblyPanel One of the main classes for the complete assembly editor. This class handles all the switching from the graph editor to the tree editor and to XML view.

4.3.3 Java classes created

The following classes were created to add the graph functionality to the assembly editor. Each class has a short description of its purpose. The UML over the classes can be seen in fig 4.9

AssemblyGraphTransferHandler This class handles the drag and drops from the palcom tree containing all available devices. It passes the object on to the GraphEditor which in turn adds it to the graph.

AssemblyGraph This class extends the mxGraph from the jgraphx package. It holds all graph information printed in the workspace, devices, variables, synthesized services and events between these. AssemblyGraph also contains an important method that translates a palcom assembly from a xml string to a graph object with all nodes and connections belonging to it.

CellConnection A cellconnection is a connection in the graph and represents an event in the Palcom assembly. This object is used within the AssemblyGraph to create a connection between two nodes in the graph from the event parsed from the xml string.

GraphDevice A GraphDevice is a graph representation of a Device or a Synthesized service. It holds the graph object displayed in the GUI as well as a root Node containing the Service(s) in the device/synthesized service. This object is rendered differently when it's a synthesized service or an online device or an offline device with some color variation. GraphDevice objects are created from the GraphEditor when a user drags a device or synthesized service into the drop area of the workspace. They are also created when AssemblyGraph are parsing the used devices and synthesized services from the xml string.

Node The Node object are stored inside a GraphDevice object and contain nested Nodes as well as in and out commands. They are the same as a PalCom service. Thus, containing AbstractServiceDecl as well as the assemblies service id.

Command A command is a representation of a Palcom command and is either an in or out command. They are stored inside the Node objects corresponding to the service which the Palcom-command belongs to. These are the objects which the user drags lines in between to create an event. Commands are created when a Device's service is displayed from the add menu or when an assembly is loaded in AssemblyGraph between the different events.

GraphDeviceView GraphDeviceView is used to render a popup menu where the user can select some options. It is used when the user wants to add more services from a device by clicking on the + button inside the device in the graph GUI or when right clicking a service or device to remove them from the graph GUI.

GraphEditor The GraphEditor class is the hub of the graph GUI. The interactions between the graph GUI and PalCom are all passed through this class. It contains all objects which are rendered inside the graph as well as the variables and synthesized services inside the current assembly. It manages what colors are to be used for the known and unknown connection types. It creates all objects that needs to exist for the graph to be rendered which includes AssemblyGraph, GraphDeviceView and the panels for management of variables and synthesized services. It creates devices when a user drags them into the drop area as well as services and connections that the user wants to add. Lastly it creates the XML representation of the graph needed when changing tabs of the assemblymanager.

GraphObjectPositioning This class contains the positioning data for the GraphDevice and GraphVariable objects. The data is saved in a new file beside the original assembly file to reduce the chance of error if opened in an older version. When opening an old assembly lacking this information a counter is used to render the graph objects on different positions to simplify reading.

GraphObjectsHandler This class is used to parse and create the xml string for all GraphObjectPositioning objects. It contains all current position for the graph objects. GraphObjectsHandler is initiated from the AssemblyPanel of Palcom before the GraphEditor object is created.

GraphSynthServiceMenus This class is used to create popup menus for the different parts in each tree node when creating a synthesized service.

GraphSynthServicePanel This is a JPanel for handling the synthesized services on the GUI.

GraphVariablePanel This is a JPanel for handling the variables on the GUI.

4.3.4 Xml to graph

The parsing from the xml-string to a graph is done inside the GraphHandler object. The parsing is done from an existing method parseAssembly inside the class OldschoolAssemblyLoader. The .assgraph file (which holds all the positions of every object on the workspace) is also parsed containing all devices, synthesized services and variables positions in the graph. The parser first retrieves all devices and for each one of them searches the PalCom network if the device is online and adds it as a flag to the GraphDevice objects. If the device is offline the whole assembly is iterated for all used services and connections for this device. Otherwise, the device is added the same way as when a user drags it into the droparea with the additional position information.

The variables are iterated through and if it has a position information from the .assgraph file it's added to the graph workspace.

The services are all iterated and added to a list containing all services that are going to be extended inside the graph GUI.

All synthesized services are looped through and just like the variables, added to the graph GUI if they have position data from the .assgraph file.

Last the events are iterated. The events consist of a command event that is the trigger and one or more actions. The graph will create a line from the command event to each of the actions unless the action uses a value from a variable. If that is the case a line is drawn to the `get variable` command on a variable and then another is drawn from the `variable` output to the action.

4.3.5 Graph to xml

The generation of the xml-string from the graph workspace is done inside the GraphEditor object. It loops through the variables and synthesized services from the respective panels and adds them to the `PRDAssemblyVer` object. `PRDAssemblyVer` is a representation of an assembly that can directly be saved as a xml string. For every device shown in the graph workspace they are added to the assemblies devices. For every extended node (aka service) the service is added to the assembly. For every command in the node that have a connection to another command, an event is created with a command event in the assembly. This also creates a connection for each service used. For every action which points to `get variable` on a variable an action event is created for each command connected to the `variable` output from that variable.

Chapter 5
Evaluation

In this chapter we will evaluate the solution described in the previous chapter. The questions that needed to be answered in the theses were:

- How can non-technical users create an assembly without a proper introduction?
- How can PalCom users get an overview of how devices are connected in a PalCom assembly with the same solutions as in first question?

The chapter will begin by discussing how the solution answered the second question (getting an overview). This is due to the fact that to solve both of the problems with the same solution - a solution to the second question must be found first.

To evaluate the solution regarding question one an experiment with the two editors will be performed on users who have never used PalCom and don't know how the editors work. This is the user group that are not PalCom developers. Thus, it will be an evaluation to see if the first problem is solved. At first we will describe the setup of the experiment and then an evaluation on the findings will be conducted.

5.1 Overview

The assemblies are created by dragging and dropping devices on the workspace and the altered solution is able to load already created assembly scripts to the workspace and display the devices accordingly as if the assembly was created by the new editor. This means that support also is provided for already created scripts to be displayed in the new editor. The connections are represented by a line between the two services that are communicating and the users only need to drag from one command to another to make a connection. This is very similar to getting an overview with the OVB and the ABB editor since it is very intuitive for the users. A difference of the OVB compared to the new solution is that it shows in detail what is connected in the assembly. The OVB displays that some services

are connected to an assembly but not in which way and what commands are used to accomplish this. The new solution displays the devices inside an assembly while the OVB displays the devices outside the assembly but connected with lines. This implies that the solution gives an overview of the assembly itself while the OVB gives an overview of how devices are used in the PalCom network, regardless of one specific assembly.

In ITTT it is easy for users to define what they want to do when some event occurs but the problem is getting an overview. The current PalCom editor uses some kind of ITTT design but it has been shown not to be sufficient for the more advanced PalCom users when they create larger and more complicated connections. Therefore, this solution is only based on Block Diagram [17] as in the ABB editor.

To not overwhelm the users with all the device specific data and their services, the solution uses visibility of different parts to be highlighted when they are interesting for the user.

- The users decide which services a device uses and not all of them are directly displayed.

- The manager for handling variables is only visible when the users want it to be visible and same is valid for the synthesized service handler.

This provides an overview of what is actually used in an assembly and does not show the complete available information contained in that assembly. Like the ABB editor this is very good for the users and the difference from the new solution compared to the ABB editor is that the users choose which services they wish to display whilst in ABB they choose exactly which commands/ports because their objects can't be divided into services or groups. A solution where the users had to choose services instead of commands was chosen, because a device can have many services containing commands with the same names. This would be confusing for the end-user.

The devices in the ABB editor can be displayed multiple times as objects in the workspace. But in the new solution one can drag the device into the workspace only once because this is more intuitive for the users that one can't copy a physical device and say that it is equal to the other copy. In the ABB editor they choose to have the ability of multiple copies of the same device because it can simplify the overview of the connections. If there are several connections going from one device to several other devices it can be favorable to divide the device into two objects so the lines don't cross all over the workspace.

Another improvement to obtain a faster overview of what exactly is happening in the assembly is by using different shapes and colors. In the new solution all the objects are rectangular like a class diagram in UML [8] and all the commands are circles with different colors that tells what type they return or need as parameter. With this, one can get a quick overview of which commands can interact with each other. In the ABB editor the same concept is used with Block Diagram and different colors for the type of objects. In contrast they haven't chosen to use different colors in the ports that can communicate with each other. This requires more knowledge from the user where he must know what connections can go where instead of assigning the tool to handle that. In the long term by assigning the tool to handle the connections, it reduces the risk for conducting any errors.

5.2 Create assembly

By conducting the experiment on persons with no previous experience of PalCom, data is obtained to answer if the new solution works for question 1. An experiment on test persons who had to try both of the editors was conducted. First the test persons began with one editor to accomplish the task and when they felt done they switched to the other editor where they had to accomplish the same task again. Both of the accomplishments were timed and the results provided two types of data:

1. Data for how long it takes to accomplish the task with each editor

2. Data for how long it takes to go from editor 1 to editor 2 and vice versa, from editor 2 to editor 1.

This data enables evaluation over the editors with two different approaches which is more accurate than only using one.

5.2.1 The scenario setup

The same computer with the same screen resolution on all our tests was used. All the devices and their services were thoroughly explained, see appendix A. The test subjects had time to familiarize with the physical devices prior the experiment began so that they felt comfortable with everything. In addition, this gave all the test subjects the same starting point, none having more necessary information than the other to carry out the test.

When they felt comfortable with the hardware the Palcom Browser was explained. Only the common parts of the solution and the current GUI was explained so that the measurements only involved the different GUIs and not the complete PalCom system, see appendix B for the description.

Lastly the scenario 1.4.2 that they had to accomplish with the introduced devices and browser was explained. A test subject was first instructed to only use either the new solution or the current one. Then the test subject had to accomplish the same task with the other editor as well.

Before the tests began, all devices were properly inspected and assuring they were connected and functioning.

An internal PalCom network was created, disabling other devices to appear during the test in order not to confuse the test subjects. Two computers were used to perform the scenario, with the reason that the bluetooth device was not functioning properly on the computer where everything was run. See figure 5.1 for an overview of the connections.

Computer 1 was a MacBook. This computer ran TheThing with the PhilipsControllerService and TellstickControllerService and the BrowserGUI. This is the computer that the test subjects used to edit the assemblies.

Computer 2 was a Windows and it was running TheThing with a bluetooth service for the buttons.

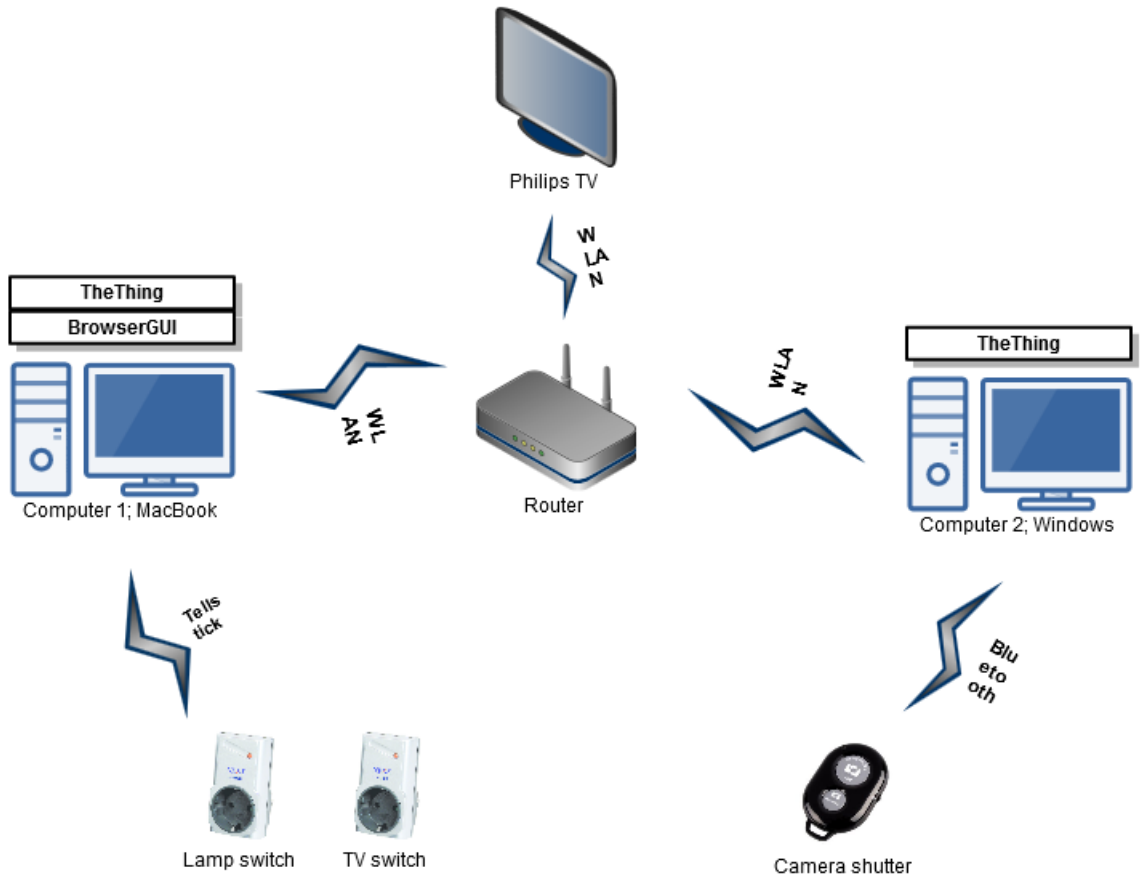

Figure 5.1: The overview of how all the devices to create the PalCom network were set up.

5.2.2 Scenario experiment

The experiment was conducted on three different users U1-U3. This was sufficient for spotting the most problematic usability problems, but to few for collecting any metrics. The most usability problems will emerge after five tests[7] but since we did see a pattern already after three we choose not to continue with more tests.

The results in table 5.1 show how long it took for each user to accomplish the task with the different editors. Two users, U1 and U3, did the experiment by starting with the new editor. All of them, U1-U3, failed running a working assembly when they felt done with the task using the previous editor. The reason why not a single user managed to create a working assembly was because they all forgot to add the services inside the connections directory. This is a step that one can't figure out by himself and further explanation is required. Only by using this statement, that not a single assembly could run using the previous editor and all the assemblies did function properly using the new solution, is enough to draw the conclusion that the new editor does solve problem one. In all the test cases the timer was stopped when the user felt that the scenario was completed.

The results clearly indicate that the time it takes to create the assembly to accomplish the task is faster with the new editor by all the users regardless if they began testing the new

editor first or not. One can argue that if the test begins with one editor the other editor will have a slightly greater advantage since the user already knows how to use the complete BrowserGUI. Thus, it can be argued that it will go faster to finalize the test. This can be the case here but it does not affect the result, since the test was performed significantly faster with the new solution regardless of starting editor.

The time difference between the users when using the new editor is mainly due to how fast the user understood that they had to drag the devices onto the special drop area in the workspace. When this was known to the testers they all took about the same time to complete the scenario.

Besides the fact that the users didn't manage to create a functioning assembly with the previous editor we think the editor performed well also for users that had not used PalCom before. It uses a similar method to ITTT (with some modifications) and people seems to be comfortable with that kind of thinking and solved it better than was initially anticipated.

Table 5.1: Result of the measured times in minutes for accomplishing the task.(S = started with this editor, F = Failed running a working assembly which the user considered was completed)

User	new solution	previous editor
U1	10:23 (S)	15:05 (F)
U2	2:53	11:52 (SF)
U3	5:57 (S)	12:32 (F)

Chapter 6
Discussion

This chapter will be discussing the findings, if the study can be carried out in some other way and if the conclusions can be altered differently using the same result but with a different approach.

The approach was to observe previous editors to obtain what is useful to be included in the new solution for solving the problems. To evaluate the solution an experiment on three test persons was conducted. The test was a scenario that required to be solved with Palcom.

Considering another approach for getting an overview browser, by doing a research in cognitive science instead, could result in ending up having a completely different solution if the previous editors that we observed didn't follow any of the design principles of Donald A. Norman [12] or the gestalt laws of perceptual organization[15]. Fortunately as the new solution was developed, the principles of *mapping* and *consistency* and the design laws regarding *similarity*, *proximity* and *symmetry* were taken into consideration. Nevertheless, it would be very interesting to conduct an evaluation completely based on interaction design in cognitive science to see if it fulfills the requirements for human interaction.

To obtain an overview based on the previous browsers required thorough understanding of them. What can be discussed here is that we didn't have the actual tools to feel and test, but we did get an understanding of them based on the images and text descriptions that were available. To actually get an understanding of the user experience whilst using these tools it is better to have the physical product for testing, but in our case we could get most of the information we needed from the images because our user experience was not focused on e.g. how smooth a transaction is when moving an object from one area to another. We were more interested in the final view rather than the style effects in between.

For the experiment only three test persons were used. This can thus only result in conclusions that are not comparable but only indicative. The scenario can also be discussed since it is adapted for a smart home that is not complicated. The users had one device with two buttons that triggered some simple events like turn on/off some switch. What if the scenario included some further steps that triggered other events in a smart home,

e.g. when one enters the home the light turn on, but only if it is dark outside, and turn on the TV but only if it isn't weekend because weekends one listens to music through the music system and play video game on the TV. Nevertheless, the target in the scenario were people who haven't used Palcom before and they will most likely not begin with creating complicated assemblies immediately.

Can the scenario be conducted in another field beside a smart home? What if the scenario was something commonly occurring in healthcare and where PalCom currently is being applied? The solution should be field independent just as the current editor is (which the solution is based on). This implies that it shouldn't affect a user that is going to use it for a smart home nor a doctor or nurse that is going to use it in health care. It would however be interesting to use a scenario in the field of health care with the appropriate users to see if the results will be the same.

Chapter 7
Future work

The solution to the PalCom editor is convenient to be used when editing an assembly to obtain an overview of that particular setup. But what if one requires an overview of all the connections of the complete system that contains many assemblies and many services connected to each other? This is something that is currently not possible with the developed solution. However, it can be implemented by using the same GUI. Instead of loading the assembly script for editing, the assemblies that are running at the moment can be loaded. Editing in this mode will not be possible since it will only show the current running state of the system. This can also be taken a step further to help the PalCom users in debugging the connection in the running mode. To visually show the user how the data is sent in real time from one service to another can help them debug when an assembly is not working properly.

This paper can also be evaluated through another perspective. In this thesis the solution has been conducted based on similar graphical interfaces and evaluated with an experiment. The research could be carried forward by the department of Design Sciences and they could do further improvements regarding the interaction design. They could look at guidelines for proper user interfaces and apply them to the Palcom assembly editor and perhaps ascertain why the ABB editor doesn't use multiple colors and how to handle user disabilities such as color blindness in the editor.

It would be interesting to test a different scenario where more advanced PalCom users would have to create an assembly which consist of variables and synthesized services as well to see if this also is simplified.

Further work to be implemented in our editor that the previous editor does support are:

- Disconnected devices should be cached so all their commands are visible even though they are offline.
- Make self. Function so that a service can be found on the running device without specifying exactly which device.

- Set constants to variables. Our solution can only set a variable by getting the return parameter from another command, sometimes a user needs to set a constant (user defined value) in the variable.
- A command can have multiple parameters. Our solution is only a proof of concept, thus only one parameter was implemented.

Chapter 8
Conclusion

IoT is a topic growing widely and there is an increasing amount of devices connected in networks each day. This implies that an increasing amount of users will get in touch with a system that combines all these devices and enables them to communicate. PalCom is such a system, and some of the users will have to do combinations themselves. It is then paramount that they can use such a system without being a professional user.

This thesis solved such a problem for PalCom. By observing similar tools and design approaches and then create a new assembly editor that was based on Block Diagram. The new solution was much more intuitive for new users and gave a better overview of the assemblies for the professional users. By conducting an experiment where the test subjects had to use the new solution and the previous editor it can be concluded that it has been a significant improvement, especially in making the users feel comfortable and understand what they are doing with the editor. There can be done further work in improving the newly developed editor by focusing completely in user interaction and improve those parts to make it even better. Our editor isn't complete for usage in real applications since it is lacking some functions. But the core design and functionality is there and it has been tested on a few users and the experiment indicates that the new editor performs better.

A next step for this editor can be to make it more useful in debugging in real time. When an assembly is running it is not possible to see exactly what and when something is happening through the user interface.

Bibliography

[1] ABB. Power and productivity for a better world. http://www.abb.se/, 2015. Accessed: 2015-08-31.

[2] Boris Magnusson, John Lindholm. Tellstick to palcom bridge users manual. http://palcom.cs.lth.se/Palcom/Users_Manuals_files/tellstick-users-manual-3.0.14.pdf, 2012. Accessed: 2015-08-31.

[3] European Comission. Community research and development service (cordis). http://cordis.europa.eu/project/rcn/74625_en.html, 2008. Accessed: 2015-08-31.

[4] David Svensson Fors. *Assemblies of Pervasive Services*. PhD thesis, Lund University, 2009.

[5] Gaudenz Alder. Jgraphx on github. https://github.com/jgraph/jgraphx, 2008. Accessed: 2015-08-31.

[6] itACih. It-stöd för avancerad cancervård i hemmet. http://itacih.cs.lth.se/itACiH/itACiH.html, 2015. Accessed: 2015-08-31.

[7] Jakob Nielsen. How many test users in a usability study? http://www.nngroup.com/articles/how-many-test-users/, 2012. Accessed: 2015-08-31.

[8] Lennart Andersson, Datavetenskap, LTH. Uml-syntax. http://fileadmin.cs.lth.se/cs/Education/EDAF10/documents/umlsyntax.pdf, 2013. Accessed: 2015-08-31.

[9] Linocell. Bluetooth camera shutter selfie remote. http://www.kjell.com/sortiment/telefoni-kommunikation/mobiltelefon-tillbehor/gadgets/selfie-tillbehor/linocell-bluetooth-fjarrutlosare-for-foto-p96117, 2015. Accessed: 2015-08-31.

[10] Mikael Steiner, Product Group Extended Automation, ABB. The control builder diagram editor. https://library.e.abb.com/public/2806d5e53b87725cc1257b55001f08fd/3BSE074022_en_2013-Apr_NEU_Partner_Day_-_Diagram_Editor_oversikt.pdf, 2013. Accessed: 2015-08-31.

[11] Nexa. Nexa tellstick on/off switch. http://www.nexa.se/vara-produkter/system-nexa/paket/eycr-2, 2015. Accessed: 2015-08-31.

[12] Donald A Norman. *The Design of Everyday Things*. Basic Books, 2002.

[13] Peter Andersen (ed.), Simon Bo Larsen (ed.). *PalCom External Report #70: Developer's Companion*. 2009. IST-002057 PalCom.

[14] Philips. Philips smart tv - 40pfs6909. http://www.philips.se/c-p/40PFS6909_12/6900-series-ultratunn-smart-led-tv-med-full-hd-2-sidig-ambilight-och-smart-tv, 2015. Accessed: 2015-08-31.

[15] Soegaard, Mads. Gestalt principles of form perception. https://www.interaction-design.org/encyclopedia/gestalt_principles_of_form_perception.html, 2015. Accessed: 2015-08-31.

[16] Computerworld Sue Bushell. M-commerce key to ubiquitous internet. http://www.computerworld.com.au/article/84178/m-commerce_key_ubiquitous_internet/, 2000. Accessed: 2015-08-31.

[17] Concordia University. Block diagram. http://web2.concordia.ca/Quality/tools/3blockdiagram.pdf, 2015. Accessed: 2015-08-31.

Appendices

Appendix A

Manual for describing the devices and their services

Philips Smart TV is a network connected TV, which means it can access services on the network or on the Internet. It has some built in apps that one can launch and it also have an open API one can use to send and read data from. The following commands in service *channels* can be used with this TV.

- *Favorite channel 1* - set the TV to channel 1
- *Favorite channel 2* - set the TV to channel 2
- *Favorite channel 3* - set the TV to channel 3
- *Favorite channel 4* - set the TV to channel 4

Note that the TV does not have any on/off command, to accomplish this a tellstick switch, *TellstickTV*, shall be used.

A Philips device is created by the PhilipsControllerService. This service is loaded into TheThing and it will create Philips TV device and display them on the PalCom network for easy interaction.

The TV to be used is called *PhilipsTV@192.168.43.153* and is shown on the network as a PalCom device.

Tellstick switch is a network connected switch that is connected to a power supply. It can then be used to switch on and off the power to a device connected to it. A Tellstick device does have many other commands for different purposes but we are only interested in the following:

- *Turn on* - a command to let power go through the switch.
- *Turn off* - a command to stop the power from getting through the switch.

TellStick devices are created by a TellStickControllerService which is loaded into TheThing. This controller service will display a tellstick device on the PalCom network for easy interaction.

The switch for the lamp is called *Lampa* and the switch for the TV is called *TellstickTV*

Linocell camera shutter is a bluetooth connected device that has two buttons which gives different outputs when pressed.

- *button1Pushed* - is triggered when button number 1 is pushed.
- *button2Pushed* - is triggered when button number 2 is pushed.

This device is loaded as a service on a TheThing called *BluetoothTheThing@Vatan*

Appendix B
Manual for describing BrowserGUI

The BrowserGUI consists of three different parts. The *devices*, The *assemblies* and the *assembly editor*. Only two of these parts will be described since the third part, the assembly editor, is what is being evaluated and the users should figure out how to use.

Devices are things, physical or software simulated objects, which can do things in the physical world or in the computer. A device can then be divided in *services* that it supports and each service has commands. One can think of these as functions of the services in the device. One can call these commands and they will give something back or when something internally happens in the devices they will notify. Let's take a camera device as an example:

The device can have a service called Picture with these commands:

- *TakePicture* - when this command is called, the camera will open its lens and grab the picture and return it.
- *PictureTaken* - This can be a service that is triggered when something internally in the device happens. Somebody pushed a button on the camera to take a picture. Then whoever was waiting for this action will get notified with the service that a picture was taken.

The users can drag and drop the devices from the list (Figure B.1) onto the assembly editor to build their own assemblies and make the connections.

Assemblies are located in the left corner (Figure B.1). They define how the devices and services are connected with each other and what will happen when commands are triggered. You can create assemblies and when you have made your connections of the devices you can run this assembly and when it is running it will perform the connections that were defined.

Assembly editor is the tool that is used to modify the assemblies to perform the specific tasks. This is the tool that is being evaluated. Therefore, it will not be shown nor described here.

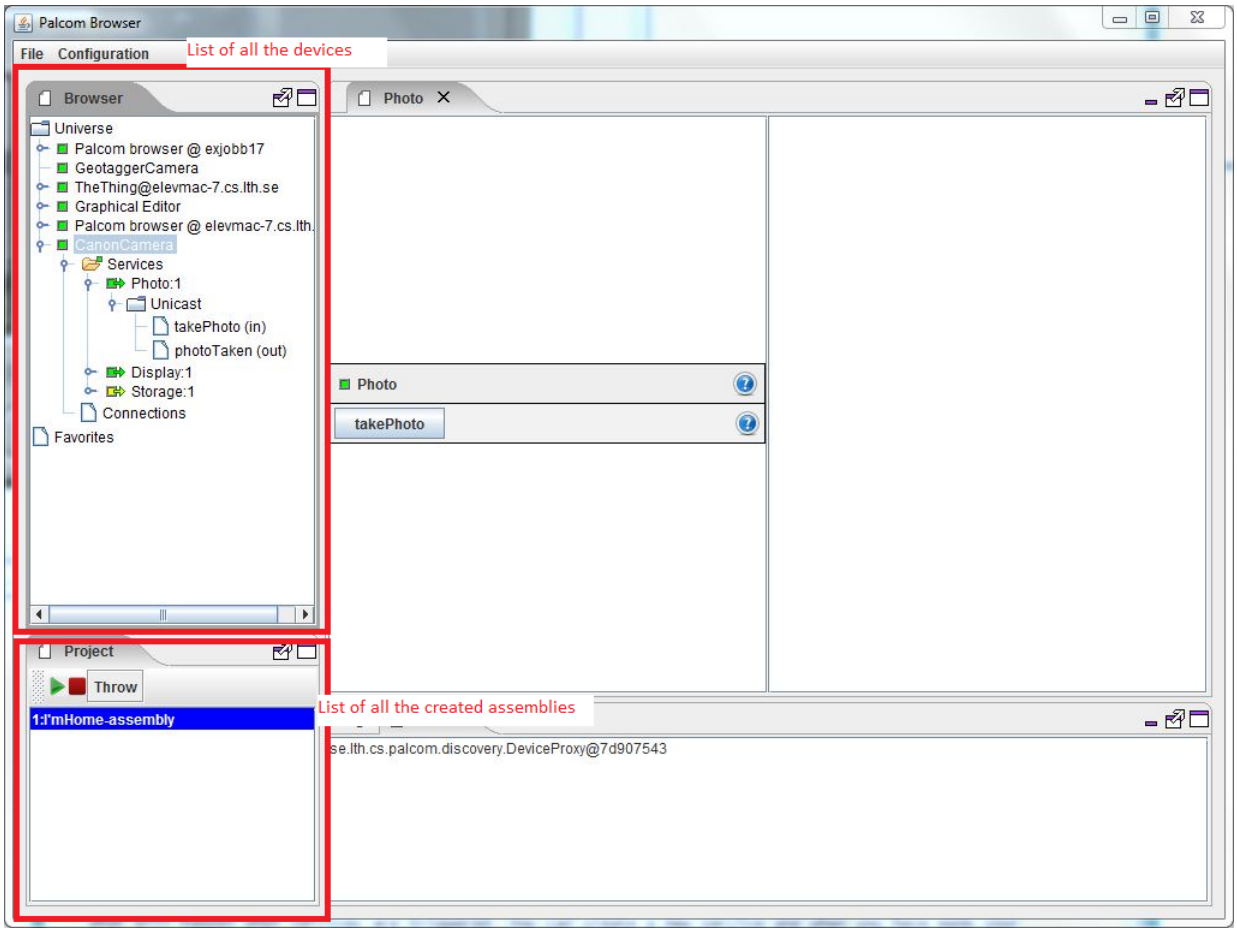

Figure B.1: The BrowserGUI showing where the devices and assemblies are listed

EXAMENSARBETE Overview Browser for PalCom Assemblies
STUDENT Vatan Bytyqi, Jonas Jinbäck
HANDLEDARE Boris Magnusson (LTH)
EXAMINATOR Görel Hedin

Lösning för bättre överblick och snabbare kom-igång-tid för ett Internet of Things-system

POPULÄRVETENSKAPLIG SAMMANFATTNING **Vatan Bytyqi, Jonas Jinbäck**

Ett krav för att den vanliga konsumenten ska använda sig av ett Internet of Things-system är användarvänligheten. PalCom efterlevde inte detta krav, men numera kan användarna genom vårt grafiska gränssnitt enkelt få en överblick över sina kopplingar.

Internet of Things, IoT, är ett hett ämne inom teknikbranschen. IoT innebär att allt fler enheter ansluter sig till Internet och andra nätverk för att kunna kommunicera med varandra. Denna kommunikation i ett smart system kan utnyttjas i bland annat sjukhus, industri och privata hem. LTH har arbetat i över 10 år med en lösning, PalCom, som möjliggör sådan kommunikation oberoende av tillverkare och protokoll (IP, bluetooth med mera). PalCom har börjat appliceras i verklig miljö, på bland annat sjukhus för att möjliggöra för patienter att hemifrån göra regelbundna kontroller som vikt, blodtryck, blodsocker med mera. Dessa mätningar kan sedan läkare och sjuksköterskor följa direkt från sjukhuset. Om mätningarna visar tecken på sjukdomar eller försämringar av patientens hälsotillstånd kan de enkelt kalla in patienten för en grundligare kontroll på sjukhuset.

PalCom-användare som har satt upp assemblies för att få dessa enheter (våg, blodtrycksmätare, insulinmätare med mera) att kommunicera med en centralenhet (sjukhuset) har problem med att få en överblick över vilka enheter som är kopplade och hur kommunikationen mellan dem sker. Tidigare har användare behövt rita upp, på papper, hur de har kopplat de olika enheterna och servicerna för att få en bättre överblick och förståelse för vad som sker. Denna metod är varken användarvänlig eller hanterbar när systemet utvidgas med personal som använder det, antal produkter som kopplas, utökandet av enheter på sjukhusen med mera.

Vår lösning är ett nytt grafiskt gränssnitt som gör det enkelt att få en överblick över hur man har gjort sina kopplingar samt hantera sina kopplingar. Gränssnittet gör det betydligt enklare att applicera PalCom till även andra områden, som till exempel hemmen, för att automatisera vardagliga sysslor.

Komplexiteten i det befintliga verktyget är orsaken till att man behöver vara en erfaren PalCom-användare och att man får en introduktion till verktyget för att det ska vara möjligt att skapa och redigera assemblies. Vi har därför tagit fram ett gränssnitt som är väsentligt enklare för den vanliga användaren och testat detta genom att utföra ett scenario för smarta hem där användarna kopplade ihop en TV, en lampa och en bluetooth-knapp. Genom att mäta tiden för att utföra scenariot med det gamla verktyget och det nya kunde vi jämföra vilket verktyg som var bäst. Resultatet av mätningarna visar att vårt nya verktyg är mycket bättre för användare som aldrig tidigare har använt PalCom. Med vår lösning kan nu PalCom distribueras till vanliga hemma-användare som enkelt vill kunna koppla enheter så att de kan kommunicera och på så sätt bygga sitt smarta hem.