Backend for an End User Portal

George Younan

Master of Science Thesis
Stockholm, Sweden 2009

TRITA-ICT-EX-2009:194
Creating a flexible and dynamic portal and investigating how to provide richer interaction with the home

GEORGE YOUNAN

Master’s Thesis at ICT
Supervisor: Roger Lindahl
Examiner: Leif Lindbäck

TRITA-ICT-EX-2009:194
Abstract

This paper examines how the Backend of a web portal can be built, investigating different techniques required to deliver a fully qualified solution. The main goal is to converge services and information from remote systems to a single access point. The communication with remote systems adds a demand for flexible usage of the software. Two design patterns that provide means to easily change the implementation in use is regarded; Dependency Injection (DI) and Service Locator. Desired uses for the Portal was gathered and a number of these were successfully implemented with the use of DI in a prototype solution. Different standards and guidelines for remote access to home devices was evaluated, gathering the strengths and drawbacks of each one. The evaluation of remote access showed that there is no silver bullet, but rather compromises will be necessary when choosing a technique.

Referat

Skapandet av en flexibel och dynamisk portal och utredning om hur man kan ge rikare interaktion med hemmet.

Contents

List of Figures i
List of Tables ii
Acronyms iii

1 Introduction 1
1.1 Outline . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

2 Problem Description and Motivation 3
2.1 Portal Uses . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
2.1.1 Remote GUI . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
2.1.2 Self Service Activation . . . . . . . . . . . . . . . . . . . . . . 3
2.2 Home Gateway (HG) Independence . . . . . . . . . . . . . . . . . . . 4
2.3 Integration with Home Devices . . . . . . . . . . . . . . . . . . . . . 4
2.4 Integration with the Operators’ System . . . . . . . . . . . . . . . . 4
2.4.1 Base Requirements . . . . . . . . . . . . . . . . . . . . . . . . . 4
2.4.2 Advanced Requirements . . . . . . . . . . . . . . . . . . . . . . 5
2.5 Integration With Virtual Configuration Management (VCM) . . . . 5
2.6 Undefined Requirements . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.6.1 Maintaining Flexibility . . . . . . . . . . . . . . . . . . . . . . . 6
2.6.2 Performance and Scalability . . . . . . . . . . . . . . . . . . . . 6

3 Use Cases 7
3.1 Portal Use Cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
3.1.1 User login . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
3.1.2 Show HG information and current setup . . . . . . . . . . . . . 8
3.1.3 Show customized documentation . . . . . . . . . . . . . . . . . 8
3.1.4 Show active services . . . . . . . . . . . . . . . . . . . . . . . . 9
3.1.5 Show available services . . . . . . . . . . . . . . . . . . . . . . 9
3.1.6 Billing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
3.1.7 Show Voice over IP (VoIP) Information . . . . . . . . . . . . 9
3.1.8 Show historic events . . . . . . . . . . . . . . . . . . . . . . . . 10
3.1.9 Reach and Configure Hard Drive . . . . . . . . . . . . . . . . . 10
### 3.1.10 Content distribution

3.1.11 Remote file Download

3.1.12 Remote BitTorrent Download

3.1.13 Media Streaming

3.1.14 Set-Top Box (STB)

3.1.15 Home Surveillance - Web Camera

3.1.16 Network topology

3.1.17 Answering Machine

3.1.18 FAQ

3.1.19 E-mail form

3.1.20 Language change

### 3.1.21 Remote Access Use Cases

- 3.2.1 Schedule Digital Recording Device (DVR) Recording
- 3.2.2 Streaming Media
- 3.2.3 File Access
- 3.2.4 Parental Control

### 4 Available Techniques and Frameworks

4.1 Inversion of Control (IoC)

- 4.1.1 Dependency Injection (DI)
- 4.1.2 Service Locator
- 4.1.3 Conclusion

4.2 Communication flow policy

- 4.2.1 Configure All Instances
- 4.2.2 Single Configuration
- 4.2.3 Summary and Decision

4.3 Retrieving Configuration Status

- 4.3.1 Polling System
- 4.3.2 Event-driven
- 4.3.3 Comparison

### 5 Remote Access Evaluation

5.1 Interaction Devices

- 5.1.1 Home Network Devices
- 5.1.2 Non-networked Devices

5.2 Standards and techniques

- 5.2.1 Universal Plug and Play Remote Access (UPnP RA)
- 5.2.2 IP Multimedia Subsystem (IMS) approach
- 5.2.3 Web based approach

5.3 Competing Solutions

- 5.3.1 Home Server

5.4 Technique Comparison

- 5.4.1 Home Device Compatibility
- 5.4.2 Required HG Capabilities
List of Figures

1.1 Overview of the Portal and involved systems. .................................. 2
4.1 Interaction of an all configuring Portal ......................................... 18
4.2 Only informing interaction .......................................................... 19
4.3 Configuring an HG through VCM Connector .................................. 20
4.4 Event-driven communication ......................................................... 22
5.1 Remote access device setup .......................................................... 24
5.2 Devices involved in UPnP RA ....................................................... 26
5.3 IMS aproach for remote access ..................................................... 29
5.4 Web based approach for remote access ........................................ 31
5.5 Features of CyberLink Live .......................................................... 32
6.1 Containers role in Dependency Injection ........................................ 41
7.1 Sequence diagram of database example ........................................ 48
7.2 Class diagram of database example .............................................. 49
7.3 Base design of Backend modules .................................................. 49
7.4 Multiple protocols with VCM ....................................................... 50
7.5 Class diagram of integration with VCM ......................................... 51
7.6 HGDevice interaction ................................................................. 55
7.7 HarddriveManager interaction ....................................................... 56
8.1 Delay test graphical ................................................................. 58
C.1 Class diagram of the prototype. .................................................. 68
List of Tables

5.1 Comparison of techniques for remote access .......................... 37
8.1 Delay test statistics ....................................................... 59
Acronyms

**ACL** Access Control List

**ACS** Auto-Configuration Server

**API** Application Programming Interface

**ASP** Application Service Provider

**CPE** Customer-Premises Equipment

**DI** Dependency Injection

**DLNA** Digital Living Network Alliance

**DVR** Digital Recording Device

**HG** Home Gateway

**HGI** Home Gateway Initiative

**IMS** IP Multimedia Subsystem

**IoC** Inversion of Control

**NAT** Network Address Translation

**RPC** Remote Procedure Call

**SSL** Secure Sockets Layer

**STB** Set-Top Box

**UPnP RA** Universal Plug and Play Remote Access

**UPnP** Universal Plug and Play

**VoIP** Voice over IP

**VPN** Virtual Private Network

**WSDL** Web Services Description Language

**XML** Extensible Markup Language
Chapter 1

Introduction

Being able to control thousands of Home Gateways (HGs) through one system is very powerful, allowing for easy management of devices. This is what an Auto-Configuration Server (ACS) delivers and Tilgin offers one such called Virtual Configuration Management (VCM). While this gives many benefits for the operators, it does not provide easy configurations to the end user. The end user usually uses a local WebUI interface for configurations, accessed on the HG itself. This limits configuration accessibility from outside the home.

The Portal is a system intended to be planted at the operator’s premises and will allow end users to log in and see their available services and configurations of their HG. The collection of user data should be retrieved from the operators system; to avoid data incoherence, double storing and for security reasons. The data will be used to authenticate users and to retrieve information on the services that are currently in use and which can be activated from the operator. Configuration information will be fetched from the operators VCM. An overview of the system and how the different elements are thought to interact is shown in figure 1.1. Practically, both the Portal and VCM will be set up at the operator. Only the Portal will be accessible for the end users, though, since VCM is intended for management purposes. The 3rd party systems are thought of as systems planted at the operator but could be placed elsewhere. Note that these three systems will work independently, even though the Portal will utilize the other two. The gateway is a Customer-Premises Equipment (CPE) and is set up in the end user’s home.

As described, the aim for the Portal is to collect data and services from both the operator and the HG to one accessible point. As a central point of access, adding interaction with home devices add for the possibilities of powerful, user-friendly features. Possible features include access to stored media and scheduling a recording on a digital video recorder, accessible from a remote device outside the home. Therefore another possibility is to gather data and services available in the home, exposing them to the Internet and allowing remote access. Naturally, an important issue here as well is authentication, remote access should only be allowed for authenticated and authorized users.
Integrating the Portal with the home opens up many possibilities and greatly increases its value from an end user perspective. The HG has a notable benefit for these purposes; end user familiarity. End users have had gateways or equivalent equipment in their home for a long time. Before broadband came into the scene, the 56K modem was a common product in homes all over the world (and still is in various countries). Gateways are simply necessary equipment in order to get Internet access. Because of this, end users are familiar with having gateways in the home and accept it to be a necessary part in receiving Internet connectivity.

Since the Portal is not designed for one specific operator but intended to be applicable for all customers of Tilgin, there are needs for it to be flexible and modular when dealing with operator premised systems.

### 1.1 Outline

Chapter 2 will give a problem description and motivation to why the Portal should be introduced. Chapter 3 will show different use cases that could be interesting for the Portal. In chapter 4, available techniques and frameworks interesting for the Portal will be covered. Directions for remote access along with their respective strengths and drawbacks are discussed in chapter 5. Chapter 6 describes use cases and techniques chosen for the prototype implementation. Chapter 7 details the design of the Backend and the API provided. Experiments and results are discussed in chapter 8, followed by conclusions and last thoughts in chapter 9.
Chapter 2

Problem Description and Motivation

2.1 Portal Uses

There are many possibilities and different uses for a Portal. Through the Portal, the operator can provide a user-friendly service and provide an opportunity to supply additional services to their customers. Supporting the Portal will be simplified, because the support line at the operator can log in to the users Portal and share a uniform view to find a solution. More direct uses follow.

2.1.1 Remote GUI

An HG often has many different settings and configurations that can be made. Some of the commonly used settings would be beneficial to change remotely, for instance enabling Voice over IP (VoIP). The primary goal with the Portal is to allow end-users to reach and configure their HG through the Internet, rather than having to be connected locally to the HG. It is not however desired that all changes available through the local WebUI are mimicked to the portal, but rather the changes that a user would want to access from outside the home.

One must also consider that some settings that are available locally, can if erroneous break the connectivity of the HG to the Internet. Since the Portal is not directly connected to the HG, this means that the HG will no longer be reachable from the Portal. Deciding which configurations that should be available through the Portal needs to be made with caution.

2.1.2 Self Service Activation

There is also the possibility of upgrading a users subscription with the operator, such as increasing bandwidth or adding services like VoIP. This provides users with an ability to add richer functionality themselves.


2.2 Home Gateway (HG) Independence

Another gain with the Portal solution is the independence of the HG. Different gateways may be provided by operators to their end users and can still share the same look and feel in the Portal. This is due to the use of standardized protocols for the management communication (between ACS and HG). Tilgin currently have both their VCM and HGs fully compliant with such a protocol, namely TR-069 [15].

To the operators, a dynamic and easily modified end user portal also gives them the opportunity to brand the interface and customize the looks to their style. It will also give one uniform interface for all their customers, no matter what brand or type of HG they have. This stands in contrast to the local WebUI, provided by the HG, which is brand specific (e.g. Tilgin branded).

2.3 Integration with Home Devices

The difficulties with remote access are requirements for interoperability and accessibility through the Internet. If the home is to be integrated with the Portal, it should be possible to communicate with devices regardless of manufacturer. Standardized techniques should be utilized, to avoid making a specialized solution for each device.

Then there is the issue of accessing the devices through the Internet. The devices need to somehow be exposed to the Internet but only to remote users that are authorized through a secure authentication technique.

Can a system for remote access be provided, that is secure, user friendly and highly interoperable with devices?

2.4 Integration With the Operators’ System

The general idea for the Portal is to act as a data aggregator; gathering services and information for each individual user to one accessible point. User data is fundamental information in order to provide a per user experience. Since the Portal is installed at the operator, this information should be retrieved from operator native system. This is relatively basic information, since operators should already have some kind of stored user data. Other features, however, might require more information out of this user data storage.

2.4.1 Base Requirements

To authenticate a user, the Portal needs to interact with some sort of user database belonging to the operator. This could for example be a MySQL server or a directory service such as LDAP. There are some requirements on the information available at that instance, first being authentication (native for databases). Secondly, a unique identifier of the HG belonging to the end user will be needed. This identifier could
CHAPTER 2. PROBLEM DESCRIPTION AND MOTIVATION

for example be a MAC address or a serial number. The described information forms the base requirements.

The basic information is something that the customer should already have in some part of their system, since they should be able to map a certain user to an HG to be able to provide support etc for the end user. The question that remains is if the customer stores this in one and the same database, and otherwise how?

2.4.2 Advanced Requirements

In order to provide a customized view, showing only configurations and services applicable to the specific end user, the end user will need to be included in some kind of group or role. These will signify what services are active, can be deactivated and what upgrades can be made. This forms the advanced requirements.

The advanced information however is a little trickier. As with the base requirements, it is reasonable to believe that the customer already has this information stored in their system, because they need to keep track of the end user’s service configurations since this could be related to billing, or other service. Once again, whether the information is stored in the same database is unknown. Even if it is, the information can be stored in more complex and different ways than with the base requirements. This makes it a bit more demanding since the integration needs to work with as many different setups as possible.

2.5 Integration With Virtual Configuration Management (VCM)

Remotely configuring an HG is today only possible through VCM, explicitly for operators. There is also a web service called VCM Connector provided with VCM. This allows the operators to write programs and automate configuration settings on one or multiple HGs, by using remote procedure calls (RPCs) through SOAP [10] messages. VCM Connector contains methods to be able to perform all possible settings to devices in VCM.

The integration with VCM requires some form of library in the Backend, containing methods that form a configuration action that is provided in the Frontend. Building up a base of this library is necessary to see that the system is working.

The integration to VCM Connector makes it easy to configure the HG, but one thing to consider is the limitations of configurations. Certain changes are not possible to make through VCM, and indirectly through VCM Connector, because only the customer should be able to set those parameters. For example, the operator should not be able to set an end users firewall port configurations etc, so this is not available in VCM. This limits the possibilities available with the portal concerning remote access. This may or may not be a major issue, however the impacts on decisions taken because of privacy demands can limit functionality of the Portal.
CHAPTER 2. PROBLEM DESCRIPTION AND MOTIVATION

2.6 Undefined Requirements

While the previous uses and problems are rather easy to measure and identify whether or not they are supported, other requirements of the portal are more vague in the confirmation.

2.6.1 Maintaining Flexibility

To be able to deliver a modular and flexible service, the structure of the system should allow for adjustment. These properties could be leveraged if it is possible to provide a black-box API to the Frontend. A module structure could then allow the Frontend to use one type of module without knowing the techniques it is using. This means that a module can be switched to one that suits the customer’s system and setup. How can a flexible solution be delivered?

2.6.2 Performance and Scalability

Performance and scalability is another factor that comes into play when centralizing the service point. Every user that accesses the Portal will put additional load on the service provider. It is thus important that the system is responsive and can scale efficiently with an increasing number of simultaneous users. Will solutions for the flexibility requirements, mentioned above, reduce the performance of the system?
Chapter 3

Use Cases

Use cases are good tools to help describe a system, giving a clearer picture of what we can and would want to do with a system. With that in mind, some use cases may not be realizable today but are listed because the functionality may be desired. As such, the following list of portal use cases contain all use cases that are interesting, and not only the ones chosen for the prototype implementation. These are followed by the use cases for remote access, presenting different ways to engage in a remote access session.

3.1 Portal Use Cases

When listing the requirements of each use case, the frontend of the Portal will not be considered since that is outside the scope of this thesis. That part will be handled in the thesis work by Adam Skoglund [29].

The purpose of this list is to gather information on different types of use cases in order to find out different requirements on the Portal. Some of these use cases could be divided into many (similar) use cases but are for this reason listed as one.

3.1.1 User login

- Enter credentials
- Authenticate against operators user database

Requirements:

- Operator provides users with credentials and stores this in some form of user database (storage types can vary)
- Customization of database type and structures in the Portal
- To set permissions in the GUI (for tabs etc.), VCM and operator based systems need to be checked here to match user properties to permission groups/roles
CHAPTER 3. USE CASES

With user customization as a top priority for the Portal, we only want to show relevant services and options to the user. While one can hope that the operator would provide such information, it is highly unlikely since there is no obvious reason why they should. Instead, the Portal should itself gather information and place a user in to groups/roles. This can be based on operator based systems (ex active services) or setup in VCM (ex 'VoIP enabled').

3.1.2 Show HG information and current setup

- Get HG information
- Show corresponding HG picture
- Show current configurations on HG
- Change the configuration on HG

**Requirements:**

- Communications with VCM Connector and matching multiple properties to single actions

An important thing to note here is that the HG information and matching picture are just as important as the current configuration. These will help make the Portal feel more personalized to each end user, which is one of the main goals with the Portal.

VCM Connector will be used to get the current configuration on the HG. In order to only show information relevant to each specific user, properties in VCM Connector should be combined to form actions. The actions found will be shown in the Portal. As an example, this could mean linking properties which notify that a Session Initiation Protocol (SIP) account is enabled, or the opposite to disable/enable the SIP account.

A user should also be able to change the configuration of their HG, or namely each action listed. The Portal should then communicate with VCM Connector in order to make the changes.

3.1.3 Show customized documentation

- Get HG information from VCM
- Show documentation matching the HG

**Requirements:**

- A record of documentation files and which HG they are intended for
CHAPTER 3. USE CASES

3.1.4 Show active services

- Get active services from operator
- Modify active services

Requirements:
- Operator keeps some record of activated services (such as billing)
- Custom modules in the Backend for different kinds of systems

The user should be able to see and (possibly) be able to modify their active services. When a modification is issued by the user, the Portal should send an indication of the request to the operator, such as writing back to the user database. The operator should be able to choose whether or not modifications should be allowed.

3.1.5 Show available services

- Get services provided from operator
- Show services available for specific user, based on HG etc.

Requirements:
- Operator keeps information on services available
- Customizable demands in the Portal for why a service can be provided for a specific user

3.1.6 Billing

- Show unpaid bills
- Show information of current and historic bills

Requirements:
- Operator keeps billing information for each user
- The Portal can retrieve this information from different types of systems

3.1.7 Show Voice over IP (VoIP) Information

- Show VoIP phone numbers and base information
- Get historic call list from operator

Requirements:
CHAPTER 3. USE CASES

- Operator keeps billing information on VoIP services for each user
- The Portal can retrieve this information from different types of systems

Being a central point of information, the Portal could also gather the history of calls made/received by the user. This history would be stored at a billing database at the operator.

3.1.8 Show historic events

- Show historic service upgrades
- Show historic service removals
- Show historic HG configuration changes
- For all these items, show a status sign (e.g. 'Pending', 'Done' etc.)

Requirements:

- Actions made by user needs to be stored in the Portal
- Status on an action should be updated (in a reasonable time) when it is operational.

A user should be able to track their requests to see if progress has been made.

3.1.9 Reach and Configure Hard Drive

- Browse content of device
- Open/view and remove files

With the USB port on an HG, an end user can connect an external hard drive to the HG. The motivation to do this rather than connect it to your PC should be to view and reach its content from any location (or at least, locally in the network). Without that, there is no real difference between connecting it directly to a PC.

The Portal is a natural point of access for this kind of interaction. Not only since it can be reached from any location but also for being able to give more user-friendly GUI than one might want to put on the HG. The first step in bringing in the hard drive to the Portal is accessibility.

3.1.10 Content distribution

- Upload file through the Portal to the hard drive
- Synchronization service; synchronize phone, camera etc. with hard drive
CHAPTER 3. USE CASES

Being able to view current data on the hard drive is essential, but the Portal should provide more communicative uses. Being able to also put data on the hard drive gives more options to users. This paves the way to synchronize and backup data with a mobile device (or a hard drive on the end users PC).

3.1.11 Remote file Download
- Enter URL & the HG will download the file to hard drive

3.1.12 Remote BitTorrent Download
- User uploads torrent-file (or link to torrent-file) to the HG
- BitTorrent download through HG to hard drive
- **Requirements:**
  - BitTorrent client on HG

This use case gives more convenient and powerful file handling options for the end user. The downside is that it puts more work on the HG, requiring it to do all the processing as well as requiring features on the HG which may not be available.

3.1.13 Media Streaming
- Choose media to stream
- Start/stop media streaming
- **Requirements:**
  - Streaming service on HG

As with the previous use case, BitTorrent download, this use case lays more burden on the HG. This requires the HG to be able to stream media from a connected hard drive.

3.1.14 Set-Top Box (STB)
- Remotely controlling the recording schedule of programs and movies
- Parental control, limit available channels (perhaps schedule-based)

Reaching and controlling a STB through the Portal integrates yet another CPE to the Portal. The end user could remotely schedule a recording or activate parental control.
3.1.15 **Home Surveillance - Web Camera**

- View live-feed
- Take a picture and view it
- Schedule picture-taking

Connecting a web camera to the USB port of the HG brings an opportunity to provide home surveillance with the Portal. The user could be able to view a live-feed from his home, choose to take a picture right now and view it or make a schedule for when pictures should be taken.

3.1.16 **Network topology**

- Show all devices connected to the HG
- Show the address and type of device

After mentioning possibilities for third-party interactions, with these connected it can become messy for users to get a clear view on their network. Providing a network topology containing all the devices connected to the users HG can help in finding possible errors in their setup and further increases the feeling of a personalized Portal for each individual end user.

3.1.17 **Answering Machine**

- Show if new messages are available
- Listen to messages

A user should be able to view and listen to messages recorded in the answering machine of the HG.

3.1.18 **FAQ**

- Show general FAQ
  - Stored locally or at the operator?
- Show customized FAQ, specifically for active services and HG type

3.1.19 **E-mail form**

- Show an e-mail form to operator support
- User can send an email to operator support
3.1.20 Language change

- In admin interface, change language
- New files matching the chosen language are used for textual output in the GUI

Requirements:

- No textual information in the Portal should be mixed with code, this should be stored in separate text-files

3.2 Remote Access Use Cases

The following use cases are intended both to expose possible uses for remote access and to clarify requirements on the HG and remote device if the use cases are to be implemented. What differentiates these use cases to Portal use cases is the fact that these allow access to a device in the home network (i.e. a device connected to the HG). An important thing to keep in mind is that the list of use cases could be endless, there are simply too many uses for remote access. This were narrowed down to use cases that are somehow unique in the interaction and requirements on devices.

For all these cases, the devices that can be used for remote access are PC:s or mobile phones belonging to either the owner of the content or a relative, friend etc. The reason to distinguish between the owner of the remote device is that they impose different levels of dynamicity. That is because if there is a required (and perhaps very complex) setup phase for some solution, it might be endurable on an end user owned device since it only occurs once. But it would be very cumbersome if a new (non user owned) device is used. Thus, for example, using a laptop belonging to the owner requires less dynamicity than using a laptop belonging to an arbitrary associate of his.

3.2.1 Schedule Digital Recording Device (DVR) Recording

- User uses a personal laptop to schedule a DVR recording
- Media is recorded and stored on the home device
- User can come home late and can watch the media

This use case relies on the use of a laptop to manage recordings of tv shows and movies. It is a basic example of an end user wanting to access home devices, while being out of the home. The motivation is that end users that have home devices providing rich features should be able to utilize these, even in this case of unexpected circumstances.
CHAPTER 3. USE CASES

3.2.2 Streaming Media

- User keeps media stored on a home device
- User goes to a friend’s house and decides to view a video clip from the home device
- The friend’s PC is used to access the home device and stream the media

By using an arbitrary device for the remote interaction to the home, convenience and user friendliness become two important issues. Since end users will want to use the device immediately and are assumed to be non-technical, any setup phase is desired to be simple and quick. A complex setup phase might hinder the practical usefulness of the use case, giving the remote access service a weaker selling point towards operators.

3.2.3 File Access

- User is out in the city and wants to show pictures stored in a home device
- Using a mobile phone, user accesses the home device with the photos
- The mobile phone can also be used to take new pictures and store them on the home device

One aim for this use case is to enable remote access to home storage devices by using a mobile phone for interactions. By allowing remote access to home media, a user on the move can always have access to their entire media library. Instead of keeping everything on the mobile phone, the media may now be stored and kept persistent on a home device.

Compared to a PC, the mobile phone is a small device but not computationally powerful. But unlike a PC, most people have their mobile phone within reach at all times. Features such as remote file access augment this inherent mobility of the device. It is however, important to remember that any client software required on the mobile phone needs to take the technical limitations in regard.

3.2.4 Parental Control

- User uses a mobile phone to access the HG
- Activates parental control
  - STB is configured to block certain channels according to time schedule
  - HG is configured to block Wide Area Network (WAN) connectivity to home PCs etc. according to time schedule
CHAPTER 3. USE CASES

As with the previous use case (3.2.3), this use case takes advantage of the mobility features inherent to the mobile phone. Parental control could be used by the end user to, for example, ensure that their kids are not staying up all night surfing the web. Likewise, this can be used the other way around. For instance, the end user can deactivate parental control if the kids request Internet connectivity for some good reason.

The communication between the HG and the STB could be Universal Plug and Play (UPnP)/Digital Living Network Alliance (DLNA) or a generic protocol such as HTTP. Alternatively, a standardized management protocol such as TR-069 could be used. In the latter case, the Portal could be incorporated to provide the feature.

Supplying end users with the possibility to both access and control their home fulfills the remote access purpose. That is, to give end users complete control over the home, wherever they are.
Chapter 4

Available Techniques and Frameworks

With the integration of VCM and the operators’ systems, and relying quite heavy on those, there are some possible techniques and protocols to use. This chapter will describe techniques and mechanisms that can be used to accomplish this task.

4.1 Inversion of Control (IoC)

A software component often needs to access some service to perform their task, which can be implemented by another component. For instance, the Frontend requires services which can be accessed through the Backend. For each wanted service, there would be a standard module in the Backend for the interaction. In case the interaction process to access the service changes, a lot of modifications to possibly both Frontend and Backend components might be required.

This is what the abstract principle IoC intends to simplify. Instead of a class directly instantiating the implementation used for a wanted service, the implementation should be received from another instance, an assembler. The implementation can then be thought of as a module or a plug-in, which can be switched for a different plug-in without affecting the rest of the system. This results in an inversion of control, with the caller not knowing which implementation was received.

There are different techniques to implement IoC, but generally the decision lies between using Dependency Injection (DI) and Service Locator. That is because both are popular choices and have matured implementations available. Descriptions follow below.

4.1.1 Dependency Injection (DI)

Dependency Injection is a mechanism to supply external dependency to a software component. Using this technique, a component in need of a service can be written with disregard to the implementing module that is handling the actual service interaction. The component will use an implementation of an interface, which will appear to work in the same way no matter what implementation module is used. Being in
the context of an object-oriented programming language, an interface specifies a list of methods with empty bodies. Classes that choose to implement an interface signs a contract, which obliges them to provide the methods listed in the interface. This interface sets up demands for implementing modules in order to guarantee that one uniform interaction procedure is used for all the modules. The module to use will be provided by a different software component.

There are three types of DI:

1. Constructor Injection
2. Setter Injection
3. Interface Injection

These styles are simply different directions that lead to the same destination. Constructor injection (as the name suggests) requires components to declare a constructor that will include interfaces all of wanted services. Setter injection instead requires components to declare setter functions for interfaces of all wanted services. With the third option, interface injection, an interface needs to be defined for all services that needs to be injected to some object. Such an interface contains a method to store the injected object, and must be implemented by all classes that wish to use that type of service (more information available at [19]).

Even with the use of DI, which implementation module of the interface that should be used needs to be specified somewhere. There are generally two ways to configure this, although they can differ slightly; using code files or configuration files. Depending on which implementation is used in the development environment, there might be limitations for both the injection styles mentioned above and configuration styles.

### 4.1.2 Service Locator

A service locator [40] is an object that contains information on available services. A software component that requires some service can ask the service locator and receive the implementing module [19]. Though using either service locator or DI can give the desired effect to the Portal, the interaction is a bit different between the two techniques.

### 4.1.3 Conclusion

Where with DI a component expects to receive the implementing module from some other component, with SL the component will instead ask for it. Service locator is intended for Java EE [26] and uses JNDI [25] to store the mapping between interface and implementing class. Comparing the configuration to DI, where an XML-file is typically used, it is clear that DI is more lightweight and easier to manage than service locator.
CHAPTER 4. AVAILABLE TECHNIQUES AND FRAMEWORKS

There are also other features provided by service locator, such as caching. Nonetheless, DI provides the wanted IoC feature in a simpler manner and is thus the chosen technique.

4.2 Communication flow policy

Moving up from the local connection means that there are more ways to interact with the system. Certain features that allows for user modifications might involve updates for all three elements: the HG, VCM and operators system. Considering such a change that affects both the operators system and VCM, it is not guaranteed that both systems will succeed in their operations. There are different directions for the Portal to take with these kinds of actions.

4.2.1 Configure All Instances

One could leave it up to the Portal to modify all affected parties; changing something at the operator and VCM. The tricky part here is atomicity. An atomic operation [33] signifies a set of operations that will all either succeed or fail. Figure 4.1 shows such a scenario where a configuration at the operator failed.

The modifications at the operator and VCM essentially build up a transaction that should be atomic: both should either succeed or fail. The Portal will need to be able to recognize and handle the case where either instance can not go through with the transaction. Certain control mechanisms would be required to handle this; rollback functionality, keeping a copy of the new configuration and enabling it once it is OK for all instances or a method of testing each instance. While a solution for similar problems is two phase commit, it is not that easy to apply to this case.
One reason is that the Portal might receive answers about the transaction 24 hours after issuing it, as some customers VCM systems only have contact with the HG every 24 hours. Which means that a possible backtracking to abort the transaction (if any instance could not accept) could happen much later. The Portal would also need to remember transactions for a long time, and there could even be two interfering transactions going on at the same time. These might even be causally related, then there is the problem of choosing one of them. Another reason is that the possible time difference between replies could lead to confusion for the end user. A requested service may be seen as activated on the HG but is not working since the operator has not or could not activate the service.

### 4.2.2 Single Configuration

Another direction would be for the Portal to only interact with the operators system and leave it up to them to configure VCM. The operator would configure VCM, handling possible errors, and then use some mechanism to inform the Portal of the status of the operation. As seen in figure 4.2, the Portal would not need to configure the operators system but only inform that the user wants some modification done. This greatly simplifies the interaction required towards the operator.

### 4.2.3 Summary and Decision

With the Portal having information aggregation as a main goal, the single configuration option might be better suited. Configuring all instances gives the Portal a different objective, to maintain proper execution. That is not the motivation for the Portal and somewhat contradicts the mentioned goal.
Although no operator systems were involved in the prototype implementation, single configuration would be a suitable choice if that was the case. The implementation process would be simplified, and if desired, the configure-all-instances direction could be taken up later for a real live implementation.

### 4.3 Retrieving Configuration Status

When a configuration is sent through VCM Connector, the configuration will be set in VCM (see figure 4.3). A reply will be sent back from the service, indicating if the configuration was successfully set in VCM. The reply does not, however, indicate whether the targeted HG actually received the configuration. That is because VCM, and consequently VCM Connector, does not insure when the HG will receive this new configuration. The described issue corresponds to an uncertain amount of delay before step 2 in figure 4.3 can be accomplished. There are two reasons why a delay would be caused; a reboot of the HG and issues with Network Address Translation devices Network Address Translation (NAT).

A relatively short delay (compared to the next example) could be added if the configuration requires a reboot of the HG, the delay being a few minutes. The Portal can, however, ask VCM Connector to contact the HG immediately, but this solution does not fix the issue because of the second reason for possible delay.

If an HG is located behind a NAT, even larger delays could be caused. A NAT can be used to hide private IP addresses behind a single, public IP address. Under these circumstances, communication between VCM and the HG would be unattainable unless initiated by the HG. So if VCM Connector is asked to contact the HG, VCM will try to contact the HG. But since the NAT will be accessed, not the HG, the operation will not succeed. Practically in these cases, a schedule is
set specifying when the HG itself will initiate the contact to VCM, thus making it possible for the HG to receive a new configuration. The delay induced because of this issue depends on the interval at which the HG contacts VCM for new configurations; it may take hours until the next contact and therefore also configuration.

Regardless of the relevance of the above issues, the fact remains that after issuing a configuration request to VCM Connector, a reply does not necessarily mean that the configuration is active on the HG. For the Portal to provide the history use case 3.1.8, it is necessary to know if a configuration is pending or actually active on the HG. This information needs to be retrieved through a different mechanism.

4.3.1 Polling System

One way of retrieving the status of a configuration is to directly check VCM and see if the configuration has been set. This interaction style corresponds to polling. The Portal would need to actively check for updates in VCM, for each user and respective HG. Although this might seem as a simple direction, a few requirements will be laid on the Portal. For every configuration that has been sent, the dates need to be gathered from VCM and temporarily stored in the Portal. That will be needed in order to recognize a configuration update on the HG.

4.3.2 Event-driven

Another way to handle the configuration status is to utilize asynchronous communication by using event-driven interaction (also known as event-based). Figure 4.4 shows the steps involved when using this mechanism. After a configuration has been sent to VCM (black line), no polling is going on in the Portal. Instead, the portal has a listener which is always active and waits for incoming events from VCM. At some point in time, the HG and VCM will communicate and synchronize the configuration of the gateway (red lines). When the HG contacts VCM and the new configuration is active, VCM will send an event to the Portal with relevant information (green lines).

This way, VCM can publish information as soon as the HG has been contacted and has the correct configuration, leading to the Portal receiving the information once available. This prevents delays between information availability and information retrieving. This kind of event-driven interaction also opens the possibility to spread other information from VCM, such as publishing support information on errors.

Axis2

Axis2 [3] uses a new architecture which is more flexible, efficient and configurable [3] than that of Axis 1.x. The new system adds support for easy addition of modules that can extend the functionality of the Web service engine. One such is called WS-Addressing [32] and adds asynchronous Web service invocations to Axis2. By enabling asynchronous communication, the reply for a request has no time limit.
and can be sent after some action has occurred. These features make it possible to provide an event-driven interaction using only modules provided through Axis2. Use of WS-Addressing can be combined with WS-ReliableMessaging (WS-RM) \[31\] to provide reliable messaging mechanisms. Through use of WS-RM, a SOAP message is given additional assurance of delivery in the presence of software component, system, or network failure.

The use of the above mentioned techniques requires the use of Axis2, but VCM Connector uses Axis1.x. Thus, if these features in Axis2 are to be utilized, VCM Connector will need to be migrated to use the newer version of Axis. The amount of work this would require depends on the implementation of VCM Connector, investigation of which is outside the scope of this paper.

**RSS**

One direction for event-driven systems, that requires less programmatic integration, is using RSS \[6\] feeds. RSS is a standardized format used to publish updated works, such as news headlines and blog entries. A RSS feed is specified using Extensible Markup Language (XML), which makes publishing information rather straightforward. The only additional resource needed is client software for reading the feed, often called a RSS reader. Having gained widespread use these RSS readers come in different flavours, easing the way for users that want to retrieve published information. With both publication and retrieval of information made clear, the technique is feasible for use as an event-driven system.
CHAPTER 4. AVAILABLE TECHNIQUES AND FRAMEWORKS

Note

While both proposed event systems allow for asynchronous communication, there is still one necessary part left: discovery of the event. This is common for both of the proposed event systems, because neither technique solves the issue. When the wanted event occurs, some instance of the system needs to discover this and take action (initiate the messaging). With RSS, you would need to know if any client is interested in the information and in that case publish a RSS feed. With the use of Axis2, there would also be a need to remember exactly who those clients would be, because a message will be sent explicitly to each one of them.

4.3.3 Comparison

The main difference between the two alternatives is the responsible party for information retrieval. With polling, the Portal (or any other client) is responsible for checking and finding out if an HG has been updated. With events, this responsibility is shifted to VCM Connector and only leaves the Portal responsible to receive the information.

When using polling, one must choose how often VCM is polled. If this is done very frequently, the information will be available in the Portal quicker, but will consume more resources of the server running the Portal. This introduces a trade off between information availability and server load. In the event based system this issue is non-existing; as soon as VCM publishes the information, an event will be sent to the Portal. This however, requires that VCM Connector itself can discover when a certain event should be triggered, then somehow publish the relevant information to the Portal. This feature is not currently available in VCM Connector, so this approach would induce new implementation requirements.

Weighing the choices against each other, the paradigm that yields the richest interaction is the event-driven system, for the following reasons:

- Less resource consumption (at least by clients)
- Higher information availability
- Technique is applicable and useful for different kinds of services
Chapter 5

Remote Access Evaluation

Remote access to devices connected to the HG is a quite powerful feature and a big step towards integrating the HG with the home. To enable remote access, many factors and requirements come into play, such as security, simplicity and device mobility. The purpose of remote access is to let home devices be accessible from outside the home. However, they should only be exposed to authenticated and authorized users. With all devices connected to the HG, it is a natural point of access and permission checks.

Regardless of which approach is used for remote access, the HG needs to be able to communicate with devices in the home. This can be UPnP or a different mechanism. The remote device will need to support the technique provided by the home device. For instance, if the home device provides a video streaming service, the

Figure 5.1. Remote access setup, with devices and an HG in the home. Remote devices in a different location communicating through the HG.
remote device will need to be able to display the streaming media. Support between
the devices involved in the interaction is of course common for all techniques. It is
nevertheless worth mentioning in order to gather all requirements.

A home device can be connected to the HG through the network or through USB
(currently, in the future perhaps through other Bluetooth or FireWire). There is a
notable difference when accessing an IP device and a hard drive connected through
USB. An IP device should be reachable through the WAN-side of the HG, but the
hard drive should be integrated in more ways. Since there is no native network
communication for the hard drive, the HG would need to provide this.

5.1 Interaction Devices

There are two general types of devices; home network devices and non-networked
devices. The characteristics of each type and what sets them apart is described
here.

5.1.1 Home Network Devices

Devices in the home that are connected to the HG are thought to be generic IP
devices, which means they are connected to the HG either through the Local Area
Network (LAN) or the WLAN. This could be devices such as a DVR, web camera
and network storage. These devices all provide some kind of service that can be
reached through a network; it can be assumed that they are intended for use at
least through the LAN. Therefore they should also have a method for interacting
with them in order to access the provided service.

5.1.2 Non-networked Devices

When building up use cases for the Portal, new ideas and scenarios came up. One
such idea was to leverage end users familiarity with HGs and provide easy-to-use so-
lutions for popular commercial products such as Network-Attached Storage (NAS).
A NAS device is connected to a computer network and provides file-based data
storage to clients in the network.

The difference from a typical commercial NAS is simplified hardware require-
ments. All additional hardware required is a regular hard drive connected through
the USB port. The HG will handle communication with the storage device through
USB. This is a significant difference from a home device (detailed in section 5.1.1) .
With the communication interface being USB, the storage device can ordinarily be
used from a single computer, as with an internal storage unit. Therefore it does not
provide any means of interaction through the network (if connected to the network
directly, it would be considered to be a home device).
CHAPTER 5. REMOTE ACCESS EVALUATION

Figure 5.2. Example setup of UPnP Remote Access, showing the role of each involved device.

5.2 Standards and techniques

There are clear benefits of working with standards, such as avoiding disorganized solutions or reinventing the wheel. The Home Gateway Initiative (HGI) [20] and the UPnP Forum [16] are two initiators for standards, both providing recommendations for remote access.

5.2.1 Universal Plug and Play Remote Access (UPnP RA)

UPnP [17] is a technique that allows home devices to seamlessly connect and communicate with each other. One of the goals with UPnP is to simplify the implementation of networks in the home. While originally intended for LANs [18], the technique might still be applicable for remote access intentions. The UPnP RA architecture is considered for this purpose and the UPnP Forum are currently in the process of standardizing it. The reason why regular UPnP can not be used for remote access is that device discovery in UPnP makes use of multicast messages. These will be difficult to forward beyond the home network, because typically Internet routers discard such messages [18].

The UPnP RA architecture specifies mechanisms which enables a remote device outside the home LAN to be logically included in the home network. From then on, devices can communicate with each other using regular UPnP procedures [17].

Interaction Details

The interactions of UPnP RA can be divided in two parts; setting up a secure communication link and synchronizing UPnP device information. Once the secure link is set up, the remote device can synchronize information about available home devices. At this point, the remote device can start interacting with the home devices.
CHAPTER 5. REMOTE ACCESS EVALUATION

To be able to utilize the features of UPnP RA, the remote device will include a component of the architecture called Remote Access Client (RAC or RAClient). Another required part of the architecture is the Remote Access Server (RAS or RAServer). The RAS will expose the UPnP devices available in the home network, as well as its own UPnP services, to the RAC. The RAS could be a PC, a 3rd party device or, more suitable for this paper, a residential gateway. It is noteworthy that although employed in the home network, the RAServer needs to be reachable from the Internet. What should also be noted is that PC software implementations of a RAServer can be expected to appear if or when UPnP RA becomes popular, as happened with UPnP [41]. That could very well become a competitor to the HG direction (for more details on competitor solutions, see 5.3.1).

Through the RAServer, a RAClient can gain access to home devices. These two components will establish secure communication channels and synchronize UPnP device information with each other. This is provided by the UPnP RA architecture components Remote Access Transport Agent (RATA) and Remote Access Discovery Agent (RADA), respectively. Both the remote access server and client incorporate these two components.

The RATAs are responsible for establishing secure channels between the RAS and the RAC. This is done through the use of Virtual Private Networks Virtual Private Network (VPN) [42]. VPN can be used to establish a secure connection between two nodes across an insecure network such as the Internet. The remote device in the VPN may also access the private home network, by receiving a private IP address belonging to the home network.

**Issues and Limitations**

One issue with the UPnP Remote Access architecture is a part of the setup phase, which must occur before a device can be used remotely. To be precise, it is the use of a VPN for the secure channels that adds this requirement. First of all, both VPN end-points need to have special VPN software. Server- and client software for VPN is needed for the RAServer and RAClient, respectively. Secondly, the end user will need to configure the security certificates of both the remote access server and client. This is required if they are to establish a Remote Access Transport channel between them (i.e. the VPN tunnel). Depending on the VPN technique, this setup phase might be required to happen when both the RAS and the RAC are in the same LAN, in those cases limiting the remote device applicability to mobile devices.

The requirement for client software limits the dynamicity of the remote access service, since a user can not use the service without the software. That might not have been a big problem if security certificates did not need to be configured as well. Together, these two demands make it difficult to access the home network from an arbitrary remote device.

One possible solution for the VPN software issue could be to use SSL VPN [39]. SSL VPN is a kind of VPN that employs Secure Sockets Layer (SSL), which makes the VPN accessible through web browsers via https. Although traditional VPN (i.e.
IPsec VPN) requires specific VPN client software to be installed on a client device, SSL VPN has no such requirement. Virtually any Internet connected browser can be used to access the VPN. The catch is that to enable full access to the home network (i.e. to provide a private IP address), a thin client is needed on the remote device. The thin client does not have to be preinstalled, though, but can rather be installed upon first contact with the VPN server (the HG). The thin client usually comes in the form of a Java applet or ActiveX Control [14]. At least in the case of a Java applet, interoperability and flexibility towards remote devices is not entirely hindered since PCs/laptops and most mobile phones support Java. That enables end users to gain remote access, for example, through a device belonging to an acquaintance. However, Internet cafés or kiosks usually blocks the download of these kinds of applications because they are a common medium for the spreading of malware. As such, even the thin client might not be applicable for this setting.

What is yet to be realized though, is compliance with UPnP RA. Although SSL VPN is not mentioned in the standard VPN solutions for Remote Access Transport, UPnP RA vendors may provide support for the technique (for remote access server or client devices).

### 5.2.2 IP Multimedia Subsystem (IMS) approach

One way to handle remote access is through an IMS [37] approach. IMS is an architectural framework for delivering IP multimedia services. Although originally intended for cellular networks, the framework has been expanded to allow any type of broadband connection. In an effort to simplify the integration with the Internet, IMS makes use of Internet Engineering Task Force (IETF) protocols where applicable, such as SIP [37].

This is one of the promoted approaches by HGI [22]. The basis of the approach is to use the IMS network to authenticate users, enable secure media communications and for routing. The remote user is authorized by either end-user or service provider controlled Access Control Lists (ACLs). The home devices targeted by this approach are generic IP, UPnP or SIP devices.

**Interaction Details**

Figure 5.3 shows the proposed architecture for the IMS approach. The remote device initiates the communication by sending an IMS invite message to the HG. Before arriving at the HG, the message goes through the IMS Control plane which then routes the message to the HG. The IMS Control plane inserts a P-asserted-ID to the message, which will be inspected by the HG to authenticate the user. Now that the user is authenticated, the IMS Control plane can set up a secure session on the IMS media plane between the remote device and the HG. At this point, synchronization of home device information is needed. In the HGI proposal, a device discovery component in the HG will collect information on available home devices. The HG will only list devices that the end-user (with an associated P-
CHAPTER 5. REMOTE ACCESS EVALUATION

Figure 5.3. HGI proposal to an IMS solution for remote access, showing home devices that may be accessed and the involvement of IMS.

asserted-ID) is authorized to access. This is achieved by checking remote access ACLs. Similarly to the HG, a device discovery component in the remote device will collect information on available remote devices. The device discovery components synchronize using a component for synchronization with the remote device and the home network, respectively. The end-user chooses a home device for the interaction and a media session is set up between the HG and the remote device. The HG will forward traffic between the remote device and the home device.

IMS gives additional benefits, such as being able to use Quality of Service (QoS) for each active session. This could be used, for instance, to prioritize a real-time video-streaming session over a background photo upload session.

Issues and Limitations

With the authentication handled by the IMS network (i.e. the service provider), an end-user can use a private mobile device to reach home devices. This is in fact the only way to access the home devices. The IMS network will not authenticate a device belonging to someone else (e.g. friend or relative), thus the remote access would be denied. In their proposal, HGI suggests that the remote device could be a Application Service Provider (ASP) system such as an IMS Application Server (AS). This could be used to enable remote access for end-users, from non-IMS terminals such as a PC in an Internet café or a friends home. Although this might be a viable solution, it does not fix the issue but rather shifts it to the AS. The AS would need to be able to authenticate users, show information on available home devices and present access to home device content. These responsibilities are similar to those of the Portal, with the extension of IMS interaction to the gateway.

The approach requires the remote device to have remote access client software, handling device discovery and synchronization. The HG would need to have the
server variants of that software, as well as supporting remote access transport and configurable ACLs for remote access.

There is a key issue with the IMS approach, concerning device discovery. The HG and the remote device need to synchronize device information. How this should be done, however, is not specified by HGI but a suggestion for using UPnP RADA is given [22]. The use of UPnP RA components add the requirements of UPnP RA, namely the need for VPNs. This limits which terminals can be used (further detailed in 5.2.1).

5.2.3 Web based approach

The web based approach is promoted by HGI for non-IMS networks and makes use of the local WebUI already available on the HG. The WebUI is a web server on the HG, originally intended for local end user configuration and graphical user interface. The interface would be used to authenticate users and allow remote access to home devices. A new requirement for the WebUI is support for secure connections using SSL/HTTPS for remote access. Because most service providers do not provide static IP addresses, HGI also promotes the use of a Dynamic Domain Name System (DynDNS) service on the HG. The DynDNS service makes it possible to find and access the HG using a unique Domain Name System (DNS) address.

A big gain with the web based approach is interoperability with end-user terminals. Virtually every device with Internet connectivity has a web browser and that is all that would be required to enable remote access. The interaction technique between home devices and the HG can be UPnP or other mechanism.

Interaction Details

The proposed architecture for the web based approach is described in figure 5.4. Using an HG unique name (i.e. a DNS name), the remote device initiates the session by accessing the HG with a web browser. The end user logs in by entering user name and password credentials. The secure SSL/HTTPS connection is set up, which enables the end user to see available home devices. The HG will only present information about devices that the end user is authorized to access.

The user may now access a selected device, after which a media session is established which does not need to be encrypted. The remote device gains access to the selected device after the HG opens up a pinhole through the firewall. Access to the selected device should only be allowed by the HG to the authorized remote device (i.e. the IP address should be inspected).

Issues and Limitations

Although the web based approach has a user-friendly use by the remote devices, it does have some unresolved difficulties. For one, the interaction between the remote device and the home device is poorly described. If UPnP is to be used between the HG and a home device, opening up a pinhole through the firewall is not enough
to address the problem of interacting from a remote device. As described in the UPnP RA architecture [18], regular UPnP is designed for use in a LAN and would not work across a WAN without extensions. A UPnP device exposes its services through multicast messages which can not be received by the remote device [18]. For that reason, the HG would need to be responsible for at least device discovery of UPnP devices and to forward traffic between the devices (after a device is chosen). For other mechanisms than UPnP, however, this might be a sufficient solution.

5.3 Competing Solutions

Before making new development plans, it is always good to look at competitors or other equivalent solutions. Considering the use cases in 3.2, can any of the remote access solutions outperform the alternative solutions? The purpose of this section is to give base information on this topic and lift distinguishing goals of available solution.

5.3.1 Home Server

Another way to get remote access to the home is the use of a home server [36]. A home server is a server that resides in the home network and provides services to local or remote devices. The server does not need to have a substantial amount of computing power [36]. On the contrary, since the number of served clients are typically low, old PC systems can be recycled and used as a home server.

There are many possible uses for a home server, including web serving and BitTorrent downloads. In the context of this paper, it is the media serving capabilities that are interesting. A home server can be used to serve photos, music and video
content to devices in the home or on the Internet. The act of exposing this service to the Internet is called placeshifting [38]. There are different ways to accomplish this, and there are several PC programs available which allow placeshifting of media [38]. Software such as CyberLink Live [11] allows media serving to the Internet through the simple use of a browser (see figure 5.5). CyberLink Live also support access of DLNA devices in the home [11], all through the same interface accessible by a browser. Placeshifting can also be achieved through the use of devices intended for the sole purpose of enabling remote access [38].

The home server is indeed a competing alternative for enabling remote access to the home and can be accomplished without direct involvement of an HG. Regardless, the HG still has a strong benefit compared to the home server solution. While the home server has access to more computational power, the HG direction is a more user-friendly option. Firstly, no additional hardware is necessary. Even though a home with an HG will most likely contain at least one PC, the home server option requires it to be powered on constantly to allow remote access at any time (although software such as CyberLink Live allows for the computer to be in sleep and awakened only when needed [11]). Secondly, the software available for placeshifting still require some setup, which can be tedious for a non-technical end user.

Another drawback is that the hardware or software providing the placeshifting may impose technical limits for the remote session. An example of one such limit with CyberLink Live is that photos and videos have a maximum allowed size [11].

**5.4 Technique Comparison**

The use cases in 3.2 present possible remote access interactions and the various demands that these impose on a remote access solution. Let us see how the techniques can handle the inherent demands of these use cases.
CHAPTER 5. REMOTE ACCESS EVALUATION

5.4.1 Home Device Compatibility

With the possibility for different techniques to achieve remote access, limitations may occur in supported home devices and communication standards. Compatibility with UPnP, DLNA and generic IP devices in the home network is regarded.

UPnP RA

UPnP RA is being standardized with the intention of enabling remote access to UPnP certified devices [18]. Consequently, the technique supports access to UPnP devices in the home. Since the technique utilizes VPN tunnels to gain access to home devices, other techniques may be supported as well (e.g. generic IP or DLNA devices). In the case of DLNA certified home devices, the device discovery technique used in UPnP RA should be able to retrieve information on these devices as well. That is because DLNA uses UPnP techniques for device discovery and control [12]. This means that information about DLNA devices in the home should also be possible to retrieve from the Remote Access Server (RAS). Because the RAS synchronizes device information with the Remote Access Client (RAC), the remote device would then be able to interact with these kinds of home devices. The remote device would need to be DLNA certified, though.

IMS approach

The IMS proposal does not require a specific direction for the device synchronization between the HG and the remote device. UPnP RA could be used for this purpose and would allow for the compatibility provided by the technique. Support for remote access to SIP devices in the home, however, distinct the IMS proposal from UPnP RA. Access to SIP devices is provided by existing functionality in the IMS enabled HG [22].

Web based approach

As for the Web based solution, it is up to the HG to provide support for different techniques. How this is to be done is not specified by the guideline. The HG will need to gather information on available devices in the home and show them in the WebUI when an end user logs in. Since the media session is set up between the remote device and the HG, the HG will need to forward all traffic between the remote device and the home device. These responsibilities of the HG essentially correspond to being a client of the home device technique, such as a UPnP or DLNA client, with the addition of forwarding all traffic to the remote device.

5.4.2 Required HG Capabilities

The role and responsibility of the HG in a remote access interaction depends on the technique used. In order to decide whether the technique is feasible, one important step is to establish the requirements imposed on the HG.
CHAPTER 5. REMOTE ACCESS EVALUATION

UPnP RA

The UPnP RA Architecture proposes that the Remote Access Server (RAS), the element providing access to home devices from an external network, could be an HG. Alternatively, this could be a PC or a stand-alone, dedicated device. The requirements of the RAS are not complete, although the main functionality would be to incorporate a RADA for device synchronization and a RATA for secure data transmissions. The RATA requires VPN functionality on the HG, which will probably be the heaviest requirement on the HG.

IMS approach

Both the IMS and the Web based approach are proposed by HGI. Consequently the HG plays a role in both cases.

The main requirement of the IMS approach is that the HG needs to be IMS enabled [21]. Additionally, to enable a remote access interaction, the HG will need software for device synchronization, a manageable ACL for remote access and remote access transport. The required software and interaction towards remote devices is not standardized, HGI only notes that it is needed.

Web based approach

The web based approach makes use of the existing local WebUI of the HG, adding the requirement of SSL/https for use in remote access. Since the WAN IP could be dynamic, an additional requirement of a DynDNS service on the HG is added. Similarly to the IMS solution (see above section), the required software and interaction towards remote devices is not standardized.

5.4.3 Applicability For Mobile Phones

Certain techniques require client software on the remote device. In order to use mobile phones for the remote access interactions, the client software has some additional requirements to fulfill.

UPnP RA

With UPnP RA, the remote device will need to be a certified Remote Access Client (RAC) device. With the technique currently in the standardization phase, it is unclear exactly what requirements this will lay on a device. For the same reason, it is uncertain whether or not the technique will gain widespread use and be supported by many devices. Since regular UPnP can be used on mobile devices, however, it can be assumed that the development of UPnP RA considers the interoperability with mobile phones. With representatives from manufacturers such as Nokia and Samsung in the working group pushing the technique forward, there is reason to believe that mobile phones are target devices and that the technique has the potential to gain popularity.
IMS approach

Both the UPnP RA and the IMS solution require software running on the remote device. With the IMS direction, the remote device will need to be IMS enabled. What this actually means in terms of device requirements is unclear and unspecified [8, 9]. Although IMS was designed with mobile networks in mind, as of today the number of IMS enabled mobile phones is unheard of [9]. Thus, it should be noted that following this approach, a realization would more likely occur later on rather than the near future, compared to the possibilities with the other approaches. Especially since the operators need to have an IMS network up and running for the approach to work, when or if that will happen is up to the operator.

Additionally to being IMS enabled, the device will need software acting as a client for the remote access interaction with the HG. The required software and interaction towards the gateway is not standardized either (although promoted by HGI).

Web based approach

By only requiring a web browser on the client side, the Web based solution truly shines when it comes to remote device interoperability. Yet, being able to access the web page does not entirely complete the requirements. As detailed in section 5, the remote device will need to support the technique provided by the home device. What makes it important to note for this approach is that the requirement will only be apparent once the specific service is going to be launched, where the other approaches force the remote device to meet the requirements initially.

5.4.4 Remote Device Flexibility

As previously mentioned, some techniques require software on the remote device in order to achieve the remote access. This can be a limiting factor when using an arbitrary device for the remote access interaction. Compared to section 5.4.3, this section goes more towards use cases such as 3.2.2, where the remote access is initiated for the first time on a specific device. The goal is thus to establish any required setup phase for each technique.

UPnP RA

The use of VPN tunnels in UPnP RA leads to remote device interoperability issues. VPN is a viable technique to use on many Personal Digital Assistants (PDAs) and smartphones, such as those using the operating system Windows Mobile [24]. Thus, (at least some) cell phones are possible to use for the remote interaction. However, the setup phase before a VPN connection can be established reduces the flexibility. The setup phase makes it inconvenient and technically difficult to enable remote access for the first time on a new remote device. If the setup phase is to be avoided when using remote access, the end user will need to use their own equipment for the
remote access interaction. In other words, the end user will need to use a private laptop or cell phone. Otherwise, if a device belonging to an acquaintance is used, the setup phase is required. Hence the practical use for the remote access interaction is more or less limiting, depending on how the end user intends to use the service.

Once again, the impact of the limit depends on how flexible the solution needs to be. With the use of SSL VPN, there is a possibility to enable both clientless and thin client access. The thin client approach enables full network access and the support for different systems of the remote device; Windows, Mac, Linux, Windows Mobile, etc. Exactly which systems are supported depends on the SSL VPN system used and the different flavours in which it can deliver the thin client. What is achieved is a VPN solution that allows for zero-setup at the remote device, accessibility through a web browser and interoperability towards remote devices. The catch is no support for Internet kiosks or cafés, as these will typically block the downloading and running of these applications.

IMS approach

This reflects on the IMS approach if UPnP RADA is used for device synchronization. If some other mechanism is used instead, the IMS approach will not be hindered by the issues related to VPN, since there would be no need for VPN. Regardless of the use of VPN, there is an issue with authentication that affects remote device flexibility.

When initiating remote access, the identity of the remote device will be inserted by the IMS network and used by the HG for authentication. Since the device is interacting with the HG for the first time, it will not get authenticated by the HG since the gateway is unfamiliar with the device. The HG decides which device is authenticated and authorized to access devices through the use of ACLs. Thus, the desired identity of the remote device needs to be added to the remote access ACL on the HG before remote access interaction can be achieved on the new device. Modifying the ACL might be accessible from the local WebUI or remotely by exposing the WebUI to the WAN. The former hinders remote device flexibility in that the HG will need to be set up while in the home, in other words spontaneous use of use case 3.2.2 is not possible. The latter is not a complete solution however, because as detailed in section 5.2.3, this solution would also imply the need for a DynDNS service on the HG.

Web based approach

Again, this is the strong point of the Web based solution. Requiring no setup at all on the remote device but simply an installed web browser (as most browsers support SSL/https), the flexibility towards remote devices is very high.
CHAPTER 5. REMOTE ACCESS EVALUATION

<table>
<thead>
<tr>
<th></th>
<th>Device type</th>
<th>UPnP RA</th>
<th>IMS</th>
<th>Web based</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UPnP, DLNA,</strong></td>
<td>Home device</td>
<td>Yes</td>
<td>Yes, but unstandardized HG interaction</td>
<td>Yes, but unstandardized HG interaction</td>
</tr>
<tr>
<td><strong>Generic IP</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PC/laptop</strong></td>
<td>UE</td>
<td>UPnP RA client</td>
<td>Through an IMS client or through an IMS AS</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Non-UE</td>
<td>The above, and VPN through either thin-client or pre-usage installation</td>
<td>The above, and an HG configuration is required prior usage, unless IMS AS is available</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Mobile</strong></td>
<td>UE</td>
<td>UPnP RAC, virtually any (if SSL) or smartphones (other VPN)</td>
<td>IMS enabled</td>
<td>Virtually any</td>
</tr>
<tr>
<td><strong>phone</strong></td>
<td>Non-UE</td>
<td>The above, and setup is required unless SSL</td>
<td>The above, and setup is required prior usage</td>
<td>Virtually any</td>
</tr>
<tr>
<td><strong>Internet café</strong></td>
<td>Non-UE</td>
<td>If UPnP RA client is preinstalled</td>
<td>Generally no, yes through an IMS AS</td>
<td>Yes, unless media transfer is blocked</td>
</tr>
<tr>
<td><strong>Encryption</strong></td>
<td></td>
<td>IPsec, SSL or other</td>
<td>Ensured by IMS</td>
<td>SSL/https for access, not required for media transfer</td>
</tr>
<tr>
<td><strong>Needed for release</strong></td>
<td></td>
<td>UPnP RA specification release, development of a UPnP RAC</td>
<td>Operators with IMS networks, handset availability, HG software development time</td>
<td>HG software development time</td>
</tr>
</tbody>
</table>

Table 5.1. Listing remote access techniques and their respective strengths/drawbacks on practical requirements.

### 5.4.5 Summary

Table 5.1 summarizes the strengths and shortcomings of each technique. Devices mentioned can be the users home devices, or devices used as clients for the remote access. The remote devices are either user owned equipment (UE) or non-user owned equipment (Non-UE) (e.g., acquaintance owned equipment). More detailed information on these findings are available in section 5.2.
Chapter 6

Prototype Use Cases and Adopted Techniques

This chapter will detail the features incorporated in the prototype and the directions taken for required techniques.

6.1 Prototype Use Cases

Although many other use cases are interesting, only the use cases listed here are considered for the implementation of the prototype. As stated when listing use cases in section 3, there are many possibilities with the portal and not enough time in the context of this paper. Furthermore, some use cases require certain features from the HG which are not available as of today.

The use cases have been chosen with a big-problems-first approach, weighed against the benefits gained from the use cases. This means that the most important use case will be one that has big issues to tackle, yet yields great benefits if handled. A short motivation follows each use case below.

6.1.1 Basic Login

The Portal is a per-user customized solution, so each user will need to log in in order to see their individual Portal information. Since the other use cases are not accessible without authentication, this is a rather important step. Even so, the log in use case is rather trivial in this sense since a log in module is not unvisited territory for a company like Tilgin.

There is one thing though, that differs the log in use case 3.1.1 from a traditional log in modules and that is customizability. The traditional authentication process goes something like this:

1. user enters credentials
2. server checks credentials with local database
if bad credentials, user is not allowed access

3. server checks user rights with local database

4. server sends page to user

As described in 3.1.1, in step 3 the Portal should check with VCM and the operators system instead of a local database. Depending on the results, the user should be disallowed/allowed the view on certain web pages. The customization of the web page does not, however, put any requirements on the Backend; other than providing means to retrieve the information (i.e. access the operators’ user database). As with many other use cases, this will require module changeability. The use cases below have other benefits and will need to handle the flexibility and mobility requirements as well.

Thus said, for the prototype implementation we will use a local database directly in the Portal, containing all users. This will ease the way for providing a user customized page.

6.1.2 HG configuration

Use case 3.1.2 was chosen because it provides benefits in multiple areas; VCM communication is used, information is shown from the HG (customized per user) and remote configuration is made possible. That said, with the implementation of this use case, some issues with other use cases can be cleared and the implementation of these will be eased.

6.1.3 Hard Drive Integration

Use cases 3.1.9 and 3.1.10 provides access and control of a hard drive connected to the USB port of the HG. The prototype will have the possibility to provide remote accessibility through a user-friendly WebUI. The integration with the hard drive will be the entry point for third-party hardware integration to the Portal.

6.2 Techniques Used

Different techniques used in the prototype implementation.

6.2.1 VCM Connector

VCM Connector is a web service on VCM, a tool that can be used to script common configuration changes. A client interacts with the web service through "known" classes, providing remotely accessed methods through SOAP.

This will be the major target to make configuration changes in the Portal, since the HG (and its connected devices) is reached through VCM. While this module
CHAPTER 6. PROTOTYPE USE CASES AND ADOPTED TECHNIQUES

does not necessarily need to be configured by the IoC Container, there are still advantages to be gained with that choice.

Managing the VCM Connector module in the Portal with the IoC Container allows for extraction of certain parameters and settings from the code to the configuration file. Keeping information such as host names or IP addresses makes it easier for operators to setup the Portal for their environment, increasing the user-friendliness of the Portal. Setting parameter names through the configuration file increases the adaptability of the Portal; parameter names can easily be swapped out in case the service is modified.

This should be the direction to take if multiple management protocols should be supported by the Portal (i.e. Recon and TR-069).

SOAP

SOAP [10] is a protocol for exchanging structured information, using XML (Extensible Markup Language). The protocol can be used by client applications as a means of access to VCM services, making Remote Procedure Calls (RPCs) through the web service VCM Connector.

Web Services Description Language (WSDL)

In order to interact with VCM Connector, the Backend will need to know which services can be accessed. This is accessed through the use of a WSDL [43] file. The file will contain detailed information on the related web service, in this case VCM Connector. The information is needed to generate Java classes for the VCM Connector API.

Apache Axis

To be able to use VCM Connector, the classes and API need to be known to the Backend. Apache Axis [1] is an implementation of SOAP, used in the Portal to create Java classes from WSDL files. WSDL2Java [2] is a tool in Axis that allows us to do this.

Practically, the class WSDL2Java provides methods to create Java classes from the given WSDL file. Once the classes have been generated, it will not be necessary to do so again until the web service changes.

The Apache Axis library will still be necessary after the previous step, though. That is because some of the classes provided by VCM Connector extend classes in the Axis library. Consequently, the classes generated by WSDL2Java will also need the library, since they too will extend their respective classes. When a request is sent to VCM Connector, these created classes are used and the Axis library is thereby required.

What is achieved through the use of this library is access to VCM Connector through SOAP, but not having to bother with actual SOAP messages. Making a
RPC is as simple as calling a function on an object of the class that is providing the remote service.

### 6.2.2 Dependency Injection - Implementations

The direction chosen for IoC for the prototype was Dependency Injection. That was mainly because DI requires less programming of extra classes and does not require JNDI while yielding the same result.

However, DI itself is only a concept describing a higher level view of the mechanism. It is up to developers to provide with implementations. A usual term for these implementations are providers or containers [34], which can be standalone products or included in a framework. Figure 6.1 shows how a container will interact with the system. The launcher needs a service from another class, which it knows will extend a certain interface. As can be seen, no dependency lies between the launcher and an implementation. The container will create and inject the implementation to the launcher (more about this interaction in section 7.1.1).

With DI being an acknowledged concept and Java a broadly used programming language, there are many different containers to choose from. A few implementations have been chosen for investigation, with uniqueness and widespread use as qualifying attributes.
CHAPTER 6. PROTOTYPE USE CASES AND ADOPTED TECHNIQUES

PicoContainer

PicoContainer [27] is an IoC container, pioneered towards Constructor Injection. The configuration to set up the injections are written in Java classes. This gives a great benefit when refactoring parts of or a complete project; all configuration code is refactored as well. There is, however, also support for use of meta-data and scripting languages to set up the configuration, giving developers the choice to hold the configuration information in separate configuration files. This feature is provided through PicoContainer Script [28], an add-on to PicoContainer.

Spring IoC Container

The Spring Framework [30] comes with a module providing IoC. The service is provided through the use of the Spring IoC container. The container is responsible for creating objects, configuring these objects and assembling the dependencies between them.

Even though it is a part of the Spring Framework, the container can be used without involving any other parts of the framework. There is also quite extensive documentation provided by Spring, including multiple examples of configurations and different cases [13].

JBoss Microcontainer

JBoss Microcontainer features DI as well as numerous other features [7]. Since we are only interested in DI, the other features do not matter.

The Spring IoC Container and the JBoss Microcontainer are very similar. They both use a similar looking XML-configuration file, both versions providing some additional parameters that can be used.

Conclusion

Even though PicoContainer has support for both code and configuration files, the meta-data support is a secondary plugin. There is not much documentation available on areas such as how to configure the configuration files and what parameters are available.

Both JBoss Microcontainer and Spring IoC Container are more mature in this area, since the configuration files have been the direction since initiation of both projects. An advantage with Spring is that the framework is widely used and well documented, giving it the upper hand against JBoss Microcontainer.

For these reasons, the choice for container implementation in the prototype was the Spring IoC Container.

6.2.3 Configuring DI

It is important to note that the use of DI not only allows for configuration of objects, but other variables in the class as well. Any String, Integer or even collection can
be injected into the object. Taking use of this functionality, variables containing
configuration information can be extracted to the configuration file. The IP address
to the VCM server serves as example of class variable configurations, as the address
is necessary for modules interacting with VCM Connector. Keeping the value in
the configuration file and injecting it into the object makes it easy to change the IP
address if the server or address changes.

**Using Spring IoC Container**

Dependency Injection is a sharp tool, but it does require some configurations from
the user. Each class that is to be injected with an object will need to be configured.
In Spring, these objects are called *beans* [13] and are typically specified in a XML-
file. As detailed in section 4.1.1, there are different methods to accomplish DI.
Although the Spring IoC container supports multiple variants, the advocated use is
setter-injection [13]. This requires a public setter method for every variable that is
container managed in the bean class. This is not that unorthodox, as can be seen
in [ref] this is just plain old Java.

A powerful feature of the Spring IoC container is the possibility to automatically
inject dependencies. When a class is trying to instantiate an object, the Spring
IoC container will intercept the call. The container will check if the object to
be instantiated is configured in the XML file. If so, the container will create an
instance of the object. After the container has filled the object with data from the
configuration file, the instance is returned to the caller class. This results in the
calling class receiving an object that is already filled with data and dependencies.

However, use of Spring Web MVC [13] (Model-View-Controller) is required in
order to use this feature. The Spring Web MVC is a module of the Spring framework,
designed to be flexible, highly configurable and interoperable with multiple view
technologies. Even though this might be a good framework to work with, it requires
that the Frontend of the Portal is using a MVC system. The Frontend is outside the
scope of this paper, but choosing to work with this framework might prove difficult
since the direction taken with the Frontend needs to comply with this standard.

This does not limit the use of DI for the Portal, but it will not be as developer-
friendly. Instead of the usage above, one class will have to do a request to the Spring
IoC container, asking for the wanted module. The module can be configured in the
same way as with Spring MVC though. The only difference is that the injection
can not be implicit, but needs to be explicitly asked for (see A in the appendix for
more details).

A useful property that can be used in the configuration file is the *alias* [13]
property. Using the alias property, a pseudo name can be passed when accessing a
bean. Any class that tries to retrieve a bean by passing the alias name will receive
the bean pointed to by the alias property. The caller class can then use the alias
when requesting the injection, rather than using the direct name of a bean. Now,
if the implementation bean is changed, the caller class remains the same and only
the specified bean in the alias property is changed.
Note that all beans can be stored in the configuration file, even those not currently configured to be used [13]. By use of aliases and keeping all beans in the configuration file, reconfigurations are simplified. A reconfiguration would only require a change in a text field, specifying which bean is to be injected (see B in the appendix for an example configuration).

So what we need to have for each Backend module is:

1. an interface that specifies the module implementations

2. a class that explicitly asks for an implementation
   a) with setter methods for managed variables

3. corresponding bean in the configuration file to allow injection of the module

**Configuration**

With the limitation on usage of the Spring IoC container (as mentioned in 6.2.3), the class that needs a bean can not automatically get injected with necessary objects upon creation. Instead, a class must ask for the bean to be injected itself.

It seems fitting to let the launcher be the requester for an implementing module for the following reasons:

- the launcher is the first point of access for the Frontend
- other than the launcher, only modules and interfaces are required for the injection(s)
- the Frontend should not have to deal with the injection
  - it is the responsibility of the Backend and demands should not be put on the Frontend unless necessary

Practically, the interaction will go as follows:

1. Launcher requests an alias bean from the IoC container

2. IoC container interprets the alias
   a) finds the bean pointed by alias
   b) instantiates bean
   c) injects properties into bean if any are specified

3. IoC container returns the bean

4. Launcher can interact with bean
6.2.4 Hard drive access through Jackrabbit

A hard drive can be connected to the HG through the USB connection. With specific software on the HG, the contents of the hard drive can be accessed through two techniques (at the time of writing): a Samba client in the LAN or a web server accessed from the LAN/WAN. Since remote access is wanted, the first option quickly becomes dismissed and therefore the web server option is regarded. This technique utilizes the local WebUI existing on the HG, adding a path specifically for hard drive access. By exposing the WebUI to the WAN, this path will also be exposed and consequently hard drive access is available from the WAN. The web interface on the HG that provides access to the hard drive supports both regular HTTP as well as Web Distributed Authoring and Versioning (WebDAV) [23]. WebDAV is a set of extensions to HTTP which allows users to collaboratively edit and manage files on remote web servers. With the HTTP option, only view and download of files is available. Creating directories and uploading/removing files is made possible through WebDAV.

To avoid reinventing the wheel and building a WebDAV compliant client from scratch, the prototype makes use of Apache Jackrabbit [4]. The library is utilized in order to reach a component that greatly simplifies the implementation of a WebDAV client. This component is called the Jackrabbit WebDAV Library component [5] and allows the Backend module to easily create, send and receive WebDAV compliant methods to/from the HG.

6.2.5 Notes and Further Details

Some decisions for the prototype imply issues if other routes are taken for a real implementation.

Change of IoC Container

As detailed in 6.2.2, the Spring IoC container will be used in the prototype implementation of the Backend. This however, does not bind the Backend to that specific container. In case a different container is to be used in the real implementation of the Backend, there are some changes to be made, namely: change of launcher classes and configuration file. The affected instances will need to comply with the new container, although it is possible that the configuration file can remain almost intact since the parameter standard may be very similar.

The same goes if the Spring MVC pattern is used in the Frontend. Since that would lead to automatic injection being possible, it might be wanted that the Backend uses that technique instead. The only change required in the Backend code would be the launchers, which should not get any implementations by their own requests any longer. Instead, each launcher would need a setter method for an object of the desired interface.
Uploading files to the harddrive

This feature currently goes through the Portal. This means that a file intended for upload is first uploaded from the end user device to the Portal (handled by the Frontend). When the transfer is complete, the file is uploaded from the Portal to the HG (handled by the Backend). Obviously, this is not an optimal solution as the transfer would probably be faster if the upload went immediately between the end user device to the Portal. This however, requires a mechanism in which the Portal somehow passes on authentication of the end user device to the HG. Such a mechanism was neither available at the time of writing nor further investigated in the report, however, solutions discussed for remote access in section 5 can help achieve this.
Chapter 7

Design and Implementation

This chapter will go through the general design and structure of the prototype solution and briefly describe how one can interact with the Backend (i.e. by the Frontend).

7.1 Architecture

The architecture of the Backend has two important demands; to provide simple interaction mechanisms to the Frontend and the ability to change which module to use without modifying any code.

The Frontend should be oblivious about what the Backend actually does and should only care about the results of issued requests. This means that the second demand on the architecture of the Backend, switching modules, should not affect the Frontend in any way.

If or when a module is changed, for instance changing the type of user database, requests issued by the Frontend should look exactly the same. Thus no modifications to code in the Frontend will be necessary. To remove the necessity of code changes in the Backend, DI will be used.

7.1.1 Example of Architecture

Figure 7.1 describes a database interaction, with the Frontend requesting user information from a Backend module. In this example, the launcher corresponds to the UserInfoManager. The scenario shows the IoC Container injecting an implementation to the launcher. When receiving a reply from the container, the launcher will retrieve an instance of some class. The launcher has knowledge about an interface, DbInteractor, and knows that the retrieved instance is implementing it. But which implementation was retrieved is unknown. The interface specifies the interaction to access the wanted service, making it possible for UserInfoManager to issue a request and pass the result on to the Frontend.
Figure 7.1. Sequence diagram of a database interaction example. Note that DbInteractor is an interface, not a class.

Figure 7.2 shows the class structure of the above example. There are two modules present with similar behaviours, both implement the interface DbInteractor and are database accessors; a MySQL interactor and an Oracle interactor. They would provide similar services, i.e. a user information retrieval service, but work differently internally.

Which one of the implementations that the container should give to the launcher is described by an XML-configuration file. The launcher will ask the IoC Container for an object that implements the DbInteractor interface. The IoC Container will provide an instance of such an object, by instantiating an object of the type listed in the configuration file. The object will be preconfigured with the listed parameters, if any. This configuration file could either specify the MySQL or the Oracle version.

7.1.2 General Structure

Figure 7.3 shows the base design in common for all services accessible through the Backend. Going through the elements from bottom to top, figure 7.3 first show some implementation classes. The implementations will act as plugins and represent the modules which the operator should be able to choose between for some service. To enable module switching, the classes provide the same, fixed set of interaction methods by implementing a common interface. Although the methods in this set may use different techniques and interaction styles internally, they are accessed in a
uniform way and should correspond to similar behaviours. Programmatically, there
is no real limit on how many modules can be available (only limiting factor would
be file sizes, which is trivial). The launcher is the communication point for the
Frontend, providing methods that can be used to access services.
7.1.3 VCM Integration

The VCM Connector integration in the prototype is split up in two parts: VCMInitiator and VCMConfigurator. The reason stems from the fact that different communication protocols can be used on different HGs, as seen in figure 7.4. This directly affects how the Backend interacts with VCM Connector, everything from the settings structure to parameter names can be completely different (e.g. the difference between RECON and TR-069).

VCMInitiator will setup the communication to VCM Connector, while VCMConfigurator is an interface that declares a set of methods. Implementations of the interface will act as modules for one specific communication protocol; the prototype implementation consisted of a TR-069 implementation.

Figure 7.5 is a class diagram that shows how VCMInitiator and VCMConfigurator are connected. HGDevice is the access point for the Frontend and will provide VCM Connector services through VCMInitiator (and, consequently, through VCMConfigurator implementations). More details on the API of HGDevice is available in section 7.3.2.

An instance of VCMInitiator is retrieved through the IoC container, already filled with implementations of VCMConfigurator specified in the IoC configuration file. This is different from previous uses of IoC in the prototype, where the implementations are directly retrieved through the container. That is because multiple modules may be active at the same time, providing access through different communication protocols. This scenario practically means that different end users may have different communication protocols on their respective HG. Hence, the Backend need to be able to concurrently serve users that have HGs using different management communication protocols.

HGDevice will start by requesting an implementation of VCMInitiator from the Spring IoC Container. The VCMInitiator is.

An instance of VCMInitiator is retrieved through the IoC container, already filled with implementations of VCMConfigurator specified in the IoC configuration file. This is different from our other use of IoC, where the implementations are di-
CHAPTER 7. DESIGN AND IMPLEMENTATION

Figure 7.5. Class diagram of the system providing VCM integration.

rectly retrieved through the container. The reason is that modules providing access through different communication protocols may be active at the same time. This scenario practically means that different end users can have different communication protocols on their respective HG.

7.2 Error Handling

In any application, some effort must be put to recognize erroneous behaviour and handle it correctly. The conditions for the Portal make this more important. Since services that can be accessed through the Backend are mainly provided by external resources, it is most likely external failures and errors that will pop up. Each interaction delivered by the Backend needs to be aware of possible errors that can occur from within as well as from service providers.

7.2.1 Reply Standards

A launcher will always be the recipient of any answers and failures from a module. However, the Frontend is always the initiator of the chain, by requesting something from the launcher. In order to find out how to standardize the replies, we need to know what kind of information is interesting.

The Frontend has two kinds of information to gather from replies; operation information and error information. The operation information can be current status of the service or whether the requested operation was successful. That essentially covers the part where the system behaves as suspected. Error information can either be that the service was not accessible or that the service received an exception and failed to complete the request.
CHAPTER 7. DESIGN AND IMPLEMENTATION

7.2.2 Exception Handling Standards

It is needless to say that an application can and probably will at some point of
time, throw an exception. With demands of flexibility with modules and simplicity
towards the Frontend, how exceptions should be handled and raised is an important
issue.

With the possibility to switch modules, the errors and exceptions that may rise
could be different depending on module. If these exceptions are not limited, the
Frontend will need to handle all errors. The Frontend could end up needing to
handle exceptions for all implementing modules, making it a long list. The kinds of
exceptions that can be thrown from modules of a certain interface/service need to
be limited.

Service Related

With the Backend of the Portal making use of distributed resources, errors with
remote instances must be regarded and handled. If there are problems accessing
a service required for some incoming request, the Backend can not go on and will
have to terminate. Errors might also occur at the remote instance and these will
also lead to a halt in the Backend module implementations. Since different kinds of
services are accessed, each may contain a long list of unique errors and exceptions
that may arise.

The service related errors encapsulates errors that are caused by:

• the server being unreachable

• an exception with unknown cause from the service

HG Related

Errors that are strictly related to the HG should be handled specifically. When
trying to access a device through VCM, an indication is shown if the device was not
found. Even though it is not yet implemented, there could be interactions towards
the operator that gives similar indications.

This might overlap with configuration related errors, but providing with a more
detailed cause for the error is a good reason to separate this.

Configuration Related

In the course of accessing some service, there will more than likely be some specified,
required parameters. There are three ways to retrieve the value for a parameter in
a Backend module:

1. argument in method (received from Frontend)

2. hard coded constant/variable in the class
3. accessed from configuration file

Looking at the variants, only one of them will have the parameter value controlled by the class. That means that there is a rather high risk of having external errors affect the execution of the module. These cases should be handled in order for stability and smoother debugging.

List of Portal Exceptions

Note that even though there are limitations on which exceptions are allowed to be thrown, a unique message for each cause can be added. That way the exceptions can still be logged and used for debugging purposes. Additionally, other exceptions may be appropriate to introduce and can be added but will require that the Frontend can handle them correctly.

- com.tilgin.portal.backend.ServiceCausedException
- com.tilgin.portal.backend.ConfigurationCausedException
- com.tilgin.portal.backend.HGNotFoundException

7.3 API

In order to provide various services through the Portal, the Frontend and Backend need to have a uniform view on how to communicate. The interaction will be initiated by the Frontend and terminated with a reply from the Backend. Since the Backend does not initiate any communication with the Frontend, an API for the Backend will be sufficient.

7.3.1 HG Locator

Mapping a user to an HG is necessary if a per user customized view is to be provided by the Portal. How operators typically store this information is not known, but it should reasonably be stored in either:

- the user authentication database
- a different database
- a directory service

As said, which one of these that operators use is not known and of course, different operators may differ in this choice. This requires support for different modules, providing access to user data through one of the above services.

The idea is to let HGDevice (detailed in 7.3.2) retrieve this information. HGDevice will both contain configuration information as well as provide methods for self
configuration. Thus it seems fitting to let that component locate the HG of a specified user. The Frontend needs one access point for finding the HG of a user, and that will be the Backend class HGIdentificationManager.

### 7.3.2 HG Device

A primary use for the Portal is per user customization and self configuration. These goals are reached through the use of the Backend class HGDevice. For both efficiency and simplicity, an instance of the HGDevice class is intended to be created and cached in the Frontend. The Frontend can then keep cached information contained in the HGDevice object, and on demand send a refresh request on the object. The configurations that can be set on the HG should also be reached through this HGDevice instance. Contacting the device to perform a reboot or factory reset should be accessible here as well. Adding up, HGDevice will need to provide:

- HG information retrieval
- storage of latest configuration
- contact interactions
- methods to change HG configuration

Let us start with how to store the configuration. Keeping this information directly in the HGDevice class would clutter the class with getters for each data retrieval. Instead, the data is stored in an encapsulated object HGConfiguration. This way, all the data about the HG can be stored inside that object and a single getter is all it takes to retrieve the information. The HGConfiguration object is accessed through the method getConfiguration().

The HGDevice class needs to have a method that sends requests to VCM and possibly other instances to receive information on the device. This corresponds to the method reloadConfiguration().

For the prototype, the only contact interaction implemented was the reboot. This was simply because the others may or may not be useful for the Portal, and also because they are more or less identical in implementation. To reboot the HG belonging to the user, the method rebootHG() is used.

Changing the configuration of the HG is done through the use many different methods. Essentially, one configuration corresponds to one, single method. The ones available in the prototype are setFirewall() and setStorage().

- com.tilgin.portal.backend.HGDevice.getConfiguration
- com.tilgin.portal.backend.HGDevice.reloadConfiguration
- com.tilgin.portal.backend.HGDevice.rebootHG
CHAPTER 7. DESIGN AND IMPLEMENTATION

Figure 7.6. HGDevice interaction represented in a sequence diagram.

- com.tilgin.portal.backend.HGDevice.setFirewall
- com.tilgin.portal.backend.HGDevice.setStorage

Figure 7.6 shows a sequence diagram of the interaction with HGDevice when the Frontend wants to update information of the HG (i.e. calling the reloadConfiguration method).

7.3.3 Hard Drive Accessor

As with other services, when providing access to a hard drive connected to the HG, different directions are possible. That said, the use of changeable modules is an advantage here as well. In this case, it is the technique with which the hard drive is accessed that can differ.

Tilgin has software for their HGs that supports hard drive usage through USB. The version available at the time of the thesis allowed for LAN and WAN access through HTTP/WebDAV. Regardless if any future technique is added, the service provided will most likely be to list, download, upload or remove files. These uses will together form the interface HarddriveAccessor.

The class accessed by the Frontend is the HarddriveManager. The HarddriveManager will be injected with the appropriate HarddriveAccessor and use that instance to
access the hard drive. Figure 7.7 shows a sequence diagram of a typical usage of the class.

### 7.3.4 API Listing

- com.tilgin.portal.backend.harddrive.HarddriveManager
- com.tilgin.portal.backend.HGDevice
Chapter 8

Experimental Results

The Portal relies quite heavily on distributed resources, the Backend especially uses these as a main source of information. As a result, the Portal acts as a middle man between the service and the end user. An interesting question arises: does this induces any delay in accessing the resource, compared to accessing it with a simple client?

8.1 VCM Connector Delay Test

Since the Portal is used as an interaction point to VCM Connector, it is interesting to know if this adds a delay. In this situation, a delay signifies an increased access time for a single request; issued at a random point of time. This can be tested by comparing accessible speeds through the Backend with those of a "thin client". This thin client should be as simple as possible, only containing required features to interact with VCM Connector.

In order to test this, VCM connector needs to be accessed through both implementations.

8.1.1 Test Setup

The purpose of the delay test is to test time passed between issuing a request and receiving a reply. From that information, we want to find out whether or not the Backend induces a delay. That said, it is not the purpose of this test to investigate how well VCM Connector handles multiple access. For that reason, there is no reason to send multiple requests simultaneously. That might actually interfere with the original aim of the test; sending multiple requests add more pressure to VCM Connector, which might increase the processing time for VCM (time which is included in our measured time).

A total of 100 requests, with an interval of 1 second, will be sent to VCM Connector through both implementations. For simplicity, the functionality through
VCM Connector to be tested is enabling the hard drive storage. When the test is started, the feature will already be enabled.

Testing of the Backend implementation will be done through HGDevice. The thin client will be tested through the class ThinConnectorClient. This client provides with a single method that enables the storage. The method used will directly interact with VCM Connector, using only fundamental code.

### 8.1.2 Test Results

Figure 8.1 shows the results of the delay test; access times for each run and the average access time for the implementations. In the graph, there are some results with significantly higher values than most runs. These spikes occur for both implementations and when the tests were run again, similar results were achieved. This leads to the conclusion that the behaviour is most likely caused by the remote service, i.e. the VCM server. In each run, both implementations will send a request to VCM Connector. The time for a reply from VCM Connector is included in the access time of a run, and will therefore directly affect the access time of the run.

Looking at figure 8.1, it is clear that both implementations provide similar access times. There is a slight difference in the intervals 55-70 and 70-85, but as can be seen in table 8.1, the mean value shows only a marginal difference. Even though
CHAPTER 8. EXPERIMENTAL RESULTS

<table>
<thead>
<tr>
<th></th>
<th>Min</th>
<th>Max</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backend Client</td>
<td>60</td>
<td>307</td>
<td>74</td>
</tr>
<tr>
<td>Thin Client</td>
<td>56</td>
<td>466</td>
<td>70</td>
</tr>
</tbody>
</table>

Table 8.1. Listing the minimum, maximum and average access times resulting from the delay test of the Backend and Thin Client implementations.

the Backend goes through some additional steps, such as retrieving certain objects through Dependency Injection (see section 7.1), it can still provide a service that is as responsive as a thin client.
Chapter 9

Conclusion & Future Work

Let us summarize the information and experience gained through this paper.

9.1 Conclusion

An ambition for the investigation of the Backend was to find means to provide a modular and flexible solution for the Portal. This was desired so that the Portal could be realized as a solution for different operators, with different kinds of systems. Gathering the list of use cases was an important step towards this goal, and helped form the idea of the Portal and what it should be able to provide. The practical appliance of Portal from an end user and an operator point of view are two big and important matters. The latter is still unanswered though, as it was not a part of this thesis. The operators use and benefits of the Portal needs to be regarded if or when Tilgin decides to go through with the Portal as a product.

The evaluation on techniques that can be utilized to achieve remote access showed that there is no silver bullet. The solutions all looked promising at first glance but also came with some flaws. One has to be realistic and prioritize certain benefits over others in order to reach a decision for a future implementation.

9.1.1 Portal and the Prototype

What became clear early on in the thesis was the fact that the Portal was not to be seen as a new WebUI, with the only difference being that it is accessible from the Internet. Neither was it to become a web page providing mirrored functionality to all VCM Connector settings.

Constructing the prototype brought ideas of the Portal to life. The adoption of Dependency Injection (DI) showed much promise for customizability, providing simple and straightforward mechanisms for usage. Changing a configuration file is all the required work to switch a module and effectively, use a different implementation. This makes it easy to customize the Portal for different operators, or to let it be customized by operators themselves.
CHAPTER 9. CONCLUSION & FUTURE WORK

Will the flexibility demands reduce the performance of the Backend? Although many benefits can be gained through the Portal, there might be a significant processing overhead when accessing distributed resources such as VCM. The experiment in 8.1 was specifically aimed to test this, by examining access times to a VCM Connector service. Access times were compared between the Backend prototype and a thin client, which only performed the most fundamental tasks. The results show that the Backend prototype, compared to a thin client, does not add any overhead in service access times. This confirms that the Inversion of Control (IoC) container used, the Spring IoC container, is perfectly capable of delivering the wanted features of an IoC container without bringing down performance.

Now that it is certain the Backend can meet the inherent demands, the operators and their needs should be targeted.

9.1.2 Remote Access

The UPnP RA solution gives full access to home devices, allowing UPnP devices in the home to be securely accessed through the use of a VPN tunnel. The use of VPN is both the strength and drawback of UPnP RA. The technique enables a remote device to securely interact with devices in the home but it also hinders the flexibility of the remote device.

The IMS approach does not directly have this issue since it does not have the need for VPN. But looking at how device synchronization is handled in this solution, it is clear that this part is not standardized. The proposal suggests that UPnP RADA can be used for the synchronization, but that would lead back to the VPN issues. The alternative is to make a non-standardized solution, such as making the HG act as a UPnP client in order to find home devices, push device information to the remote device and somehow transfer data from a home device to the remote device. Evidently, this alternative does not provide with a standardized strategy and actually increases the risk for hasty and quick-fix solutions in the implementation phase. Therefore, use of UPnP RADA is more suitting.

Similarly to the IMS approach, the Web based approach does not explicitly describe how device discovery is handled. The HG should somehow retrieve device information and show these to logged in and authorized users. As the proposal does not consider the media transmission either, the overall specification received by interpreting the approach leaves something to be desired. As previously mentioned, the benefits of standards are, amongst others, to avoid hasty and quick-fix solutions. The Web based approach is simply not specified enough to alone provide those benefits.

What seems to be the strongest candidate is UPnP RA, even though there are some drawbacks. At the time of writing, the technique is still being standardized but there are hopes for widespread use on devices. In the end, the IMS solution’s real strength over UPnP RA is access to SIP devices. For that reason, I would only recommend the IMS solution if Tilgin has many customers using IMS, to whom they want to offer a remote access solution. Otherwise, UPnP RA should be the choice.
for remote access. So once again: can a system for remote access be provided, that is secure, user friendly and highly interoperable with devices? Looking at UPnP RA, the answer is clearly yes. Only the user friendliness might be hindered, but as said previously, this can be eased with SSL VPN.

There is also the possibility to use the Portal for remote access purposes. In the case of UPnP, the Portal could take the role of a remote access client. A VPN would then be set up between the HG and the Portal. That way, remote access could be provided to devices by using any web browser and without the need for setting up VPNs and credentials. Although this option might provide lesser performance than with direct use of UPnP RA on the remote device, there are benefits regarding end user friendliness. A similar solution could be used with the IMS approach, where the Portal also incorporates an IMS application server providing remote access to non-IMS devices.

9.2 Future Work

A very important issue is still unresolved: in what form should the Portal be provided to operators? Key questions that should be targeted are:

- Should Tilgin deliver a fully flexed solution or an extensible solution?
- How to sell and market the Portal as a product to customers?
- Should the Portal be provided solely as a web application or as a service layer?

If a full solution is to be provided, more investigation about the operators will be necessary; what user database systems are used, how the mapping between user and HG is handled. Or, if the complete Portal system is not to be delivered, leave this to be done by the operators. The issue is then how to handle an integration of the Portal for each customer.

Event-driven messaging with VCM Connector should be further investigated, as well as which technique to use in order to provide it. Specifically, the investigation should try to find out whether the migration cost to Axis2 is outweighed by the gains and inherent features of the newer version, with regard to other possible techniques (RSS etc.) that does not require any upgrade.
Bibliography


BIBLIOGRAPHY


BIBLIOGRAPHY


Appendix A

Spring IoC injection example

Method that utilizes the IoC container to retrieve and return an object that implements the interface HGLocator.

```java
private HGLocator getHGLocator(){
    Resource resource = new FileSystemResource("spring-beans.xml");
    BeanFactory factory = new XmlBeanFactory(resource);
    return (HGLocator) factory.getBean("hglocator");
}
```

Example class that can be dependency injected through setter injection.
Appendix B

Spring IoC configuration file example

Sample file with two beans configured, showing how an alias can be used to signify the bean in use. Imagine the scenario that a WebDAV module is used for the hard drive interaction (as in the prototype solution), and the module is to be replaced by a FTP implementation instead. If the name field in the alias parameter is changed to `ftpAccessor`, a class that requests and needs the bean `hdAccessor` will not need any code change. Only the new module (`FtpAccess`) is needed since both modules conform to a uniform interface.

```xml
<beans>
    <!-- Change the name to use a different bean for acsconfigurations -->
    <alias alias="hdAccessor" name="webdavAccessor"/>

    <bean name="webdavAccessor"
          class="com.tilgin.portal.backend.harddrive.HttpWebdavAccess">
        <!-- Any additional bean configurations.. -->
    </bean>

    <bean name="ftpAccessor"
          class="com.tilgin.portal.backend.harddrive.FtpAccess">
        <!-- Any additional bean configurations.. -->
    </bean>
</beans>
```
Appendix C

Prototype Class Diagram

Figure C.1. Class diagram of the prototype.