TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY

Department on mechatronics

Chair of Mechatronics Systems

MHK70LT

Caspar Romot

SECURITY SOLUTION IN MODERN CONNECTED CONTROL SYSTEMS

Master's Thesis

Author applying for

Master's of Technical Sciences

academic degree

Tallinn

2015

AUTHOR'S DECLARATION

I hereby declare that this thesis is the result of my independent work on the basis of materials not previously applied for an academic degree. All materials used in the work of other authors are provided with corresponding references.

The work was completed under Professor PhD Mart Tamre's guidance
"....."....2015 a.

The author ..... signature

The work meets the requirements for a master's work.
"....."....2015 a.
Supervisor ..... signature

Permit to defense
..... curriculum defense superior
"....."....2015 a.

..... signature

TUT Department of Mechatronics
Chair of Mechatronics Systems

MASTER'S THESIS SHEET OF TASKS

2015 academic year, spring semester
Student: Caspar Romot 132938
Curricula: MAHM02/13
Specialty: Mechatronics
Supervisor: Professor PhD Mart Tamre
Advisers: Jüri Toomessoo, Principal Software Architect at ICD Industries Estonia, +372 5201487

MASTER'S THESIS TOPIC:
(in English) **Security solution in Modern Connected Control Systems**
(in Estonian) **Moodsate ühendatud kontrollsüsteemide turvalisus**

Thesis tasks to be completed and the timetable:

Nr	Description of tasks	Timetable
1.	**Problem statement and analysis.** Description of problem, possible dangers and examples. Comparison and analysis of available solutions and further research tasks	10.04.2015
2.	**Designing graphical user interface** for managing remote network connections	20.04.2015
3.	**Designing code architecture for connecting to remote network** to set up a secure connection for further usage	25.04.2015
4.	**Designing code architecture for connection forwarding** to be able to connect to devices in private networks otherwise inaccessible	15.05.2015
5.	**Analysis of the work done**, future uses and possible improvements, based on the example of ICD Industries Estonia software CDP Studio	21.05.2015

Solved engineering and economic problems:
Software deployment and information transfer via the internet will be done using an encrypted protocol, thus providing a convenient and secure method for managing remote systems. A software plugin will be created that is easily extensible for multi-purpose usage.
Language: English
Application is filed not later than 15.05.2015
Deadline for submitting the thesis 22.05.2015

Student: Caspar Romot date

Supervisor: Mart Tamre date

TABLE OF CONTENTS

EESSÕNA **6**

1. INTRODUCTION **7**
1.1. TERMINOLOGY 7
1.2. CHOICE OF TOPIC 7
1.3. OVERVIEW OF THE TOPIC 8
1.4. GOALS OF THE WORK 8
1.5. MARKET ANALYSIS 10
1.6. USED TOOLS AND METHODS 10
1.7. CONFIDENTIALITY 11
1.8. OVERVIEW OF THE THESIS 11

2. EXAMPLE OF A USE CASE **12**
2.1. EXAMPLE OF DEPLOYING A CDP PROJECT 12

3. POSSIBLE SECURITY SOLUTIONS **14**
3.1. USABLE SSH LIBRARIES 15
3.2. CHOOSING THE BEST SSH LIBRARY 16

4. SOLUTION WITH QSSH **18**
4.1. POSSIBLE PRINCIPLES OF OPERATION 18
4.2. USING SECURE SHELL 20
4.3. EXISTENT CODE 23
4.4. MAINTAINING INDEPENDENCE FROM THE PLUGIN 23

5. USER INTERFACE SOLUTION **24**
5.1. USER INTERFACE SOLUTION CONCEPT 24

6. INITIAL CONNECTION **27**
6.1. LOCAL AND REMOTE CONNECTIONS 27
6.2. INVERTING THE DEPENDENCY FOR REMOTE CONNECTIONS 28
6.3. IMPLEMENTATION OF IREMOTECONNECTIONMANAGER 31
6.4. CODE ARCHITECTURE 31

7. SSH TUNNELING **34**
7.1. SSH TUNNEL 34
7.2. CREATING AN SSH TUNNEL 35
7.3. DEFAULT TRANSLATION IMPLEMENTATION 38
7.4. PROVIDING IMPLEMENTATION FOR TRANSLATOR INTERFACE 40
7.5. NETWORK ADDRESS TRANSLATION IMPLEMENTATIONS 44

8. FINAL SOLUTION **46**
8.1. FULL DESIGN 46
8.2. WORKING PRINCIPLE WITHOUT REMOTE PLUGIN INSTALLED 48
8.3. WORKING PRINCIPLE WITH REMOTE PLUGIN INSTALLED 50

9. FUTURE IMPROVEMENTS AND USES **55**
9.1. FLAWS AND SHORTCOMINGS IN CURRENT DESIGN 55
9.2. FUTURE DEVELOPMENTS 55

10. SUMMARY **60**
10.1. GOALS OF THE WORK 60
10.2. SOLUTION 60
10.3. PRACTICALITY AND FUTURE DEVELOPMENTS 62

11. KOKKUVÕTE **63**
11.1. ÜLESANDE TAGAMAAD 63
11.2. ÜLESANNE 63
11.3. LAHENDUS 64
11.4. PRAKTILINE VÄÄRTUS JA TULEVIKUARENGUD 66

12. REFERENCES 68

EESSÕNA

Käesoleva magistritöö ülesanne põhineb Norra ettevõtte ICD Industries Eesti haru ICD Software poolt väljatöödatavale tarkvarale CDP Studiole seatud nõuetel. Antud lahendus leiab rakendust CDP Studio igapäevase kasutamise osana, kontrollsüsteemide operaatori ja kontrollerite vahelise suhtluse loomiseks. Kogu töö on tehtud ICD Software algatusel ja CDP Studio jaoks ning töö detailid ei kuulu kolmandatele osapooltele jaotamiseks.

Tööks vajaliku materjali kogumisel ning konsulteerimisel abistasid autorit sellised ettevõtte töötajad nagu Jüri Toomessoo, Riho Pihlak, Armin Riiet ja Peeter Salong. Samuti oli suureks abiks TTÜ professor ja Mehhatroonikasüsteemide õppetooli juhataja Mart Tamre. Autor tänab kõiki lõputöö koostamisel abiks olnud isikuid.

1. INTRODUCTION

1.1. Terminology

In this thesis a number of technical and contextual terms are used. The less commonly known terms are listed and explained in this section.

- Qt – an open source software package for creating cross-platform software applications [1]
- Qt Creator – an open source integrated development platform (IDE) for creating Qt applications [1]
- CDP – Control Design Platform. It is a software package being developed by ICD Industries for creating multipurpose control systems for several different platforms including Linux and Windows operating systems [2]
- CDP Studio – a software package being developed by ICD Industries and built upon the Qt Creator. CDP Studio is essentially a number of plugins created for Qt Creator. CDP Studio is meant to develop and maintain CDP applications
- SSH – Secure Shell. It is a cryptographic data transfer protocol [3]
- QSsh library – a library of Qt Creator that is used for creating, maintaining and using SSH sessions
- Component – an integral piece of programming code like a library or a plugin

1.2. Choice of topic

The topic was chosen because the author of the thesis works in ICD Industries as a software engineer, developing CDP Studio. The CDP Studio or Control Design Platform Studio is a software product that is currently (April 2015) being developed to be an extension for CDP. It is going to be a user friendly IDE for developing CDP applications.

The task explained in this thesis is what the author is currently (April 2015) working on. Security is extremely important in any modern system due to both personnel safety and intellectual property concerns. This is especially amplified when dealing with systems that have to be accessible via the internet or systems that are physically large and potentially harmful if used improperly. This is the case with many CDP-operated systems because this software is mostly used on large ships and gangways.

1.3. Overview of the topic

The CDP Studio is going to be a user-friendly integrated development environment built upon a free software platform called Qt Creator and meant for developing CDP applications for different control systems. CDP Studio gives users all the tools necessary for developing, building, deploying and maintaining projects of CDP. The latter two tasks inherently involve some sort of communication with the controllers that CDP applications are to be run on. This is where the security issue arises. The Studio has to guarantee a safe connection from the operator's machine to the controller that is impenetrable by hackers and unobservable by network monitoring. As explained, this is important to ensure the physical safety of people working on the controlled systems as well as guarantee the secrecy of the company's intellectual property.

Currently many control systems are not manageable via the internet, which means that operators have to physically go to the controllers, plug their computers into the local network and program the controllers this way. It is also the case with CDP applications, because currently no convenient and secure method is available that would allow the operators to do this over the internet. Of course it is possible, to connect to the network via either a virtual private network (VPN) or SSH protocol, which would ensure the safety of the connection, but deploying or maintaining the system without proper complimentary software would be very uncomfortable and slow for the user. For deploying a new program (i.e. CDP application) into a controller one would have to manually transfer all the files from their machine to the controller but when there are hundreds of controllers in a system, this would be a very tedious task and a huge waste of time. The main problem that the thesis at hand addresses is how to make remote systems easy to maintain and how to automate as much of the manual labor associated with deploying programs as possible while maintaining the same level of security as when doing all that in a private network.

1.4. Goals of the work

Qt Creator is an IDE that is meant for developing and maintaining Qt applications. Since CDP Studio is built upon Qt Creator that functionality carries over. Therefore CDP Studio enables users to do the following:

1. Connect to local devices using SSH protocol

2. Deploy CDP applications to those devices
3. Monitor CDP applications on the devices
4. Scan for controllers on the local network

The goal of the project covered in this thesis is to create a plugin (hereinafter called remote plugin) and the necessary complimentary code for CDP Studio that extends that functionality to remote networks and devices in remote private networks. As it is not safe to make all controllers in a control system accessible straight from the internet, they should be behind a firewall machine that is the only gateway from the internet to the devices. The remote plugin should provide the possibility to access devices hidden behind firewall machines and allow services to perform the aforementioned actions on those machines exactly as if they were local devices.

This thesis focuses on using the remote plugin for deployment of CDP projects but the resulted work can be used by other services as well.

This plugin is set to be an optional piece of software for CDP Studio. Users who have not installed the remote plugin however have to still be able to perform the aforementioned tasks on a local network. This means that the plugin is only needed for connecting to remote network devices. This implies that CDP Studio can not depend on this optional piece of software as it might be absent.

In the future this plugin should allow users to connect to several different remote networks simultaneously and also use the remote plugin for other purposes than just deployment, like for example creating database connections to devices in remote networks. These issues are outside the scope of this thesis but they have to be acknowledged to make the remote plugin easily expandable.

The solution must therefore meet the following requirements:

- CDP Studio can not depend on the remote plugin
- Services that provide system maintenance functionality should not know whether the device connected is a local or a remote device because the plugin is optional and services can not depend on it
- All communication must be encrypted

- CDP Studio must enable local system management even without the remote plugin
- The plugin has to be expandable to enable creation of several simultaneous network connections
- The plugin has to be expandable to be used by other services besides application deployment service

1.5. Market analysis

1.5.1. Available solutions

The solution required has to be compatible with Qt Creator, because CDP Studio is based on it. No plugin can be found online that would achieve the goals stated in this work and it is therefore unavoidable that ICD Software has to write it.

Qt Creator supports local project deployment and maintenance and this functionality can be used in the creation of the remote plugin. To optimize the development of the final solution it is useful to make use of this existing code and integrate it into the remote plugin's work.

1.6. Used tools and methods

As mentioned above, the CDP Studio is being built upon a software package called Qt Creator. Qt Creator itself is built using Qt, which is an open-source software package for building applications for all types of platforms. To ensure cohesion within CDP Studio source code, ICD Software uses Qt's and Qt Creator's API (application programming interface) as much as possible. This applies to the task covered in this thesis as well.

C++11 is used as the programming language throughout the CDP Studio project and the remote plugin.

When writing code for this task the author follows the so-called S.O.L.I.D principles [4]. These ideologies help create code that is easy to read, simple to expand upon and fool-proof to use. The whole process of code writing is also done following the method of test driven development (TDD) [5] which means that the whole code is covered with automated tests that check the integrity and functionality of the final product. TDD and the S.O.L.I.D principles work hand-in-hand thus making the written code user- and developer-friendly.

1.7. Confidentiality

Due to CDP Studio being a commercial product, it is the author's responsibility to ensure the confidentiality of most if not all of the production code. Therefore this thesis will mostly demonstrate pseudo code and illustrations instead of the actual code used in CDP Studio.

1.8. Overview of the thesis

In this thesis the main goal is to create a plugin for CDP Studio that enables clients to not only connect to local controllers, running CDP applications but also controllers located on remote networks that are hidden behind a gateway machine. The plugin has to enable users to create secure networking sessions to those remote networks to deploy new applications into controllers and to monitor the work of those controllers or even the whole network. The plugin has to be built so that CDP Studio works without it as the plugin is to be sold separately from the CDP Studio.

2. EXAMPLE OF A USE CASE

In this section an example of a user problem will be given. An overview of the use case for the solution is illustrated on Figure 2.1.

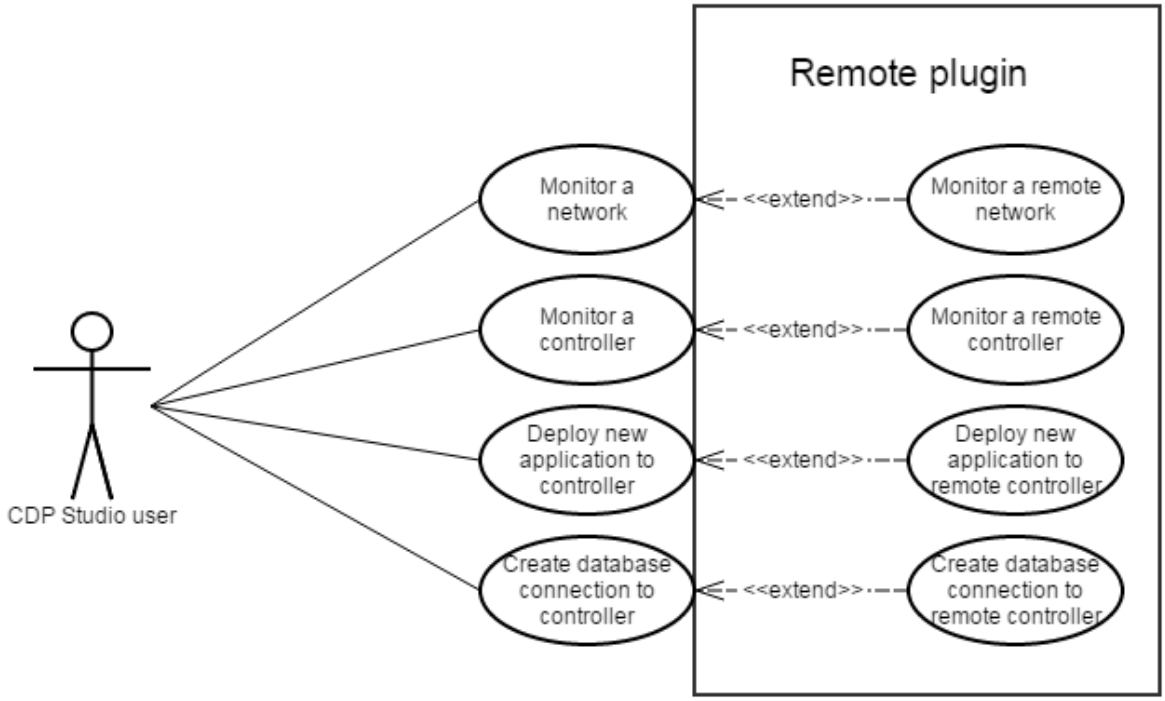


Figure 2.1 Use case

2.1. Example of deploying a CDP project

Let us say the client has created a CDP application that controls the speed of an electric motor and let us say that he/she wants to upload this application to two controllers. One is located in a remote network and it is impossible for him to connect his own computer to that network directly. Let us also assume that the private network which this remote controller is connected to has a server machine that has a public IP address (i.e. it is accessible from the internet) and works as a firewall for that network. The second controller is connected to the same network that the client's machine is in.

For the client the deployment process has to be as simple as possible and he/she does not need to worry about the details of creating a connection to either controller.

For the client the deployment procedure is as follows, when deploying to a local controller (when using the functionality inherited from Qt Creator):

- Insert the IP address and SSH port of the controller in the system configuration page of CDP Studio
- Specify the SSH clients username and password or public key
- Click “Connect” to connect to the controller
- Click deploy

Connecting to a remote device is not currently possible with the functionality provided by Qt Creator, so this procedure is defined by ICD Industries:

- Insert the IP address and port as well as authentication details of the gateway server (the public firewall machine) in the system configuration page of CDP Studio
- Click “Connect” to connect to it. IP scan is then performed automatically and a list of available machines is generated
- Choose a suitable controller and insert the authentication details for it
- Click “Connect” to connect to it
- Click deploy

The deployment of the project is then handled automatically without any further intervention from the client as if the connection was created to a local device.

3. POSSIBLE SECURITY SOLUTIONS

Solutions that would achieve similar goals stated in this thesis involve setting up a VPN (Virtual Private Network), SSH (Secure Shell), Telnet or another similar server on the remote network and manually connecting to that server from the client's ´machine to perform the necessary tasks by hand. There are however several drawbacks to all of these approaches.

VPN server would allow creating a secure connection from the client's machine to the remote network, which would make the network there accessible for the client as if it was a local network. This would allow the user to monitor the system running on the controllers on that network and even transfer files from his/her machine to the controllers to deploy new programs. But it would all have to be done outside the CDP Studio and it would be very inconvenient for the user, especially considering it would have to be done manually.

SSH server would allow the user to do basically all the same things, but without a proper graphical user interface (GUI) it would be even more inconvenient.

Telnet and other similar protocols would be even worse, because they do not provide any encryption for the data that is being transferred. Therefore these solutions are out of the question.

As the CDP Studio is basically a bunch of plugins for Qt Creator it inherently has all the functionality of it built in. Qt Creator supports project deployment into local machines and it works similarly to how it is expected to work in CDP Studio. This said it is only logical that it should be used to its full potential to minimize the work, to achieve the final goal.

This existing deployment mechanism uses SSH as transmission protocol so it would make sense, if the remote plugin used the same methodology. Also when using the same methods as the existing code, it is easy to make the local and remote connection more convenient and seamless for the services using the connections and the system operator. This is why in this work SSH is chosen over VPN – to keep a uniform design in the software and reuse as much of existing code as possible.

3.1. Usable SSH libraries

In this section an overview of the most common free SSH libraries is given, to choose the most optimal one to be used in the remote plugin.

3.1.1. Libssh

One of the two most commonly used free SSH libraries is libssh [7]. This library has an API for both the SSH client and server side functionality and it supports several file transfer protocols like SFTP (SSH File Transfer Protocol) and SCP (Secure Copy). It can be used in both blocking and non-blocking mode and the developers of it claim it to be thread-safe. Another important factor that is essential in this work is the possibility to create TCPIP tunnels via SSH sessions. The good side of this library is the fact that it is constantly seeing new updates. The latest stable version is 0.6.0 as of April 2015.

3.1.2. Libssh2

Libssh2 library is somewhat similar to libssh library but there are some differences that the developers of libssh2 have brought out on their webpage [7]. These differences can also be seen in Table 3.1

Table 3.1 Differences between libssh and libssh2

Library	libssh2	libssh
License	BSD	LGPL
Server-side support	no	yes
GSSAPI authentication	no	yes
Elliptic Curve Key Exchange	no	yes
Elliptic Curve Hostkeys	no	yes
Automated test cases with nightly tests	no	yes
Stable API	yes	mostly
C compatibility	C89	C99
strict namespace	yes	no
man pages for all functions	yes	no
Doxygen documentation for all functions	no	yes
Tutorial	no	yes
SSHv1 support	no	yes
Build concept	Autotools	CMake

It is clear that both libssh and libssh2 would be suitable for the task at hand because only client side implementation is needed. A thing that makes libssh2 better according to this comparison is the stability of the API. A stable API ensures that the libssh2 source code can be updated without its users needing to re-implement their existing code.

3.1.3. QSsh

The third SSH library that was considered for this work was that of Qt Creator's. This is the library that Qt Creator uses by default for all local area network project deployment. Two of the main advantages of using this option are:

- It is a much higher level library, meaning that less work is required to implement the client side.
- Using this option allows using a lot of existing code without rewriting it with another library. For example the deployment of projects already uses QSsh so it does not need to be rewritten to use another library.

3.2. Choosing the best SSH library

All the described SSH libraries are suitable for this project. However there are some considerations that come to play when choosing the best solution.

The biggest thing is the amount of work required when using each library. Since Qt Creator uses QSsh for SSH networking, some of the functionality is already implemented. Although the usage of the remote plugin is out of the scope of this task it simplifies the integration of the work into the CDP Studio project.

Another aspect that speaks in favor of QSsh is the fact that using it preserves the consistency of code in Qt Creator and does not bring in any additional dependencies into the project. Any external library adds complexity into CDP Studio and this is unwanted.

Third consideration that favors using QSsh is that the usages of both local and remote connections have to be seamless. Since Qt Creator has the functionality to create local SSH sessions which uses QSsh (which can be exploited for the remote plugin), making it seamless with any other library would be almost impossible without major changes in Qt Creator's code.

All things considered, the amount of work that using QSsh eliminates is so substantial that it is logical to use that.

4. SOLUTION WITH QSSH

In this section of the thesis the main principles of the solution are set to place, considering the fact that the chosen SSH library is QSsh. This involves the working principles and the user interface for using the system.

4.1. Possible principles of operation

Based on the existence of available code in Qt Creator there are a number or possible options for solving the task at hand.

One option would be to copy the deployment code from Qt Creator and use it in the remote plugin exactly like it is used for local deployment in the QSsh library except that the SSH session would first be created to the remote network, where it is tunneled to the necessary controller.

Second option would be to use the remote plugin to create a tunnel to the controller in the remote network and then use NAT (network address translation) approach to use Qt Creator's existing deployment code without copying it to the remote plugin.

For clarification of both methods please refer to Figure 4.1

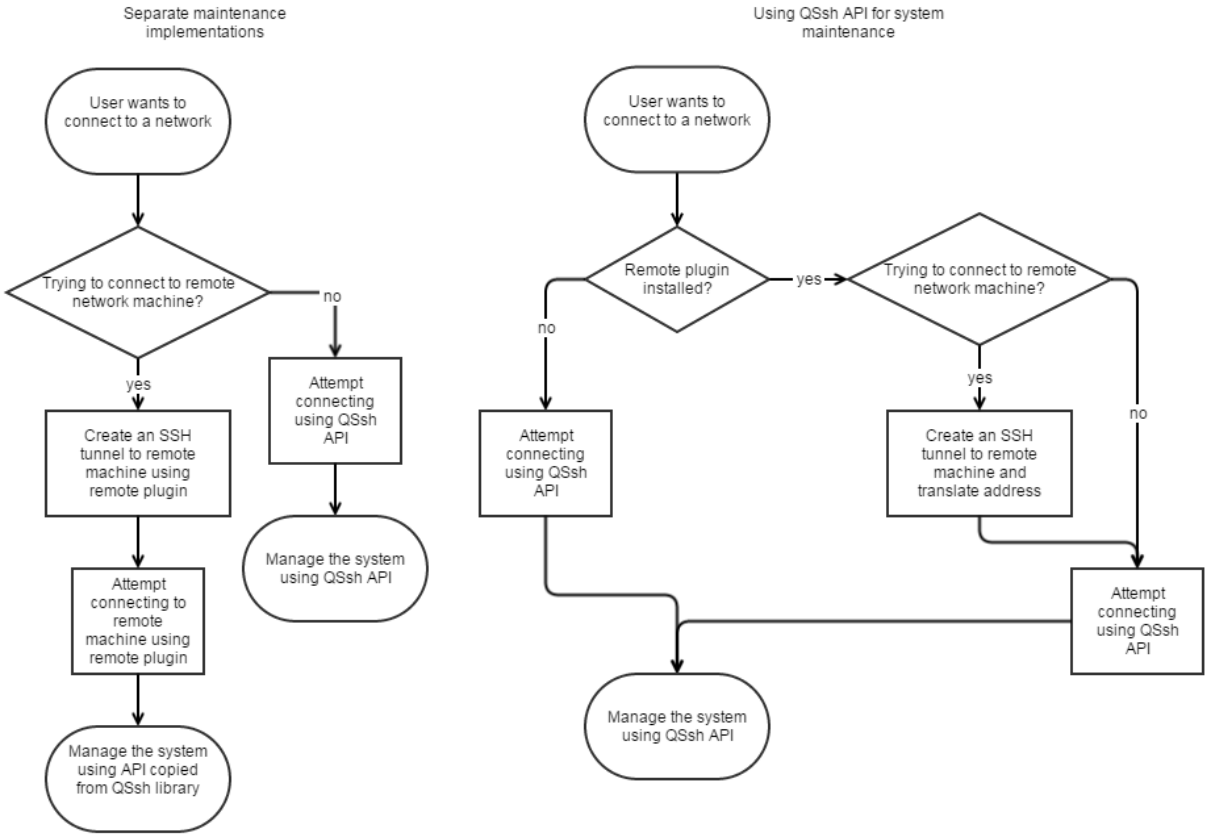


Figure 4.1 Two methods for remote system maintenance

As seen on the Figure 4.1 both methods would yield similar results and would actually meet all the aforementioned requirements for the plugin, if the dependency inversion principle [4] is used properly. However to minimize overhead and work needed, it is more optimal to use the second solution.

Here are the pros and cons of the chosen method:

Table 4.1 Pros and cons of the chosen method

Pros	Cons
No Qt Creator code has to be copied	Some Qt Creator code has to be modified
The remote plugin only has to create a tunnel to the remote machine, whereas the QSsh library handles all the rest	
Very little duplicated code	

One of the biggest drawbacks of this solution as seen in Table 4.1 is the fact that the existing Qt Creator's code has to be changed. This would not be a problem if Qt Creator was also maintained by ICD Software and not The Qt Company [1]. However since this is not the case it means that every modification that is made in Qt Creator's source code for CDP Studio specific reasons and is not appropriate for pushing upstream to the development branch of Qt Creator has to be maintained manually in the future if a new version of Qt Creator is released. When a new Qt Creator release happens a patch needs to be created of the changes made by ICD and after updating the Qt Creator to a new version that patch needs to be applied on top of it. But because the Qt Creator's source code already has changes made by ICD, adding these new ones would not substantially increase the required work for applying the patch. It is however important to minimize the modifications of existing Qt Creator code because conflicts might occur during patching if the developers of Qt Creator have made changes to the same code ICD has.

4.2. Using Secure Shell

The chosen solution will be using SSH protocol for communications between the CDP Studio client and the controllers on site. Before continuing the work the main principles behind SSH have to be understood. This paragraph gives a short overview of the Secure Shell and how it will be used in this work.

In the history there have been two versions of SSH: SSH-1 and SSH-2 [6]. The SSH-1 is mostly considered to be deprecated and thus SSH-2 is used.

The Secure Shell is a network communication protocol that uses encryption to ensure safety of all data being transmitted. When a client tries to connect to a server then the server has to be authenticated, before any communication can begin. If configured accordingly, the same goes for the client. Whereas the server authentication is public key based, the client authentication could happen via a number of different methods, including password authentication, public key authentication or keyboard-interactive method.

In CDP Studio it is required that the user must enter all the authentication information into a special form and if the information is correct, the connection should then be created without further client intervention. This eliminates the possibility for any interactive authentication

methods. CDP Studio requires that the SSH servers on sites have to allow either SSH public key authentication or simple password authentication or both.

4.2.1. SSH sessions and tunneling

When a client wants to connect to a local network controller or even to a publically accessible remote machine it is a simple case of just creating an SSH session from his/her machine to the SSH server on that controller. This is illustrated on Figure 4.2.

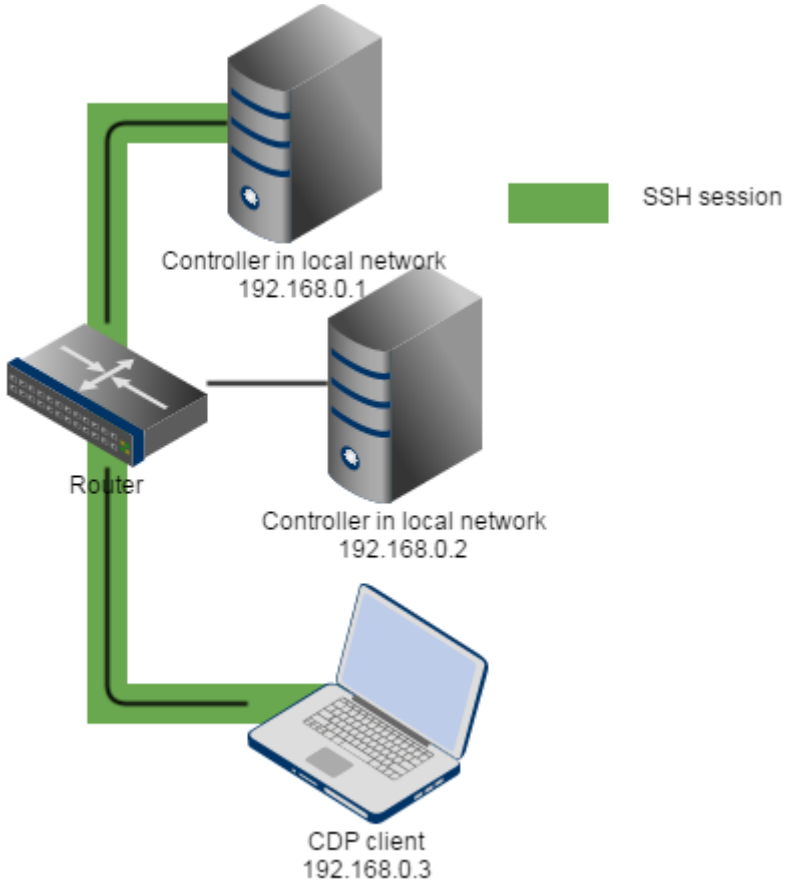


Figure 4.2 Local network SSH session

However if the client wants to connect to a controller that is located on a remote network and is not directly accessible via the internet but is instead behind a gateway (or a firewall), things get a little more complicated. In that case the procedure is as follows (see Figure 4.3)

- Client creates an SSH session to the SSH server located on the gateway machine. This is the “First SSH Session” on the illustration. In this example, this session is created to the following IP address and port pair: “10.1.2.3:22” (22 is the default SSH port)

- TCP tunnel is created from the client's machine to the necessary controller through the first SSH session and the gateway machine. In this example, let us say that the tunnel is created to the controller with local IP address 192.168.0.1 and the port 5555 of the CDP Studio client machine is forwarded to that controller's IP and SSH port.
- The client creates a direct SSH session from its machine to the forwarded port which connects it to the controller at the end of the tunnel. In this example client has to create the second session to the following IP address and port pair: "localhost:5555". This is the "Second SSH session" on the illustration

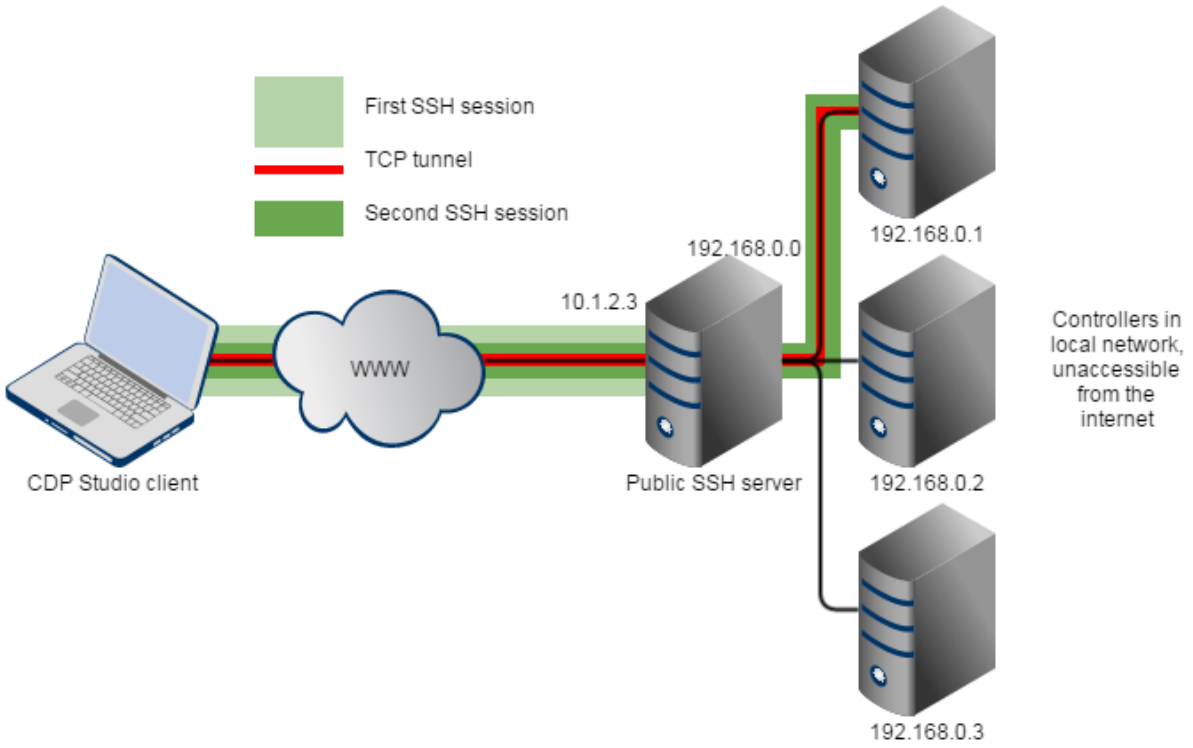


Figure 4.3 Remote SSH tunneling

This is the methodology that will be used within the work described in this thesis, thus hereinafter the following terms will be used:

- Gateway – the machine that that a public hostname and is accessible from the internet. Usually a firewall machine that protects the devices in a subnet it is connected to. In Figure 4.3 it is the machine with the public IP 10.1.2.3
- First SSH session/primary SSH session/preliminary SSH session – SSH session that is created from CDP Studio client to the gateway machine (or the firewall machine)

- Tunnel/TCP tunnel/SSH tunnel – the TCP tunnel that is created through the first SSH session from the CDP Studio clients machine to the end-destination device behind the gateway
- Second SSH session – SSH session that is created from the CDP Studio client's machine straight to the end-destination device behind the gateway

4.3. Existent code

As explained earlier in the thesis the Qt Creator has implementation for connecting to and deploying projects onto local machines. In reality this means that it can also be used for connecting to remote machines provided that they have a public IP address and a running SSH server that is configured accordingly. This means that the code for creating an actual SSH session from the client machine to an SSH server is already done. That leaves the question of connecting to machines that are not accessible straight from the internet.

A very useful piece of code that also already exists is the IP scanner. This provides the CDP Studio the functionality to scan a network for possible controllers or machines to deploy CDP projects onto. This code was written by ICD Software and currently it only works in local networks. Since it is an extremely useful thing to have it is logical that when connecting to a remote network the users would like to use it there as well. Right now however it is not usable for remote networks so it has to be modified a little to use the remote plugin to allow that functionality. This is outside the scope of this thesis but it is implied that the modifications have already been done in order to simplify the explanation of the remote connection handling.

4.4. Maintaining independence from the plugin

A major issue that arises when trying to come up with a solution for this problem is how to achieve CDP Studio's independence of the remote plugin while maintaining the possibility to perform all the actions in a local network. This would not be a problem if the first solution was chosen in paragraph 4.1, but is a major issue when trying to use as much of the existing code as possible. This is where the dependency inversion principle comes to play. The details of this will be explained later in this thesis but the main idea is the following: in the case when the remote plugin is not loaded during CDP Studio startup, the users of the plugin are provided with a default implementation that only allows local communication.

5. USER INTERFACE SOLUTION

Managing of networks (wither local or remote) is done in the system configuration page of CDP Studio. For this a GUI must exist for users to perform actions with remote networks or remote devices. In this section a concept of a suitable graphical user interface (GUI) is presented for connecting to a remote machine that is located behind a gateway machine. The procedure described in section 2.1 is taken as the basis for creating this.

Note that the illustrations shown in this thesis are not representative of the actual design of CDP Studio but simply demonstrate the main concepts.

5.1. User interface solution concept

The user should have a possibility to save several remote locations in a list for future use. Taking the procedure described in section 2.1 as the starting point for this, the first thing that the GUI must provide is a table for the user to enter the necessary information to connect to gateway machines. This includes:

- IP address or host name of the gateway machine
- SSH port of the gateway machine (22 by default)
- Username for the SSH server on the gateway machine
- Password of the user (in case password authentication is used)
- Path to public key file of the user (in case public key authentication is used)

Then the user has to have a way to connect to a chosen gateway machine. For this there has to be a “Connect” button.

When the connection is established, IP scan will be performed. Since the gateway machine might have more than one network interface and thus be connected to many local networks with different subnet masks, it is necessary that the subnet mask is provided by the user as well.

A table for available local machines in that subnet has to be generated. In this table there has to be the local IP address of each machine because it will be used when creating a tunnel to that machine. The user has to insert the authentication details for these machines into the table, so the controllers in that list have to be identifiable by some measure. The MAC address

is the best identifier for this purpose because it is different for each machine and does not change in time unless the networking hardware is changed. The user has to know the SSH username and password or alternatively has to have placed his/her machine's public key into the machines he/she intends to connect to. Concluding from this the table has to have the following fields:

- MAC address
- IP address
- SSH port (22 by default)
- Username
- Password of the user (in case password authentication is used)
- Path to public key file of the user (in case public key authentication is used)

Again there has to be a "Connect" button to initiate an SSH session to that machine.

The user interface table that contains the remote networks can be seen in Figure 5.1. For exemplification three possible remote networks have been configured. The connect buttons are on the right. Note that the design used in these illustrations is not he actual one used in CDP Studio. This is just a representation of the functionality of the GUI.

Remote networks							
No	Hostname	Port	Username	Password	Public key	Subnet mask	
1.	10.0.1.2	22	peter	peter123	/home/peter/.ssh/id_rsa	255.255.255.0	Connect
2.	10.1.2.3	22	henry	henry123	/home/henry/.ssh/id_rsa	255.255.255.0	Connect
3.	10.2.3.4	22	tom	tom123	/home/tom/.ssh/id_rsa	255.255.255.0	Connect

Disconnect

Figure 5.1 Table of remote networks

On Figure 5.2 the GUI is shown in the case where the user has connected to a remote network gateway and the IP scan has been performed, presenting the user a list of available devices. Here the user can select the device he/she wishes to deploy a project into or just to monitor and connect to. When a connection to a remote device is established the appropriate "Connect" button is replaced with a "Disconnect" button which can be pressed to end the SSH session to the corresponding destination.

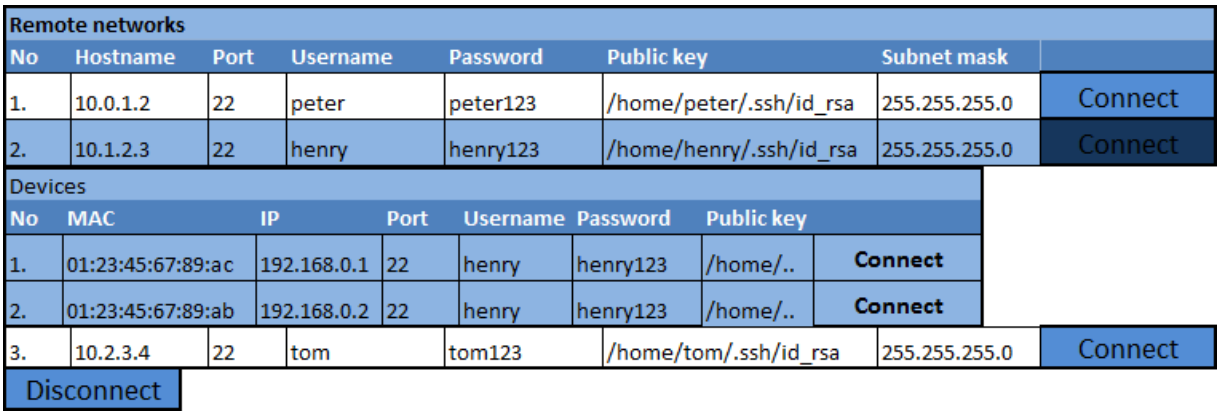


Figure 5.2 List of devices in remote network

If a user wants to disconnect from a remote network all together, he/she must click the "Disconnect" button at the bottom. Since there can only be a single remote network activated at a time, one "Disconnect" button is sufficient.

6. INITIAL CONNECTION

In this section the programming behind the GUI is explained as well as the principles used to achieve the goals set for this work. Creating SSH tunnels and setting up SSH sessions is explained and the workflow for the user is implemented on a concept level.

Please note that the actual code used in CDP Studio is confidential and is therefore not exhibited in this thesis but instead the main concepts are presented.

6.1. Local and remote connections

6.1.1. Local connections

In the section 5 it was explained and demonstrated how a remote connection GUI looks like and how it is used. Local connections work similarly to remote connections as seen from the GUI with the exception that the IP scan can be performed right there in the same network where the client's machine is and no preliminary connection to a gateway is required. Therefore the local network table looks exactly like the device table under the remote networks table (see Figure 6.1).

LAN Devices							
No	MAC	IP	Port	Username	Password	Public key	
1.	01:23:45:67:89:ab	192.168.0.1	22	tim	tim123	/home/..	**Connect**
2.	01:23:45:67:89:cd	192.168.0.2	22	roy	roy123	/home/..	**Connect**

Figure 6.1 Local area network devices

As the Qt Creator supports local SSH connections and has the appropriate API in QSsh library, making local connections work from the GUI is as simple as connecting the "Connect" button with the correct class method and passing in the necessary information from the remote network table.

6.1.2. Remote connections

Although a session could be created to a remote machine the same way as a local connection, this method could not be used for devices in remote private networks. Reason being that this

way the created session can not be easily managed for tunneling without making huge changes in the QSsh library.

This is why the remote SSH session creation has to be done and managed in the remote plugin. Luckily the API of QSsh is rather high level and the remote plugin can depend on it and use it easily without too much hustle.

When user clicks on a “Connect” button in the remote networks table, a method *connect()* is called in the remote plugin and a data structure with necessary information from the table is passed in as a parameter. This data structure consists the following:

- Hostname
- Port number
- Username
- Password
- Public key

The subnet is not needed at this point for creating the initial SSH session to the remote gateway machine.

Now a problem arises – CDP Studio can not be dependent of the remote plugin and thus can not directly call the *connect()* method in the remote plugin. This is where the dependency inversion principle comes in handy.

6.2. Inverting the dependency for remote connections

The dependency inversion principle as the name suggests helps switch the direction of the dependency in the code architecture. If the GUI implementation were to call the *connect()* method directly on the remote plugin the dependency would be as shown in Figure 6.2. Everything would work fine in the case where the remote plugin was installed. However if it was not installed, the CDP Studio would be unable to resolve this dependency.

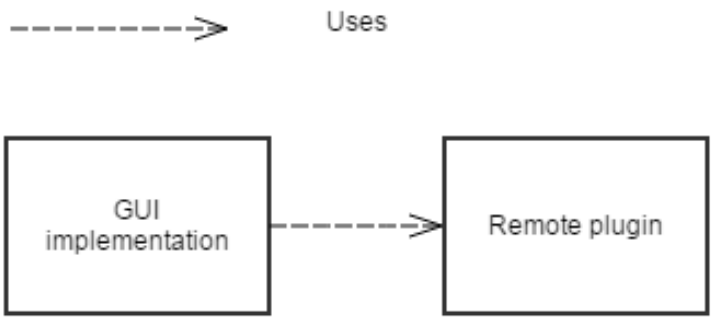


Figure 6.2 Unwanted dependency

In Figure 6.2 the boxes represent components that have the following purpose:

- GUI implementation – a part CDP Studio plugin that among other things manages GUI for remote connections. This component needs the remote plugin to be able to create remote connections that can be used for tunneling
- Remote plugin – this plugin provides the API for remote connection creation for further tunneling

To fix this issue the dependency inversion principle suggests that a layer of abstraction has to be created between the two components. This is demonstrated on Figure 6.3.

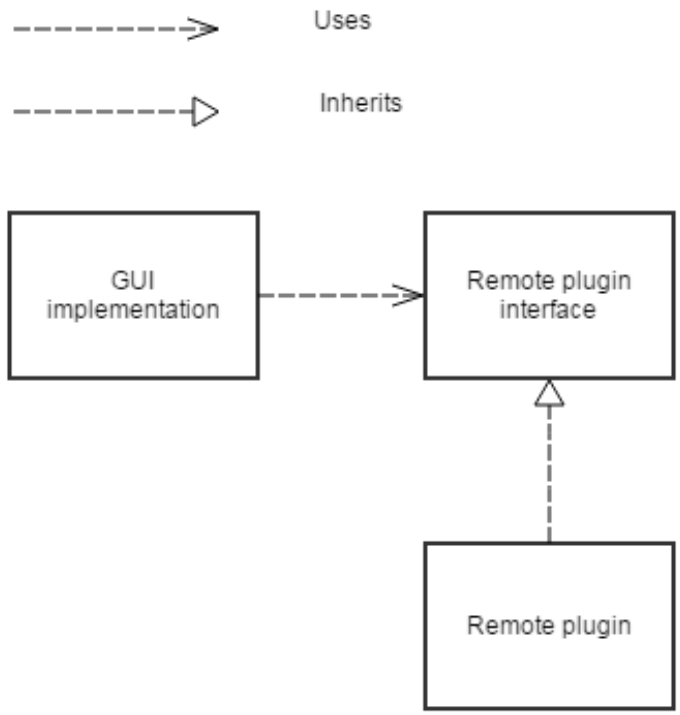


Figure 6.3 Dependency inversion

In Figure 6.3 the Remote plugin interface is a purely virtual class that both the GUI implementation and remote plugin components depend upon. In order to have the CDP Studio be independent from the remote plugin the interface class has to be places inside the same component that its user (GUI implementation) is in.

Now the GUI implementer depends on the interface instead of the remote plugin. The interface class has the required method *connect()* that the implementer needs to call. However it is only an interface and it does not provide an implementation for this method. This is the job of the remote plugin. The remote plugin now has to have a class inside of it that inherits from this interface and creates the actual SSH connection.

This results in an inverted dependency because now both remote plugin and GUI implementation depend on the interface but the interface is inside the core CDP code and not the optional remote plugin.

6.3. Implementation of IRemoteConnectionManager

When the GUI implementer tries to call *connect()* on the interface, the interface obviously has to have an implementation, because the interface by definition does not have one. This means that the implementation for the interface has to be given to the class that uses the interface.

If the remote plugin exists, it could be done a couple of different ways:

- They GUI implementer has a public method for setting the implementation that the remote plugin calls
- The GUI implementer might have a static public static member for the implementation that is set by the remote plugin
- The GUI implementer uses a global variable that is set by the remote plugin
- There might be a global vector of implementations that the GUI implementer chooses one from

In this case there is actually only one implementation that this interface can have, because the remote plugin is the only thing that provides an implementation. This means that a vector of implementations is not necessary. Public members and global variables are not a very safe way of doing things in general so the public method (setter) is used.

So the remote plugin must create an instance of the implementation class and pass a pointer of it to the GUI implementer using this setter method.

Now what if the remote plugin does not exist? In that case there is no implementation to give to the GUI implementer. But since the remote connections are not even allowed if no remote plugin is installed, there is no point in even showing the remote networks table to the user and therefore no implementation is needed.

6.4. Code architecture

In this section the actual design of session creation is described taking into account all the previously discussed considerations.

Figure 6.4 shows the code architecture solution for how remote connections are created, managed and how the dependency inversion discussed in section 28 is implemented. Each box in the figure represents a C++ class.

System configuration plugin is the plugin where the GUI for remote connection handling is implemented. Yellow boxes represent the classes in the system configuration plugin that are added for the SSH connection functionality implementation (that means excluding the classes that implement the GUI).

Blue boxes represent the classes in the remote plugin that are used for creating and handling remote connections.

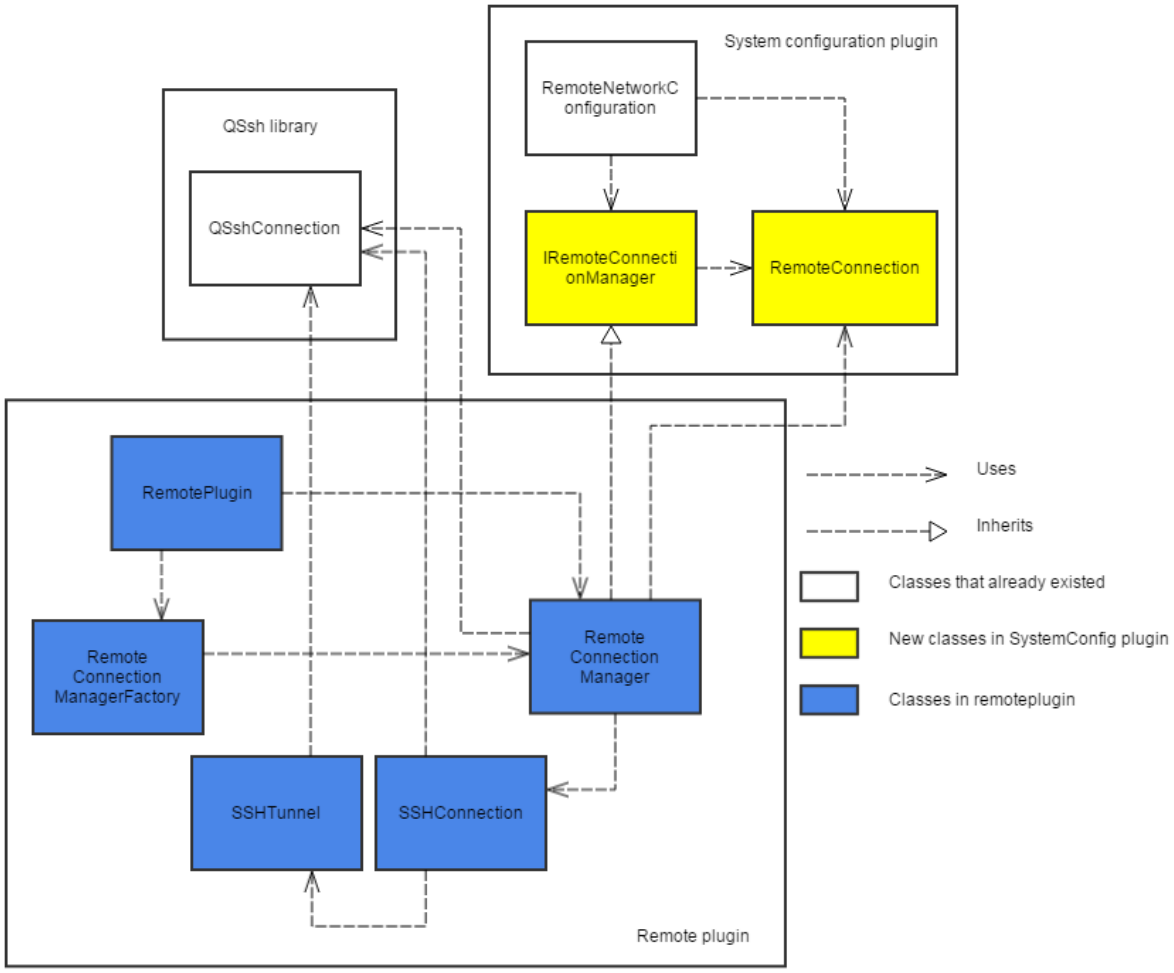


Figure 6.4 Remote connection creation

Following is the explanation of each class shown in Figure 6.4.

- QSshConnection – this is the class inside the QSsh library that provides an API for creating SSH sessions and SSH tunnels
- IRemoteConnectionManager – this is an pure virtual interface class (as indicated by the capital letter 'I' in front of the class' name) that provides methods for connecting to and disconnecting from gateway machines
- RemoteConnection – this class' instances are only used as a handle to the currently running primary SSH session
- RemoteNetworkConfiguration – this class handles the "Connect" calls from the GUI and uses the IRemoteConnectionManager interface to connect to SSH servers or disconnect from them. It also uses RemoteConnection class to keep track of the current session if there is one
- RemotePlugin – this class is used to set up and tear down the remote plugin. It is responsible for variable initialization and other similar matters. It also uses the RemoteConnectionManagerFactory class to create an instance of RemoteConnectionManager and gives that instance to the RemoteNetworkConfiguration class as the implementation for IRemoteConnectionManager via a setter method as described in 6.3
- RemoteConnectionManager – this class provides an implementation for IRemoteConnectionManager. It creates instances of SSHConnection class when its method *connect()* is called and deletes them when *disconnect()* is called.
- RemoteConnectionManagerFactory – this class is used to keep a handle of an instance of the RemoteConnectionManager class. It provides access to that instance via a static variable
- SSHConnection – this class wraps the necessary methods from QSshConnection for setting up a primary SSH session and SSH tunnels. It also keeps handles of all running SSH tunnels in a vector
- SSHTunnel – this class is used to set up the functionality of SSH tunnels and configure the data transfer through them

7. SSH TUNNELING

When an SSH session to a remote gateway has been established, an IP scan is performed on the subnet determined by the subnet mask in the remote networks' table to populate the devices' table with all the available devices in this newly connected network. In this section the creation of an SSH tunnel is explained and the initiation of the second SSH session is described. At this point it is assumed that the IP scan is already done and the devices' table is populated (as seen on Figure 5.2).

7.1. SSH tunnel

Tunneling is basically setting up a TCP connection from one machine to another but through an intermediary gateway machine (see Figure 7.1).

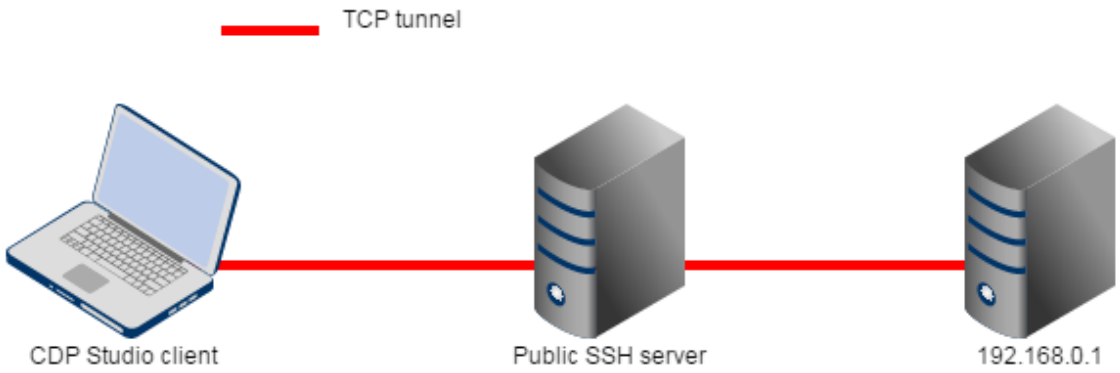


Figure 7.1 TCP tunnel

For this tunnel to work and the remote machine (the device with IP address 192.168.0.1 in this example) to be accessible, its originally private IP and port pair has to be mapped to an accessible IP address and port pair for the CDP Studio client machine. In this thesis the port that is of interest, is the SSH port, which for this example is assumed to be the default 22. So 192.168.0.1:22 has to be mapped to an IP address and port pair in the CDP Studio client machine. This is called network address translation (NAT). Usually localhost (127.0.0.1) is used as the host address and the port is chosen either manually or some tool is used to get a free port. In Figure 7.2 an example is shown of this address translation, where the host-port pair 192.168.0.1:22 is mapped to a new address-port pair 127.0.0.1:5555 on the CDP Studio client machine.

Figure 7.2 Network address translation

This works exactly like in a local network, except in this case the remote network is made available with the first SSH session that is created from CDP Studio client machine to the gateway server. As this TCP tunnel passes through that initial SSH session, it is called an SSH tunnel

7.2. Creating an SSH tunnel

The user does not need to know anything about the SSH tunnel, how, when or where it is created to. This means that the tunnel is created when the user clicks the “Connect” button in the device table and immediately before the second SSH session is created.

To minimize the work that has to be done, it is most optimal to use the Qt Creator’s local connection creation methodology to also connect to devices in remote networks. This has to be made possible with the SSH tunnel and network address translation that was described in the previous section.

Easiest way to achieve this goal is to modify the QSsh library code, so that when a session creation is initiated, the host and port pair is passed to the remote plugin where a tunnel is created and the translated host and port pair is returned and the session is then created to this newly translated destination. It is essential however to keep any modifications to the QSsh library minimal as it is a Qt Creator’s library. Although every change made there by ICD can be turned into a patch and applied onto a newer version of Qt Creator, there still remains a small chance of merge conflicts. This means that if the Qt Creator’s developers change something in the QSsh library in the same place as a patch by ICD the process of applying said patch might fail.

A couple of problems arise with this solution at first glance. The first one is similar to the issue that was described in section 6.1 – the problem of dependency. The second problem is

that the tunnel and translation has to be made only if there is a connection created to a remote gateway server.

The second problem is a simple matter of writing the translation algorithm in the remote plugin so that it checks for the existence of an SSH connection in the remote plugin (meaning a connection created in the remote plugin as explained in section 6, not just a local connection). If an SSH session is available then it is necessary to create a tunnel.

```
translate(initialHostAndPort)
{
    translatedHostAndPort = initialHostAndPort;
    if(hasSshSession)
    {
        translatedHostAndPort = createTunnel(initialHostAndPort);
    }
    return translatedHostAndPort;
}
```

Figure 7.3 Translation algorithm

Note. All the code presented in this thesis is conceptual pseudocode and not the actual code used in CDP Studio.

The basics of this algorithm are shown in Figure 7.3 in the form of a pseudocode. In this case the method *createTunnel()* takes in the devices local IP and SSH port as a parameter and creates the tunnel to this destination, while translating the IP to a local IP and port pair and returning it.

This algorithm does however not create a tunnel nor translate the IP and port if no preliminary SSH session exists. In that case the initial IP and port pair is returned.

```
createSshSession(hostAndPort)
{
    .
    .
    .
    //some set up for creating an SSH session
    .
    .
    .
    connectToHost(hostAndPort);
}
```

Figure 7.4 SSH session creation in QSsh library initially

In Figure 7.4 the SSH session creation code in QSsh libraries class QSshConnection is shown as it works for local connections. This has to be modified to also work with remote connections. This is the place where the existing QSsh library code has to be changed. Luckily with the chosen design the change is very minor and does not actually affect any other functionality inside the library.

```
createSshSession(hostAndPort)
{
    .
    .
    .
    //some set up for creating an SSH session
    .
    .
    .
    translatedHostAndPort = remotePlugin.translate(hostAndPort);
    connectToHost(translatedHostAndPort);
}
```

Figure 7.5 SSH session creation after modifications

The Figure 7.5 shows the usage of the *translate()* method to either translate the initially given IP and port to that of the remote device or not translate if the user is trying to connect to a local machine (or to a machine with a public IP address). The changes are highlighted with green.

Now that this *translate()* method is added to this part of the QSsh library code, every SSH connection that is initiated applies translation if necessary, provided that the remote plugin is installed. If the connection is directed to a device in a remote network, a tunnel is also created as seen in Figure 7.3.

This solution would be final if the remote plugin was a default part of the CDP Studio. However as it is optional, the solution has to be improved to remove the dependency of the remote plugin and maintain previously available functionality even if it is not installed.

7.3. Default translation implementation

Since this *createSshSession()* that is shown in Figure 7.5 method is used for local connections as well and has to work even if the remote plugin is not installed, there has to be an alternative implementation available for the *translate()* method. The QSsh library can not be dependent of the remote plugin. To be able to provide different implementations to this method, a level of abstraction has to be added. This will be achieved similarly as in section 6.1.

The main principle is again that an interface has to be created and the remote plugin has to inherit from it and provide an implementation to it that does the translating and tunneling. The interface again goes where its user is. In the given situation this means the interface has to be inside the QSsh library, next QSshConnection class.

As mentioned an alternative implementation has to be available as well for when the remote plugin is not installed. This implementation shall also be inside the QSsh library because there it is always accessible for the QSshConnection class.

This default implementation actually does nothing more than return the same IP and port pair that was passed into it (see Figure 7.6) because it is wanted that only CDP Studio clients with the remote plugin are able to connect to remote devices.

```
translate(initialHostAndPort)
{
    return initialHostAndPort;
}
```

Figure 7.6 Default translate() implementation

The Figure 7.7 shows the code architecture for how the interface is used and how the implementers are located. This structure also provides the dependency inversion just like in section 6.1.

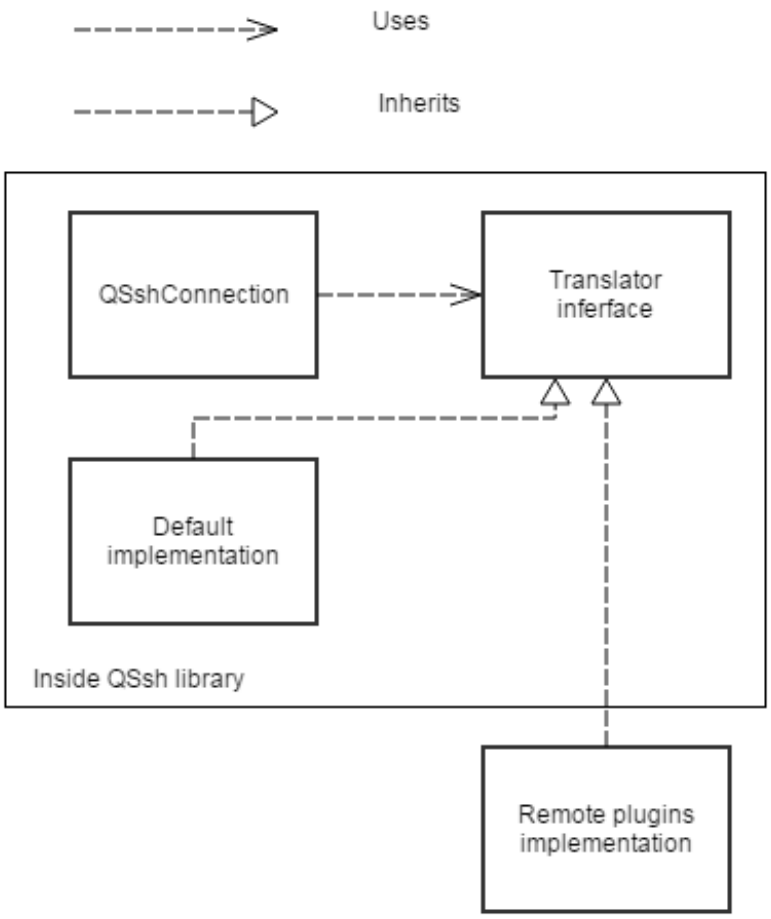


Figure 7.7 Translator implementations

In Figure 7.7 the boxes represent conceptual C++ classes that have to following purposes:

- QSshConnection– a class inside the QSsh library that sets up and configures SSH sessions. This class uses the translation method from the Translator interface class
- Translator interface – a class inside the QSsh library that is added as a part of this project and is used by the QSsh library Session Creator class as an interface for implementations that QSshConnection class does not need to have any knowledge of
- Default implementation – this class provides an implementation for the Translator interface that is used in case the remote plugin is not installed

- Remote plugins implementation – this class provides an implementation for the Translator interface that is used if the remote plugin is installed

Now it is clear what both the default and remote plugin's implementation do. Next problem is how to give QSsh library the correct implementation for the translator interface.

7.4. Providing implementation for translator interface

In this section it is explained how an implementation is provided for the interface that QSshConnection class uses for executing translation of the network addresses and for tunneling.

One thing to note before going into the chosen solution here, is that according to CDP Studio requirements, Studio without the remote plugin must allow users to have only one concurrent SSH session running. However the remote plugin has to enable them to open several SSH tunnels to the same network at the same time and create SSH sessions through those tunnels. The task described in this thesis does not involve concurrent SSH connections to different remote networks, but since this is an issue for future development, the code architecture has to be built up so that it supports creating several network address translator instances. In that case each translator would handle the translations of each remote network.

To create the initial SSH session (the session from CDP clients machine to the remote gateway server) the client clicks "Connect" in the remote network table and if there are no SSH sessions running already then the remote plugin has to create a new session. A pointer to this session is saved in the RemoteConnectionManager class for future usage. As for the creation of the SSH session the QSshConnection class is used, the translation method already needs to have an implementation. To provide a possibility for future multi-connection redesign, a factory class has to be introduced. The QSshConnection class can use the factory class to create an instance of a translator to perform the translation each time a new SSH connection is initiated. This way when the multi-connection functionality is added each connection to a remote gateway can create an instance of a network address translator to use.

A factory class is a class that has a method for getting an instance of another class. That instance is either created when that method is called or an already existing instance is

returned. In this case the factory will be used to create instances of network address translators.

Because the factory for the remote plugin's translator implementation has to be inside the remote plugin as well, there needs to be another layer of abstraction between the QSsh library and the remote plugin. The default implementation of the translator also needs to have a factory class. Therefore an abstract factory class is created that serves as an interface between the QSshConnection class and the two factories. This abstract class is called AbstractTranslatorFactory.

The AbstractTranslatorFactory class will have a static member that by default holds a pointer to an instance of the default translator and it is accessible via a method called *instance()*. If however the remote plugin is installed then this method has to return an instance of the factory provided by the remote plugin. For this the RemotePlugin class has to overwrite the static member in the AbstractTranslatorFactory during the initialization.

Again this layer of abstraction is added in order to prevent the QSsh library from depending on the remote plugin.

Let us take a look at the design of this in Figure 7.8.

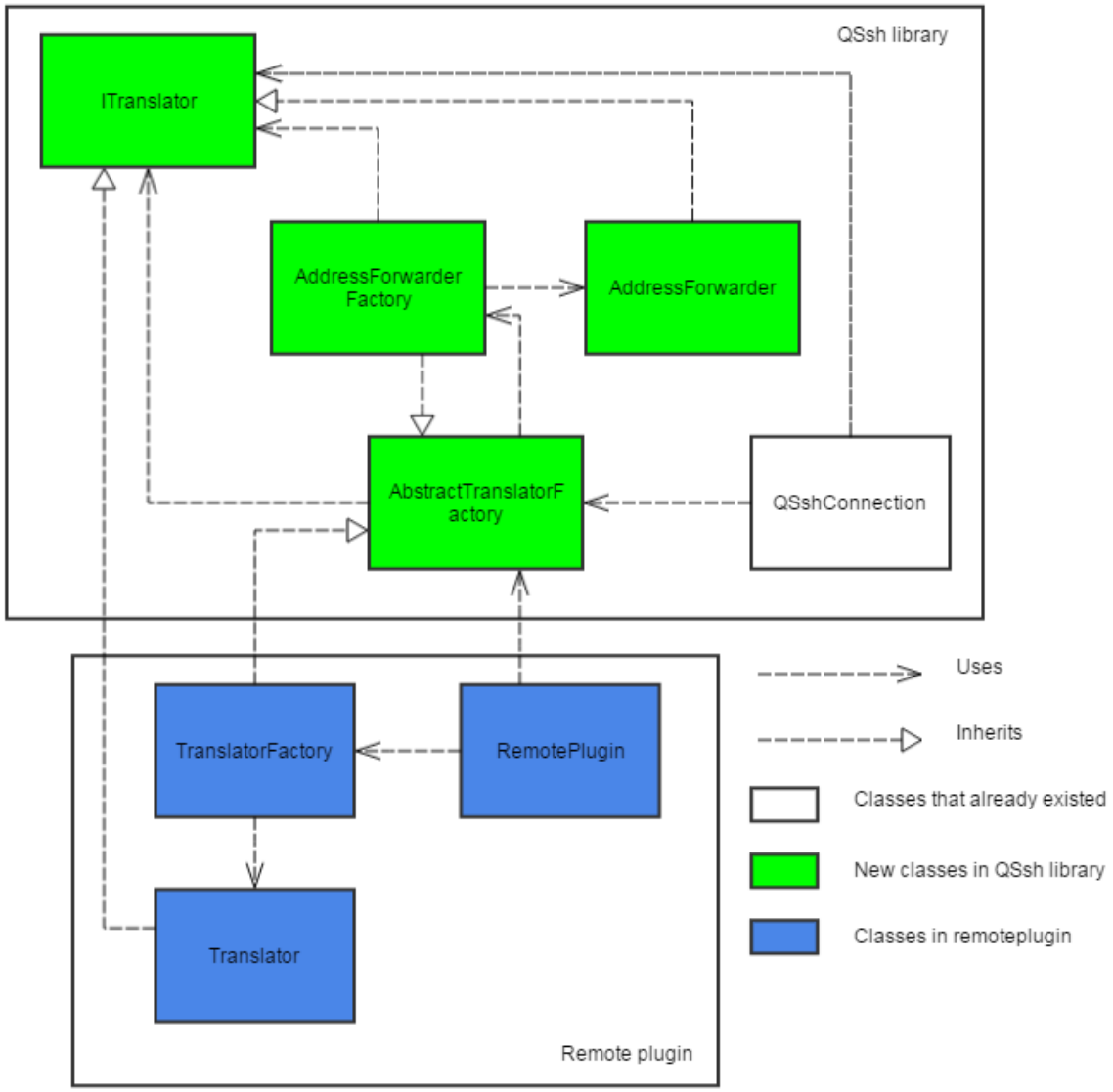


Figure 7.8 Translator factories

In this figure the each box represents a C++ class. A prefix ‘Abstract’ in front of a class’ name refers to the fact that the class is an abstract class in the context of C++ programming language. A letter ‘I’ on the other hand refers to the fact of it being a pure virtual interface class.

The green boxes represent classes that were added into the QSsh library for the purpose of this task. These are new classes that did not exist prior to adding the functionality for SSH tunneling. They are only needed for this task and thus do not in any way modify other functionality of the QSsh library beyond what is described in this thesis.

The blue boxes represent classes that are in the remote plugin and are needed for network address translation. The remote plugin contains other classes for tunneling and setting up SSH sessions to remote networks as described in section 6.1 but they are not relevant in the context of this section and are therefore not shown.

Following is the explanation of each class shown in Figure 7.8:

- QSshConnection – this is the class inside QSsh library that is used for creating SSH sessions. This class is also the only existing class inside the QSsh library that is modified for this task. The modification is briefly explained in section 7.2 and demonstrated in Figure 7.5
- ITranslator – this is a pure virtual class that is used as an interface for network address translators
- AddressForwarder – this is a class that inherits from ITranslator class and provides the default implementation for address translation as explained in section 7.3. As this implementation does not actually change the IP or port, but only forwards them, the class is named AddressForwarder
- Translator – this is a class that inherits from ITranslator class and provides an implementation that does network address translation and SSH tunneling
- AbstractTranslatorFactory – this is an abstract class that is used by the QSshConnection class to get an instance of a Translator. It is also a layer of abstraction between the QSsh library and the remote plugin to provide a factory for either the AddressForwarder or the Translator class
- AddressForwarderFactory – this class inherits from AbstractForwarderFactory and provides an implementation to its otherwise virtual method *createTranslator()* which is used by the QSshConnection class to get an instance of an ITranslator type class. This implementation of the AbstractForwarderFactory is used to create the default implementation of the ITranslator – AddressForwarder
- TranslatorFactory – this is the alternative for AddressForwarderFactory and implements the *createTranslator()* method so that it returns an instance of the Translator class
- RemotePlugin – this class handles the initialization and destruction of the remote plugin. It is also used to pass an instance of TranslatorFactory to

AddressForwarderFactory that keeps it as a static variable so that it can be used by QSshConnection

7.5. Network address translation implementations

In this section it is described in more detail how QSshConnection is provided with an instance of a ITranslator type class.

In the case when the remote plugin is not installed, only the classes shown in Figure 7.9 are present. In that case AddressForwarder is the implementation of ITranslator that QSshConnection has to use for address translation. For that AbstractTranslatorFactory creates an instance of AddressForwarderFactory and stores it in a static variable of the type AbstractTranslatorFactory. This instance is taken by QSshConnection with the use of AbstractTranslatorFactory's static method *instance()*.

A thing that seems likely to cause a problem with the design shown in Figure 7.9 is the cyclic dependency between AddressForwarderFactory and AbstractTranslatorFactory. However it is not an issue because forward declarations are used properly and the header of AbstractTranslatorFactory does not actually include the header of AddressForwarderFactory.

In the case when the remote plugin is installed, the same thing happens inside QSsh library – during program start up the static member in AbstractTranslatorFactory is initiated with an instance of AddressForwarderFactory. However as the remote plugin is initialized, the RemotePlugin class (see Figure 7.8) overwrites this member with an instance of TranslatorFactory with the use of AbstractTranslatorFactory's static method *setInstance()*. Whenever the QSshConnection calls the AbstractTranslatorFactory's method *instance(I)*, it now returns the instance of TranslatorFactory which can be used to create Translators that enable the user to do SSH tunneling.

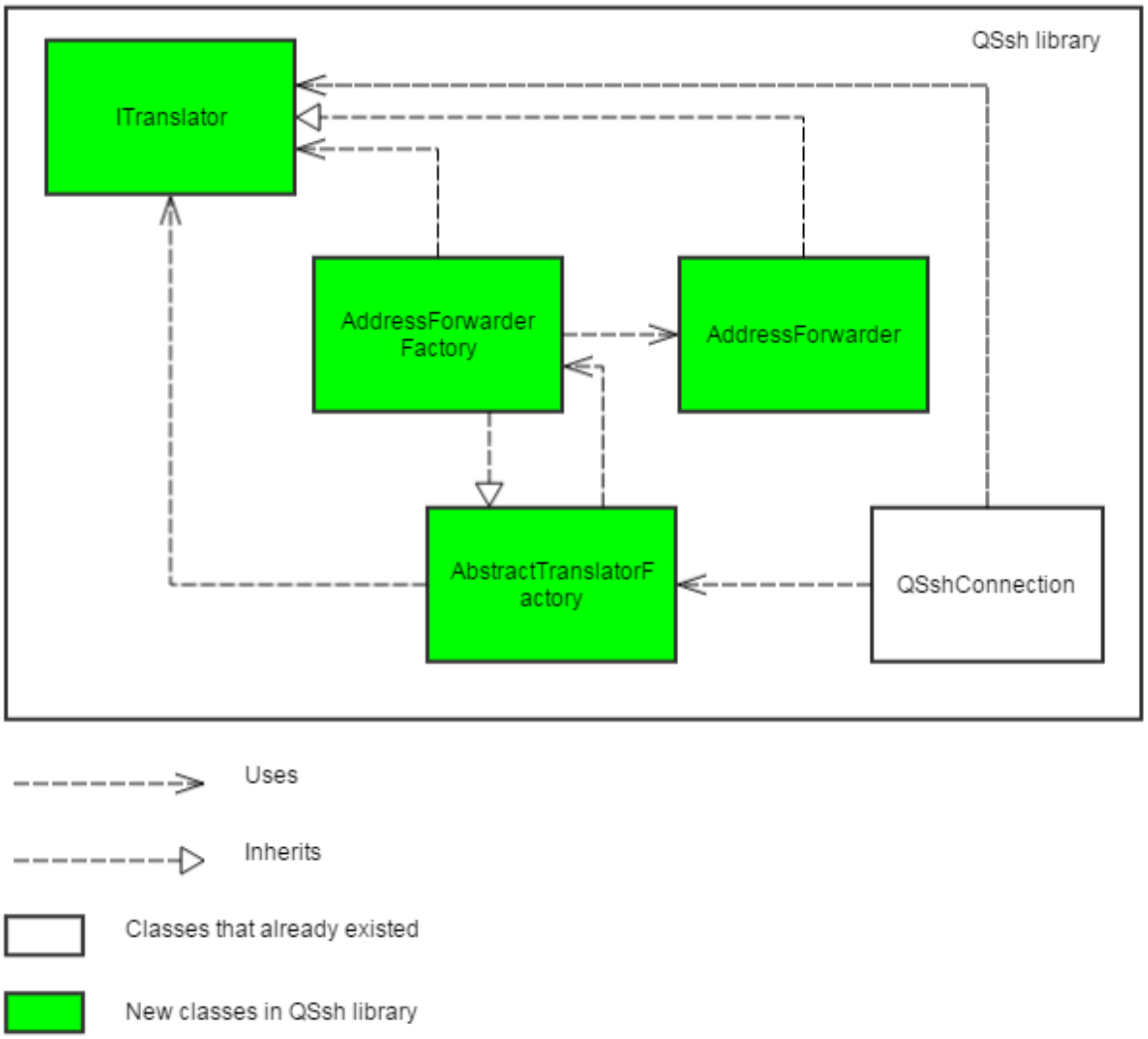


Figure 7.9 Added classes in QSsh library

8. FINAL SOLUTION

In section 6 and section 7 two main parts of the remote plugin were introduced and their working principles presented. In this section it is shown and explained how the remote plugin works as a whole. A final design is demonstrated and the full working principle is explained.

8.1. Full design

Connecting to a remote gateway server and setting up a SSH session for further tunneling is explained in section 6 and the code architecture made for this is shown in Figure 6.4. This is a prerequisite for the SSH tunneling which is described in section 7 and the code architecture of which is shown in Figure 7.8. Of course the two parts are actually inept without one another so in practice they were designed together. In Figure 8.1 the integration of the two aforementioned parts is shown. There are three components that are affected by this design:

- QSsh library
- System configuration plugin
- Remote plugin

The QSsh library and system configuration plugin existed before the beginning of the task described in this thesis. The remote plugin however is entirely a result of the current project.

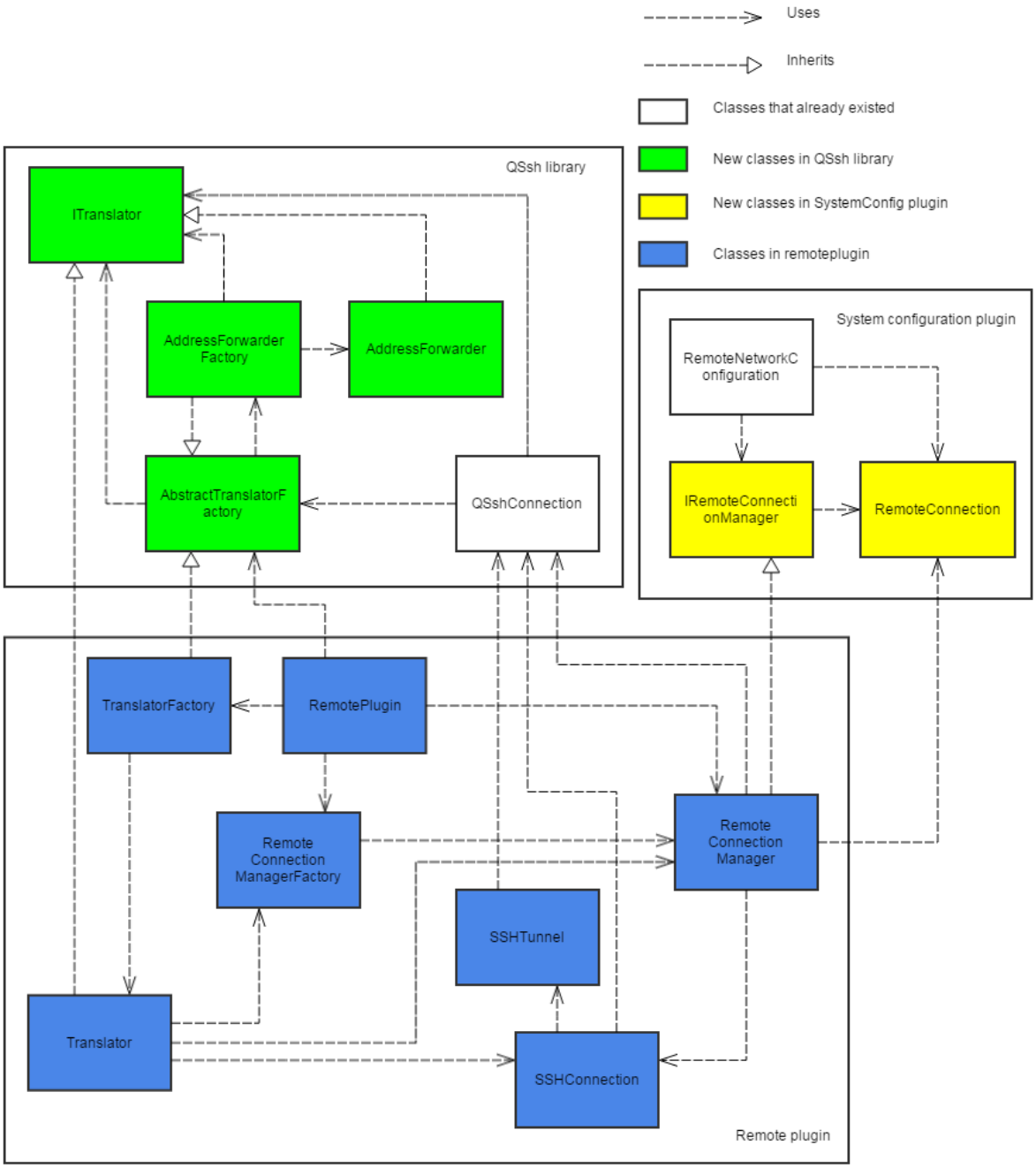


Figure 8.1 Final code architecture design

8.1.1. Total independence

In section 1.4 it was stated that the remote plugin must be completely optional for CDP Studio and that any of the core code of the Studio can not depend on the plugin. As seen on Figure 8.1 all of the dependency arrows that cross the border of the remote plugin are pointing away from it. This means that no class outside of the remote plugin depends on any class inside of

it. This coupled with the fact that all functionality of the CDP Studio prior to adding the remote plugin is intact means that the goal of independence is achieved.

The basis for achieving the independence is again the dependency inversion principle as explained in section 6.2.

8.2. Working principle without remote plugin installed

In this section it is explained how local connection creation works with all the added code but without the remote plugin.

8.2.1. Initialization

In QSsh library during CDP Studio start up AbstractTranslatorFactory, the class that is used by QSshConnection as an interface for a translator factory, initializes its static member *s_instance* to be an instance of AddressForwarderFactory. This means that whenever QSshConnection needs to create an instance of a translator, it will do so by using AddressForwarderFactory because this is the implementation of AbstractTranslatorFactory that is returned when it calls *AbstractTranslatorFactory::instance()*.

In system configuration plugin during start up a static pointer variable that is of type IRemoteConnectionManager is initialized to be a *nullptr*. This means that it is not pointing to any instance of an object.

In Figure 8.2 the relevant initialization steps are shown when the CDP Studio is started up without the remote plugin installed.

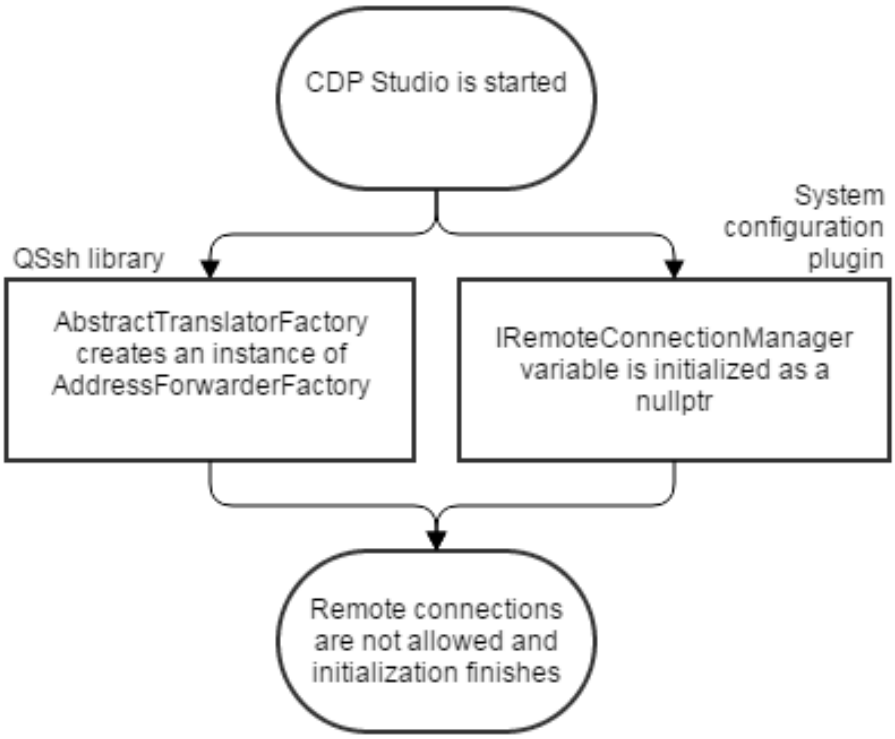


Figure 8.2 CDP Studio initialization without remote plugin

8.2.2. Connecting to local machine or remote network

After initialization if a user enters the system configuration page, where network connections are handled, only GUI for local connections is shown, because as the IRemoteConnectionManager variable is not pointing to any implementation (is *nullptr*), it is assumed that the remote plugin is not available thus remote connections can not be made.

When a user tries to create a local connection however, Qt Creator's default session creation API is used. The back end of this API uses QSshConnection class for actually setting up a session so that is where the changes made in QSsh library come to play. The QSshConnection gets the instance of AbstractTranslatorFactory to create an instance of AddressForwarder. It uses that class instance to perform the translation, which in reality does nothing but return the same address and port pair. Then the SSH session is created.

8.2.3. SSH Tunneling

Without the remote plugin it is not possible for users to set up sessions to devices in remote networks, so CDP Studio does not allow SSH tunneling.

8.3. Working principle with remote plugin installed

In this section the whole work of the remote plugin is explained from initializing the plugin to creating SSH sessions to remote gateway servers and SSH tunnels to remote network machines.

8.3.1. Initialization

During CDP Studio startup the core of the Studio is initialized first. That involves all Qt Creator code (more specifically QSsh library) and all essential plugins. Everything that is described in section 8.2.1 is performed during this. When it is done, the remote plugin is initialized.

In QSsh library the static variable in AbstractTranslatorFactory that currently points to an instance of AddressForwarderFactory is replaced with an instance of TranslatorFactory in the initialization method of RemotePlugin class. This means that now whenever QSshConnection needs to create an instance of a translator, it will do so by using TranslatorFactory instead of the AddressForwarderFactory.

In system configuration plugin the static pointer variable of type IRemoteConnectionManager that is currently a *nullptr* is replaced with an instance of RemoteConnectionManager.

In Figure 8.3 the relevant initialization steps are shown when the CDP Studio is started with remote plugin installed. The important difference with Figure 8.2 is that the implementations that are registered by default are overwritten by the remote plugin.

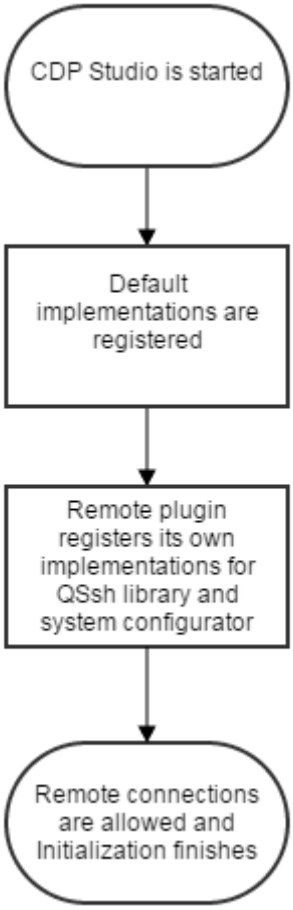


Figure 8.3 CDP Studio initialization with remote plugin

8.3.2. Connecting to local machine or remote network

Creating local SSH sessions is still done by using the Qt Creator's API. However inside QSshConnection when the *translate()* method is called an instance of the Translator class is used. Since local connection creation does not require any SSH tunneling or address translation, it basically still works as the AddressForwarder (see Figure 7.3).

As the remote plugin is installed and the IRemoteConnectionManager variable is not a *nullptr*, the GUI for remote connections is displayed to the user (see section 5). When the user clicks the "Connect" button for remote connection creation, instead of the Qt Creator's default API the RemoteConnectionManager is used for setting up an SSH connection to a machine. This session is handled inside the remote plugin as an instance of a class SSHConnection and kept track of in order for SSH tunnels that might be created later on.

If a remote SSH session is running when a user tries to create a new one, it is then checked if the destination is the same as the previous one or not. If the destination is the same, the "Connect" button click is ignored. However if the user is trying to connect to a different destination, all the SSH tunnels using the current session are closed, the session is disconnected and the old instance of SSHConnection is deleted. Then a new session is created and a new instance of SSHConnection is created for further reference.

The remote plugin also uses QSshConnection class for setting up the session so again the issue of translation arises. When the *translate()* is called, it is checked if a remote connection already exists. If it does not then no translation is done and the Translator works as AddressForwarder. This is required to be able to create SSH connections to remote gateway servers.

The logic behind setting up the primary connection to the remote network (gateway server) is demonstrated on Figure 8.4.

When a user connects to a remote gateway machine, the GUI in system configuration page changes (see Figure 5.2) to display all the available devices in the remote network.

At this point it is possible to create SSH sessions to those devices in the remote network. When the user clicks the "Connect" button of a remote device, the Qt Creator's default API is used to create this SSH session exactly as if it was a local device. It is so because that way users of that SSH Session do not need to know that it is a session created through an SSH tunnel and the session can be used exactly as if it was a local connection.

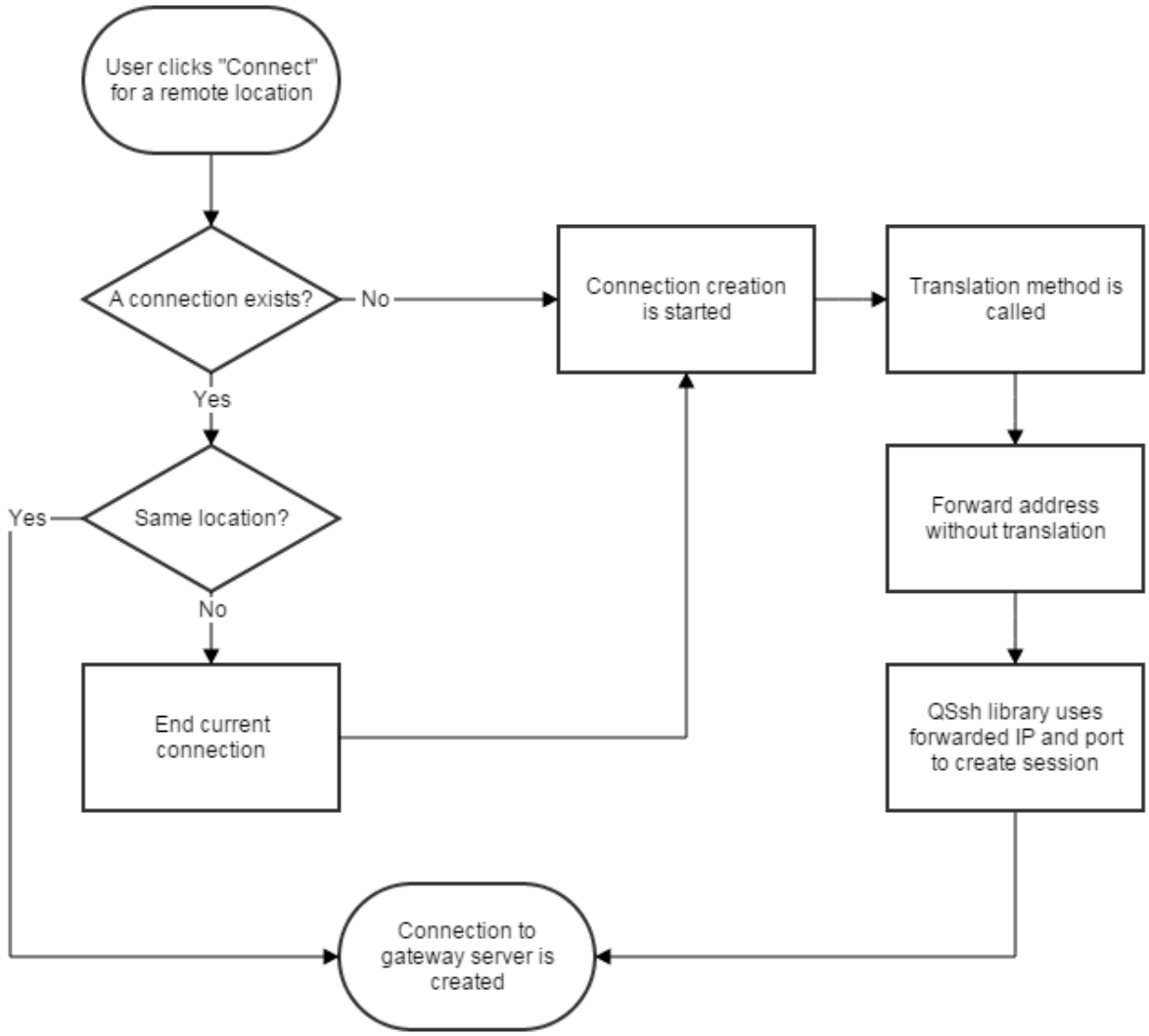


Figure 8.4 Connecting to gateway server

8.3.3. SSH Tunneling

In QSshConnection the *translate()* method now does its job. As the method is called and because a remote SSH Session already exists, it assumes that the session that is currently being created needs to go through an SSH tunnel. It gets a reference of the remote SSH session (the session that is between the client and the remote gateway) that was created earlier (the instance of SSHConnection) from RemoteConnectionManager. The Translator uses that SSHConnection instance to create an SSH tunnel through it. The SSHConnection class uses the SSHTunnel class to set up this tunnel and configure the data transfer through it. For this a TCP server is created on *localhost* and an available port is taken for address translation. The tunnel is forwarded to the *localhost* and the port. This new address and port pair propagates

back through SSHConnection and Translator to QSshConnection. At this point the connection is made to this new address as if it was a local machine.

In Figure 8.5 the main steps when creating an SSH tunnel are shown.

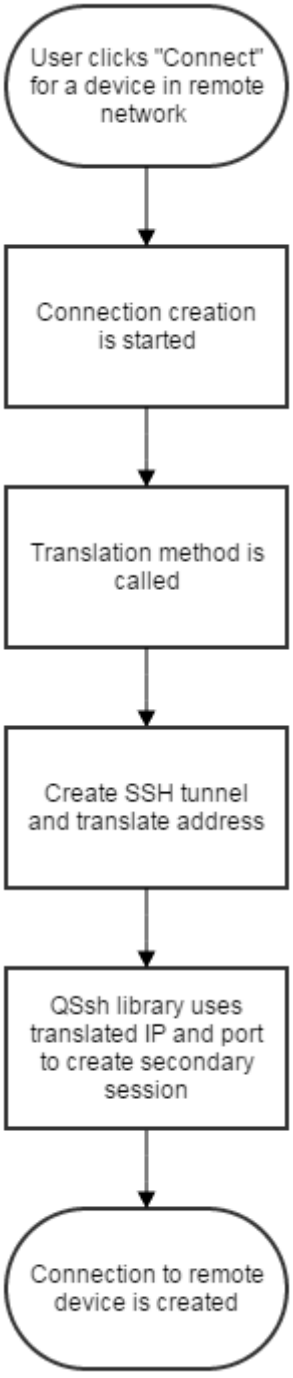


Figure 8.5 SSH Tunnel creation

9. FUTURE IMPROVEMENTS AND USES

In this section the main shortcomings of the chosen design and the final solution are pointed out. Some potential improvements are described and some other uses for the plugin are discussed. Everything discussed in this section is outside the scope of this thesis

9.1. Flaws and shortcomings in current design

9.1.1. Modifications in Qt Creator's code

An issue that has already been described earlier in this paper but what could possibly become troublesome is the fact that some Qt Creator's code was modified during the process of creating this solution. Though not very likely, this might cause problems in the future, when ICD decides to switch over to a newer version of Qt Creator. If something goes wrong when applying these changes onto a newer version of Qt Creator then the code merge has to be done manually and that requires some extra work.

9.1.2. One remote network at a time

With the current design users are only able to connect to a single remote network at a time. Although not very common, the users might want to have running SSH sessions to two or more different remote networks. For example if two networks have CDP applications that communicate with each other via the internet and the user wants to debug or monitor both of them at the same time. As it stands right now, the user has to switch back and forth between the two remote networks.

9.2. Future developments

9.2.1. Several remote networks at a time

As explained in section 9.1.2 it would be preferable if users could connect to several remote networks simultaneously and be able to create SSH tunnels to the devices in those networks. With the current design it is not possible and it would require some extra work to make it function this way.

It would actually be possible to set up several SSH sessions to remote gateway servers with some minor changes in the code architecture because the code architecture was built with this in mind. However when it comes to SSH tunneling it gets tricky. The problem is that if two or more remote networks have the same subnet mask, then since the QSshConnection class has no idea which one of those the user is trying to create a tunnel to, it is impossible to make it work properly with the current design.

This is something that is planned for the development of the CDP Studio.

9.2.2. Other uses

As mentioned in section 1.4 this thesis does not cover other uses for the SSH tunneling functionality besides for CDP application deployment which is done through the QSsh library. A future improvement that is planned for this project is to make it possible to use the remote plugin to be able to use SSH tunnels for other purposes like database connections.

For this some of the code that was put inside the QSsh library has to be extracted to a separate library (called translator library) that other services can also use as an interface for SSH tunneling. The code that has to be extracted is the code that was initially put inside the QSsh library since the translation and tunneling part is what must be usable by other components as well.

In Figure 9.1 the translator library is shown as it is extracted from the QSsh library.

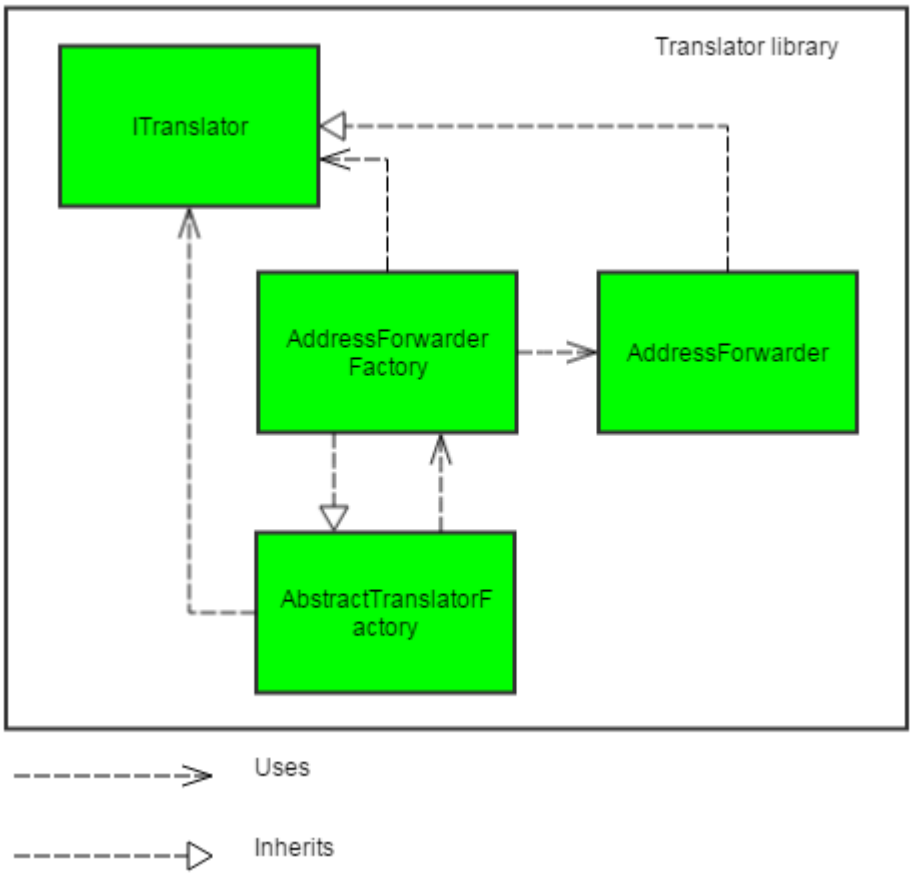


Figure 9.1 Translator library

The remote plugin is still dependent on the classes just as it was before and thus provides an alternative implementation for both the ITranslator and the AbstractTranslatorFactory classes, so if the remote plugin is installed it will be used.

Any service that needs to perform some action in another device via an SSH session can now use this translator library as an interface for creating a tunnel to a remote device and translating its IP and port pair to one that is usable for whatever it is needed for (see Figure 9.2)

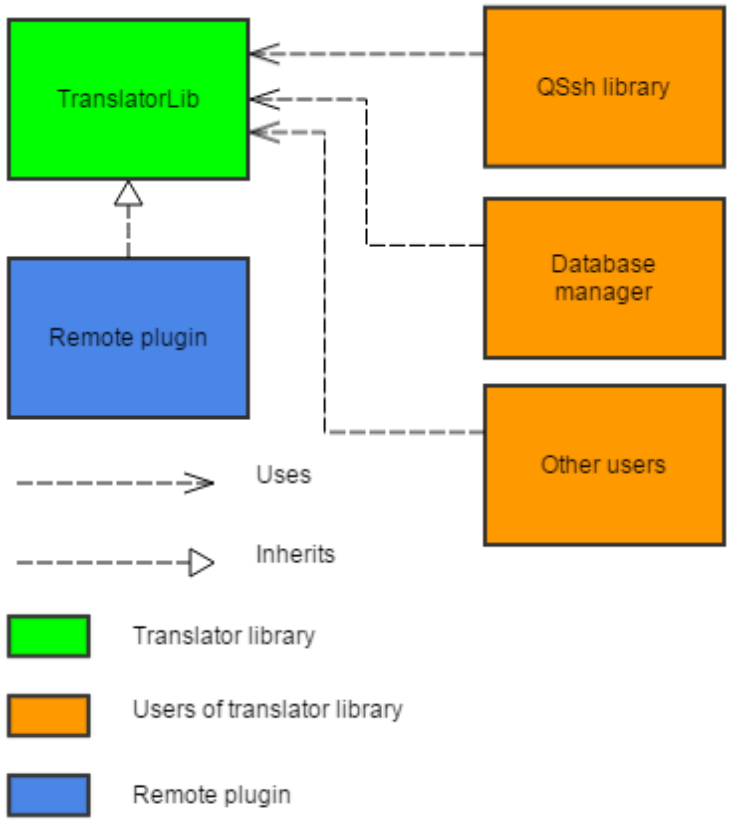


Figure 9.2 Using the translator library

For example if a user wants to create a database connection from his/her machine to a device that is in a private remote network, he/she will first create a secure SSH session to the remote gateway machine just like explained in section 8.3.2. Then the user uses the database manager component exactly like he/she would use it for a local connection. Inside the database manager component the translation method has to be called that creates an SSH tunnel to the remote device and returns an IP and port pair that the database manager will use to create the connection.

The GUI at the system configuration page should then be modified to not only allow users to connect to the remote devices for project deployment but also allow for other types of actions.

For example instead of the “Connect” button of a device there might be a button called “Choose action” which opens a drop-down menu for the user to pick whether to connect for deployment or to set up a database connection or do some other action.

9.2.3. Stand-alone instead of plugin

Currently the remote plugin as the name suggests is a plugin. It is developed for CDP Studio and is rather specific to it. This software could however be turned into a standalone program that could be used by other software packages or even by itself. The program could then be used as a tool for creating SSH tunnels for any purpose with simple API.

If it was a standalone program, it could be marketed more widely and it would have much more uses than just CDP Studio. Turning it into a standalone would require a lot of work writing the whole SSH connection handling with libssh or libssh2 library because it would not be optimal to use Qt for it as it would result in a lot of overhead.

This improvement is theoretical and is not actually planned to be developed.

9.2.4. License

As the remote plugin is planned to be sold separately from CDP Studio, some licensing mechanism has to be put to place so that users could not install or at least not use the remote plugin if they haven't bought it.

This is planned to be developed before the release of the plugin. There are two main ways that software companies implement licensing:

- Users can not install the program without a license
- Users can install the program but do not have full access to its functionality without a license

Although CDP Studio itself is planned to use the second option it is probably more reasonable to use the first one for the remote plugin because there is not that much functionality inside the remote plugin that could be blocked from the user.

10. SUMMARY

10.1. Goals of the work

CDP Studio is a program that is being developed by ICD Industries that is meant for managing CDP (Control Design Platform) based systems. The goal of this thesis was to develop a plugin for CDP Studio that would allow users to create secure connections over the internet to devices behind gateway machines. Currently no such solution exists that could be integrated into CDP Studio, so it had to be specifically created. A major consideration that had to be taken into account was that ICD Industries wishes to sell the remote plugin separately from the CDP Studio. This means that the remote plugin has to be completely optional and could be absent in the program and the program should still provide basic connectivity functionality. Therefore total independence from the plugin had to be achieved.

The CDP Studio has to allow users to create connections to any publically accessible devices even without the remote plugin. The remote plugin's purpose is to extend that functionality to allow users to forward those secure sessions to devices that are hidden behind firewall machines and have those sessions be usable as if they were local connections. This means the services that create those connections should not be aware whether the connection is to a local or a remote destination.

10.2. Solution

10.2.1. Security solution

The Qt Creator software, which the CDP Studio is built upon, has basic functionality for SSH tunneling that allows creating secure data transfer sessions from the client machine to a publically accessible device with a running SSH server. It was obvious that the optimal way was to use as much of that existing code to achieve the given goal. Therefore SSH was chosen as the secure data transfer protocol. Qt Creator has a library called QSsh library that was used for this project for all SSH session and tunneling related functionality.

10.2.2. Graphical user interface

First the graphical user interface was designed based on the requirements given by ICD Industries. The user manages network connections from a system configuration page. The

author designed a GUI for connecting to remote networks and to devices in those networks. Users are expected to know the host address, port and login information of the SSH servers they want to connect to as well as the subnet masks of the local subnets that the devices are on.

The GUI was created to be a table that the user can enter new network destinations into. When connected to a remote network gateway machine that is publically available, a second table is generated that displays the devices available for connection in the given subnet.

The GUI was handled by the system configuration plugin that manages all other configuration related aspects of CDP systems.

10.2.3. Remote plugin solution

The CDP Studio is built upon the Qt Creator as a number of plugins. The remote plugin that had to be separately installable and provide functionality for SSH tunneling was created the same way.

To minimize the work and duplication of code the chosen solution was to modify the QSsh library to perform a network address translation before each SSH session creation. This means that each time a user wants to create an SSH session and the QSsh library is used for that, the IP and port that are given as the destination are passed into a translation method that performs SSH tunneling and address translation if it is needed, or just returns the same IP and port pair if it is not needed. The translation and tunneling is not needed if the client is trying to set up a connection to a local machine or if there is no preliminary SSH connection set up to a gateway machine. If the remote plugin is not installed, the translation and tunneling is also not performed, because that functionality is only provided by the plugin.

To retain the default functionality to connect to publically accessible machines, an alternative implementation for the translation method was provided that simply forwards the IP and port that was passed into it.

Independence from the plugin was achieved with proper usage of abstract and virtual classes as well as factory classes. The code structure used for the final solution is shown in Figure 8.1.

10.3. Practicality and future developments

The remote plugin allows system operators to manage remote systems over the internet without needing to physically go to the location of the devices running the CDP applications while providing cryptographic security over the data sessions. This conserves a lot of time and transportation costs related to traveling to the location of the controllers, especially because most of ICD's clients have systems running on marine vessels.

This thesis focused on developing the remote plugin for SSH session creation for project deployment to remote devices but in the future it is planned to further develop it to be able to use this plugin in other services as well. If the interface and its accompanying classes that were placed inside the QSsh library were to be extracted into a separate library, the plugin could be used by other components like the database connection handler.

As the remote plugin is supposed to be sold separately from the CDP Studio, there has to be some anti-piracy mechanism to prevent illegal acquiring of the software. A licensing system has to be created. This is planned to be developed in the future.

11. KOKKUVÕTE

11.1. Ülesande tagamaad

Käesoleva magistritöö üleanne põhineb Norra ettevõtte ICD Industries Eesti haru ICD Software poolt välja pakutud probleemil. Nimelt töötab ettevõte hetkel välja tarkvarapaketti nimega CDP Studio, mis on mõeldud samuti ICD poolt loodud programmi CDP (Control Design Platform) aplikatsioonide loomiseks, paigaldamiseks ja haldamiseks. CDP Studio saab olema kasutajasõbralik tarkvara arendus keskkond, mis on loodud Qt Creator nimelise tarkvara baasil. Qt Creator on Qt Company poolt loodud vabavaraline ja kergelt modifitseeritav IDE (integrated development environment). ICD on võtnud aluseks Qt Creatori koodi ning hakanud sellele pistikprogramme juurde looma. Selle magistritöö ülesandeks on luua üks sellistest pistikprogrammidest.

11.2. Ülesanne

CDP Studio üheks ülesandeks on programmide paigaldamine kontrolleritesse ja arvutitesse. Qt Creator, mille peale CDP Studio ehitatakse võimaldab aplikatsioonide paigaldamist SSH ptotokolli abil kohalikus võrgus asuvatesse arvutitesse ja ka vabalt üle interneti ligipääsetavatesse arvutitesse. See funktsionaalsus kandub üle ka CDP Studiosse.

Kuna enamus ICD poolt hallatavad süsteemid on funktsionaalsusega, mille töökindlus ja turvalisus on äärmiselt kriitiline (merealused, naftaplatformid jms) ei ole võimalik neid juhtivaid kontrollereid vabalt internetist kättesaadavaks teha. Samas soovitakse nendesse masinatesse ligipääsu saada ka üle interneti ja mitte ainult kohalikust võrgust. Selle saavutamine ongi antud magistritöö ülesanne.

Vaja on luua CDP Studiole pistikprogramm, mis võimaldaks hallata CDP aplikatsioone kontrollerites, mis asuvad kusagil privaatses võrgus, millele pääseb ligi vaid läbi selles võrgus asuva tulemüüri masina. ICD sooviks on ka see, et kõnealune pistikprogramm oleks eraldi paigaldatav ja ei oleks CDP Studio baaspaketis, et seda oleks võimalik eraldi müüa. See tähendab, et antud pistikprogramm peab olema täielikult valikuline ning ülejäänud CDP Studio kood ei tohi selle olemasolust sõltuda. Samuti peab säilima Qt Creatori poolt tagatud funktsionaalsus, mis lubab aplikatsioonide paigaldamist ja haldamist kohaliku võrgu

masinates. Äärmiselt oluline on ka tagada andmeside krüpteeritus CDP Studio kliendi masina ning objektil asuva masina vahel, et vähendada turvariske.

Töö käigus keskenduti aplikatsioonide paigaldamiseks vajaliku koodi kirjutamisele, kuid koodi arhitektuur loodi selliselt, et see oleks kergelt kasutatav ka muu funktsionaalsuse jaoks nagu näiteks andmebaasiühenduste loomiseks privaatvõrku.

11.3. Lahendus

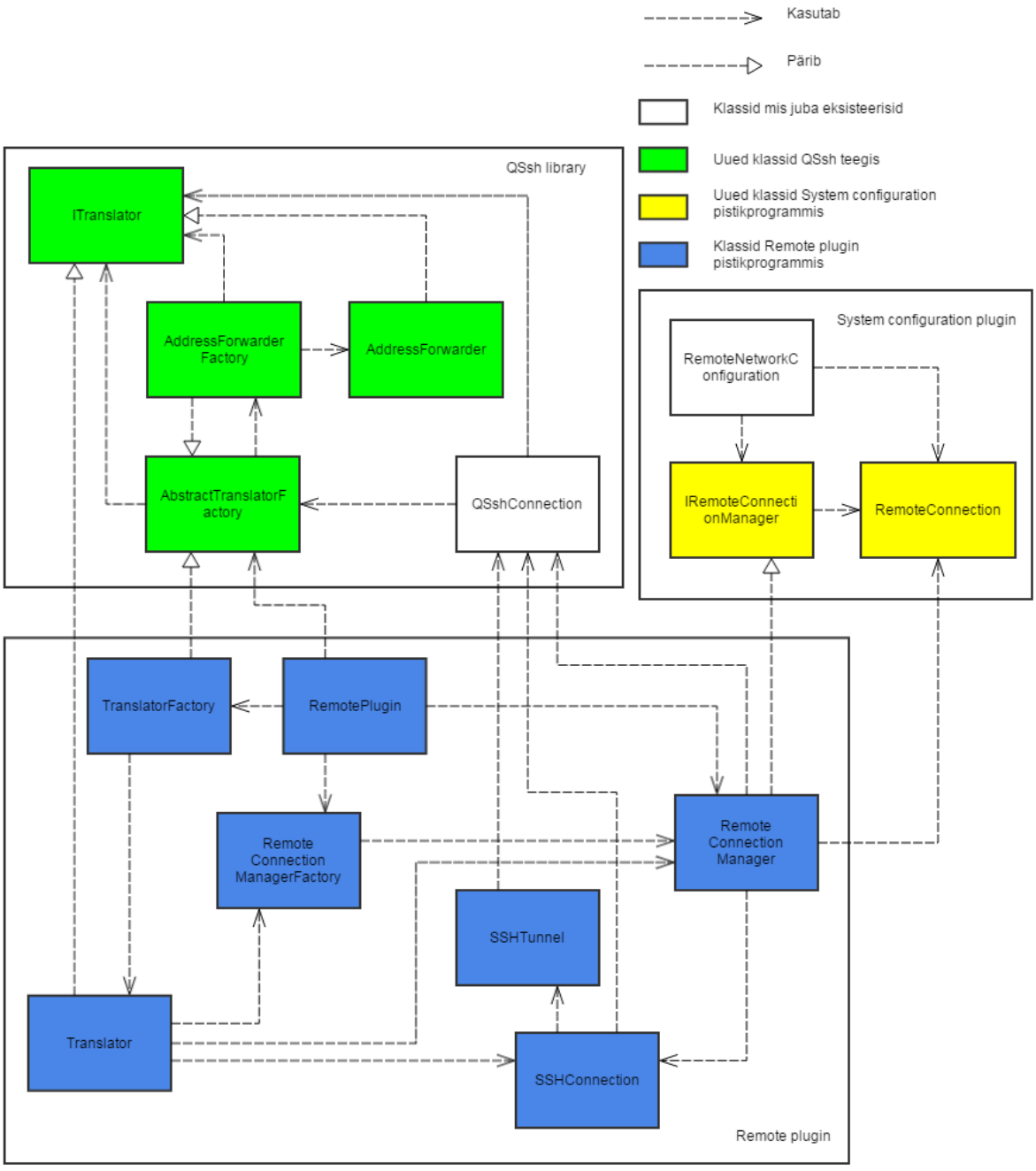
Otsustati kasutada ära võimalikult palju Qt Creatori lähtekoodi, mis tähendas seda, et andmeside turvalisuse tagamiseks kasutati SSH protokolli, mille kasutamise tegi äärmiselt lihtsaks Qt Creatoris asuv QSsh teek. Töö käigus tuli veidi QSsh teeki muuta, et pistikprogrammi funktsionaalsust toetada. Selle lähenemise puuduseks on see, et uue Qt Creatori versiooni välja tulemisel on vaja ICD poolt tehtud muudatused sellele manuaalselt üle kanda. Eeliseks aga on äärmiselt suur võit kirjutatava koodi mahus.

Sele 11.1 demonstreerib lõpliku lahenduse jaoks loodud koodi arhitektuuri. Iga kast joonisel sümboliseerib ühte C++ klassi, kusjuures valge joonega piiritletud klassid moodustavad järgmised komponendid:

- QSsh library – QSsh teek
- System configuration plugin – pistikprogramm, mis hoolitseb muuhulgas võrguühenduste haldamise graafilise kasutajaliidese eest
- Remote plugin – autori poolt loodud pistikprogramm privaatvõrkudesse ühendamise võimaldamiseks.

Nagu näha osutavad kõik remote plugin pistikprogrammi piire ületavad nooled sellest eemale. See tähendab et ei eksisteeri ühtegi programmlist sõltuvust remote plugin pistikprogrammist. Sellega on tagatud antud pistikprogrammi valikuline paigaldamine. Selle saavutamiseks kasutati sõltuvuse inversiooni põhimõtet, mis on üks S.O.L.I.D põhimõtetest [4], mida CDP Studio loomisel järgitakse. See tähendab et kahe klassi, mille sõltuvus tuleb ümber pöörata vahele tuli luua abstraktsiooni kiht. Seepärast on QSsh teegis ITranslator ja AbstractTranslatorFactory klassid ning system configuration plugin pistikprogrammis IRemoteConnectionManager klass. „I“ täht klassi nime ees tähistab sõna *interface*, ja näitabm et tegemist on puhtvirtuaalse klassiga. Sõna „Abstract“ tähistab abstraktset klassi.

QSshConnection klass on QSsh teegis asuv klass, mida kasutatakse SSH sessioonide loomiseks ja seadistamiseks ja mida töö käigus modifitseeriti.



Sele 11.1 Lõpliku lahenduse koodi arhitektuur

Loodud koodi tööpõhimõte seisneb selles, et iga kord kui hakatakse looma uut SSH sessiooni, tõlgitakse vajadusel privaatvõrgus asuva masina IP ja port kohaliku võrgu IP-ks ja pordiks. Selleks kutsutakse QSshConnection klassis välja ITranslator klassi meetod *translate()* (tõlgi)

mis vajadusel loob SSH tunneli läbi privaatvõrku kaitsva tulemüüri masina soovitud kontrollerisse ning tõlgib tollesse kontrollerisse viiva tunneli IP ja pordi kohaliku masina IP-ks ja pordiks. Seejärel on kasutajal võimalik ühenduda sellele tõlgitud aadressile täpselt nii nagu see oleks kohalikus võrgus asuv masin. See tunnel luuakse aga ainult juhul, kui kasutaja on eelnevalt loonud esmase SSH sessiooni tulemüürimasinasse. Kui aga esmast sessiooni pole või kasutaja üritab ühenduda kohalikus võrgus asuvasse masinasse, siis tunnelit ei looda ja meetod *translate()* ei tee midagi.

Kui remote plugin pistikprogrammi pole aga installeeritud, siis esiteks ei kuvata kasutajale välisvõrkude haldamise lehel üldse vastavat graafilist kasutajaliidest ning teiseks antakse QSshConnection klassile ITranslator klassi implementatsiooniks selline klassi instants, mille meetod *translate()* ei loo ei SSH tunnelit ega tõlgi ka sisse antud IP-d ja porti vaid lihtsalt tagastab need muutmata kujul.

Selline toimimine võimaldab kõiki SSH sessioone kasutada ühtemoodi, nagu need oleksid kõik loodud kohalikku võrku. Samuti on antud koodistruktuur loodud selliselt, et see on võimalik laiendada ka muudeks otstarveteks kui ainult SSH sessioonide loomiseks privaatvõrgu masinatesse. Kui QSsh teeki lisatud klassid välja tõsta eraldi teeki, siis saavad seda teeki kasutada ka muud teenused. See on tulevikus plaanis kuid jääb antud magistritöö mahust välja.

11.4. Praktiline väärtus ja tulevikuarengud

Lõputöö käigus loodud pistikprogramm võimaldab süsteemi operaatoritel mugavalt ja turvaliselt üle interneti objektidel asuvaid programme hallata. Peale kontrollerite monitoorimise saavad kliendid nendesse ka programmilisi uuendusi laadida ja palju muud teha. Lisaks spetsiifiliselt CDP aplikatsioonidega seotud asjade on võimalik üle SSH sessiooni teha ka kõike mida SSH protokoll lubab.

Selline funktsionaalsus hoiab oluliselt kokku operaatorite aega ja transpordile kuluvat raha sest praktiliselt kõik vajalik on võimalik ära teha üle interneti kaotamata oluliselt andmeside turvalisuses.

Tulevikus on plaanis remote plugin pistikprogrammile juurde programmeerida litsentseerimissüsteem, mis ei lase seda illegaalselt paigaldada sest nagu öeldud, ei kuulu see

pistikprogramm CDP Studio baaspaketi juurde ja tuleb eraldi osta. Samuti on plaanis programmi funktsionaalsust laiendada selliselt, et seda oleks võimalik kasutada ka teistel teenustel nagu näiteks andmebaaside haldamise teenus.

12. REFERENCES

1. The Qt Company webpage [WWW] http://www.qt.io (04.05.2015)
2. ICD Software's homepage [WWW] http://www.icdsoftware.no/products/control-design-platform (19.05.2015)
3. SSH protocol developers homepage [WWW] http://www.ssh.com/ (19.05.2015)
4. CODE Magazine homepage [WWW] http://www.codemag.com/article/1001061 (22.04.2015)
5. Agile Data [WWW] http://agiledata.org/essays/tdd.html (23.04.2015)
6. The Cisco Learning Network [WWW] https://learningnetwork.cisco.com/blogs/network-sheriff/2008/09/22/sshv1-or-sshv2-whats-the-big-deal (24.04.2015)
7. The libssh homepage [WWW] https://www.libssh.org/ (24.04.2015)
8. The libssh2 homepage [WWW] http://www.libssh2.org/libssh2-vs-libssh.html (04.05.2015)