TAMPERE UNIVERSITY OF TECHNOLOGY

TAPIO PIIPARI

# DYNAMIC CONFIGURATION MANAGEMENT

Master of Science Thesis

Examiner: Prof. Tommi Mikkonen (TUT)
Supervisor: M.Sc. Petteri Kylliäinen
(Cargotec Finland Oy)
Examiner and topic approved in the Faculty of Computing and Electrical Engineering council meeting on 3rd of October 2012

# TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Signaalinkäsittelyn ja tietoliikennetekniikan koulutusohjelma
**TAPIO PIIPARI: Dynaaminen konfiguraationhallinta**
Diplomityö, 42 sivua, 0 liitesivua
Maaliskuu 2013
Pääaine: Sulautetut järjestelmät
Tarkastajat: Professori Tommi Mikkonen
Avainsanat: konfiguraationhallinta, konfiguraation jakaminen

Viime vuosien aikana ohjelmiston määrä kontinkäsittelykoneissa ja niiden hallinnassa on kasvanut ja automaatiolla on yhä suurempi rooli konttiterminaalien toiminnassa. Automaatiojärjestelmän pitää pystyä mukautumaan muuttuvaan ympäristöön ja asiakasvaatimuksiin. Mukautumisen lisäksi järjestelmän tilaa pitää pystyä tarkkailemaan, jotta voidaan varmistua että se on toteuttanut vaaditut muutokset onnistuneesti.

Tässä diplomityössä tutkitaan Cargotec Finland Oy:n kehittämän kontinkäsittelykoneiden tiedonhajautusjärjestelmän, UniQ:n, konfiguraation hallintaa. Työssä esitellään uusi ohjelmisto, UniConf, joka on suunniteltu konfiguraatiotiedostojen muokkaamiseen. UniConf mahdollistaa konfiguraatiotiedostojen helpon muokkaamisen graafisella käyttöliittymällä, osaa validoida luodut konfiguraatiotiedostot, sekä luoda tarvittavan asennuspaketin jolla tiedostot voidaan asentaa kontinkäsittelykoneisiin. Lisäksi konfiguraatiotiedostojen hajauttamisen toteuttamiseksi tutkitaan valmiita konfiguraation hajauttamisjärjestelmiä. Tutkimus suoritetaan järjestelmien valmistajien tarjoamien dokumentaatioiden avulla tutkimalla niistä järjestelmien keskeisimmät ominaisuudet kontinkäsittelykoneiden konfiguraation hajauttamisen kannalta.

# ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Signal Processing and Communications Engineering
**TAPIO PIIPARI: Dynamic Configuration Management**
Master of Science Thesis, 42 pages, 0 Appendix pages
March 2013
Major: Embedded Systems
Examiner: Prof. Tommi Mikkonen
Keywords: configuration management

During recent years, the amount of software in container handling equipment has increased and automation has a greater role in container terminal operations. Automation systems must be able to constantly adapt to changing environments and container operator requirements. In addition, it should be possible to monitor the state of the system in order to confirm that it has adapted to the new configurations correctly.

The focus of this thesis is on the configuration management of UniQ which is a data distribution framework developed by Cargotec Finland Oy. Within this thesis, a new piece of software, called UniConf, will be introduced for editing the configuration files of UniQ. UniConf allows users to edit configuration files with graphical user interface. UniConf can also validate the configuration files and create necessary installation packages so that configuration files can be installed into container handling equipment.

To accomplish distribution of configuration files, existing configuration distribution systems are studied herein. This research has been done by studying manuals and other documentation offered by manufacturers of configuration distribution systems. The study focuses on the most crucial aspects concerning configuration distribution for container handling equipment.

# PREFACE

This thesis is made for Cargotec Finland Oy at Tampere Finland to solve the increasing challenges with UniQ configuration management.

I would like to thank my colleagues at Cargotec Finland Oy for all the help I have received, particularly the guidance from Petteri Kylliäinen and Aleksi Lehtonen. I would also like to give thanks to Professor Tommi Mikkonen for the examination of this thesis. In addition to the help from colleagues at Cargotec Finland Oy, I also received help from Veijo Arponen from Arpotechno, who allowed me to use his code in UniConf. Thanks also to Stephanie Levy for language briefing.

Tampere, March 2013

Tapio Piipari
Vaajakatu 5 E98
FIN 33720 Tampere
Tel.:+358 50 347 8533
tapio@piipari.fi

# CONTENTS

# 1. INTRODUCTION

During recent years the amount of software in container handling machines has grown, and operators are more often trusting automation in their container yards. Automation improves the efficiency of container terminal, but also increase the complexity of the automation system and the software. Container terminals are demanding new features and are adding new equipment into the system. The automation system must be able to dynamically adapt to this constantly changing environment. In addition it should be possible to monitor the state of the system to confirm that it has adapted the new configurations correctly.

The purpose of this thesis is to describe the needs of configuration management during container terminal automation system delivery project. Also maintenance of old projects is considered. The thesis will focus mainly to how to manage configuration files for UniQ system, which is a container terminal data distribution system developed by Cargotec Finland Oy.

The thesis consist of a literature survey of existing configuration management systems and developing a easy to use configuration file editor with capability to validate configuration files automatically before deploying them. The literature survey focuses on how to deploy new configuration settings to all the nodes in the network and monitor the state of the system. The implementation of the configuration deployment is not part of this thesis but suggestions are given about it.

The remainder of this thesis is structured as follows. Chapter 2 is a general introduction of configuration management, and it includes explanation why configurable is needed and how it can be done. Chapter 3 explains the challenges that this thesis will answer. It will go through the structure of container terminal, the current procedure for configuration management, and the requirement set for the configuration management system. Chapter 4 focuses on comparing existing configuration maintenance systems that can be used for configuration deployment and verification. At the end of the chapter is a summary that collects the features and methods all the software use. Target is to understand what can be done with existing systems and what kind of methods they use. Chapter 5 explains the architecture and design of configuration editor design in this thesis, the UniConf. Chapter 6 introduces most essential features of UniConf and how to use them. Finally, Chapter 7 concludes this thesis and considers the further developing goals as well as proposals

for improvement. Also, some suggestions on how to integrate UniConf and UniQ delivery project to the selected configuration maintenance system and to the related development projects are given.

# 2. CONFIGURATION MANAGEMENT

The amount of software in automation devices has grown to the extent that it can be the biggest single expense in the product. Although software is easy to copy after it has been created, often that is not possible. Customers can have different needs and they want tailored features to products they buy. Customers' needs can be fulfilled by reprogramming part of the software differently for every customer. This solution is easy in beginning but can cause severe consequence later. Manufacturer must keep the code updated separately for every customer and new features and bug fixes needed to be tested separately for every customer. To reduce this extra work, software should be designed to be configurable so that different features can be switched on and off according to customers' needs without reprogramming anything. Configuration can be done during the compiling process or at runtime by loading different configuration settings.

## 2.1 Basic definitions

Configuration management (CM) organizes and groups configuration items, such as source files, libraries and documentation. Configuration management is an essential part in modern software projects since projects consist of great numbers of files and libraries often collected from multiple sources. Accordingly, it is impossible to organize all of them without any tool or guidelines. International Organization for Standardization (ISO) defines that CM is [13]:

> *"A discipline applying technical and administrative direction and surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements."*

Every software project includes configuration items (CI). A CI is the smallest item that configuration management can manage. Internal version control of the CI is handled by a version control system, ISO defines CI as follows [13]:

> *"An aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process."*

CIs are often dependent upon each other. For example, module A depends upon library B, meaning that module A needs library B in order to compile or uses it in run time. Configuration management tools help developers to understand and organize these dependencies. A example of a very simple configuration management tool is *Make*. Make was built in the mid-1970s and has very simple syntax to describe dependencies between modules. Make can also automate the compiling process and make it faster by compiling only those modules that have changed since a last compiling. [1]

Make uses a file called *makefile* which specifies how modules depend on each others. The basic way to describe a dependency is shown in Listing 2.1. In the example `module5` depends on `module7` and `module8`, stating that if `module5` is out-of-date respect to its dependencies, then the associated command sequence has to be run to bring `module5` up-to-date. [1]

```
1 module5: module7 module8
2   commandsequence
```

Listing 2.1: Basic dependency declaration in a Makefile [1]

The term *baseline* is often associated with CM but no common and universal agreement exist on its meaning [11]. Therefore one should be careful and define it well when using it to avoid misunderstandings. This thesis use baseline as defined by ISO [13]:

> *"(1) specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures (2) formally approved version of a configuration item, regardless of media, formally designated and fixed at a specific time during the configuration item's life cycle"*

According to the definition, baseline can be understood in two ways. Baseline can be a set of requirements that defines specific project milestone or a particular approved version of CI. In both cases baseline and all the changes to it must be formally approved via project's change control process. Thus baseline and all the approved changes to the baseline, represents the current approved configuration.

CM can also save information about the compiler and its version, preferred hardware, and other things related to the code and its handling. CM can also include hardware configurations, both electronic and mechanical. Traditional software developing does not care much about underlying hardware, however when producing embedded systems, it is important to be aware of the compatibility of software and hardware. [15]

## 2.2 Version Control

Version control can be considered as a subset of configuration management. While configuration management controls the whole software project, version control focuses on a single configuration item and its history. Version control does not document how modules depend upon each other, but instead focuses on internal structure of a module and the associated change history over the life cycle of a module. Version control tools are designed to describe change process and allow developers to inspect older versions of configuration items. ISO defines Version Control as follows [13]:

> *"Establishment and maintenance of baselines and the identification and control of changes to baselines that make it possible to return to the previous baseline."*

Even in simple software projects, there are often hundreds of small changes carried out every day, and thus managing all these changes without a proper system would prone a heavy challenge, both in time and effort, and could therefore be considered impossible. Furthermore, if a project includes multiple persons in different locations editing the same information at the same time, errors are likely to occur. In addition to changes in information, version control also keeps track of who has made a change and when. [23]

In industry environment, product's life cycle can extend for very long durations. For example, Cargotec products' typical life-time can be over 10 years [16]. As such, in order to be able to provide support, product manufacturers must know which configuration has been delivered to the customer and which is the newest configuration that can be delivered without effect to other systems. Therefore, it is essential to keep track of all the changes in code and also in hardware to be able to know which software version works in which hardware version. In this context, hardware can mean both electronics and mechanics. During product development, configuration management can feel like unnecessary bureaucracy, but later it will reduce work enormously and offer better service for customers.

Version control is a rather well researched area, and many mature tools exist which can be efficiently used [23]. The first real version control software was Source Code Control System (SCCS), published in early 1970 by Marc J. Rochkind [23, 22]. The early version control software were designed to be used locally on one machine, but when entering into the 1990's, version control software like Concurrent Versions System (CVS) started to support networking [23]. The first networking version control software were based on centralized server-client model, but later it became clear that server-client model did not scale easily for large distributed projects. To solve this problem, distributed version control software were developed, such as

BitKeeper, Mercurial, and Git [4, 18, 10]. In distributed version control there is no need for a centralized server and every developer can work independently and share progress with the other developers. A benefit of distributed system is that it does not have any single point of failure that could compromise the whole project.

Besides versioning project with respect to time, they also allow multiple parallel versions of a project, enabling the users to easily see how a modification would work without the need of changing the current main branch of the project. Tools make it easy to later merge changes to the main developing branch by implementing automatic merge functionality for text files. If two changes in the same file are in different place, those files can be automatically merged. This works well with source code files, however can raise problems when tried with structured files such as XML.

## 2.3 Configuration management process

This section will introduce a CM process as defined in the IEEE Standard for Configuration Management in Systems and Software Engineering (IEEE Std 828), which establishes the minimum requirements for processes for CM in systems and software engineering. The latest reversion of the standard was published in 2012 and is targeted mainly to people responsible for planning, managing, and performing CM. [12]

There are numerous standards available, and often different books introduce their own CM processes as well as the segregation of different CM styles for different areas of industry. Standards and other guidelines reflect the experience of experts and can provide inspiration on to how to handle CM [11]. More important than following a specific standard is to define a CM process that is suitable for a company and its needs, and accordingly, implement and follow the CM process plan strictly.

CM process in IEEE Std 828 consist of nine lower-lever process. They are reviewed and detailed in the following paragraphs. The lower-lever processes are (as ordered in the standard):

- CM planning
- CM management
- configuration identification
- configuration change control
- configuration status accounting
- CM auditing
- interface control

- supplier configuration item control
- release management.

*CM planning lower-level process* produces a CM plan document and schedule for the other lower-level processes. This plan includes what status information is needed regarding CIs, how the information is to be reported, and how often reports are created and distributed. The plan should also contain naming convention for product builds, required resources (human and physical), CM tools and equipment, estimated costs of activities, dependencies between CM activities and other project activities, among others. In large organisations it can be practical to distribute the CM planning and management amongst two or more functional units. In that case, a streamlined organisation wide CM function is needed to maintain the standardization between the units. [12, 24]

The purpose of *CM management lower-level process* is to implement, monitor, control, and improve CM services. This includes monitoring of the CM activities and tasks to ensure accuracy and that they are being carried out according to the CM plan. All of the required personnel are to be informed of their responsibilities and provided with proper training in order to carry them out efficiently and effectively. CM management also takes care that all of the systematic tools and environments needed for CM are properly installed and configured. If necessary, CM management can modify the CM plan during the life cycle of the product. The CM plan should be regularly reviewed and proposed changes evaluated. [12]

In *configuration identification lower-level process* main responsibilities are to establish the structure and hierarchy of CIs, identify CIs, name them, describe them, and place them in CM repository. This process determines the naming convention of CIs and what information is used to describe the CIs. The naming convention ensures that every CI has a unique name and that different versions can be distinguished. Items that are commonly identified as CI include, but are not limited to, interface specifications, design, code, builds, database schema and scripts tests, and manuals. After the CIs are identified, any change to the CIs are performed only as an outcome of the change control lower-level process. [12]

To enable placing CIs in CM repository, a CM repository must be established and documented. If needed, a repository should be created for both electronic and physical CIs. CM specifies procedures to backup baselined data and ensure that procedures are deployed. [12]

A related article that should also be documented is the process by which baselines are established, the events that establish a baseline, items that belong to the baseline, and procedures to change the baseline. These must be defined and placed in the CM plan. [12]

*Configuration change control lower-level process'* purpose is to maintain the integrity of the product in all its states. Change control applies only to CI's for which a baseline has been established. Therefore, if the item has not yet been submitted to a CM repository and baselined, the change control is not responsible for it. An item should not be submitted to CM repository before it passes the organisation's internal quality requirements, typically developer's own tests. [12]

Before changes are approved, they must be carefully examined by the configuration control board (CCB) to determine how the change will effect other CIs. If the change is not necessary, or does not have significant benefit, it should not be applied [14]. Changes should be clearly communicated to all affected parties, this is particularly important for requirement and design changes. A basic change control process is shown in Figure 2.1. [12]

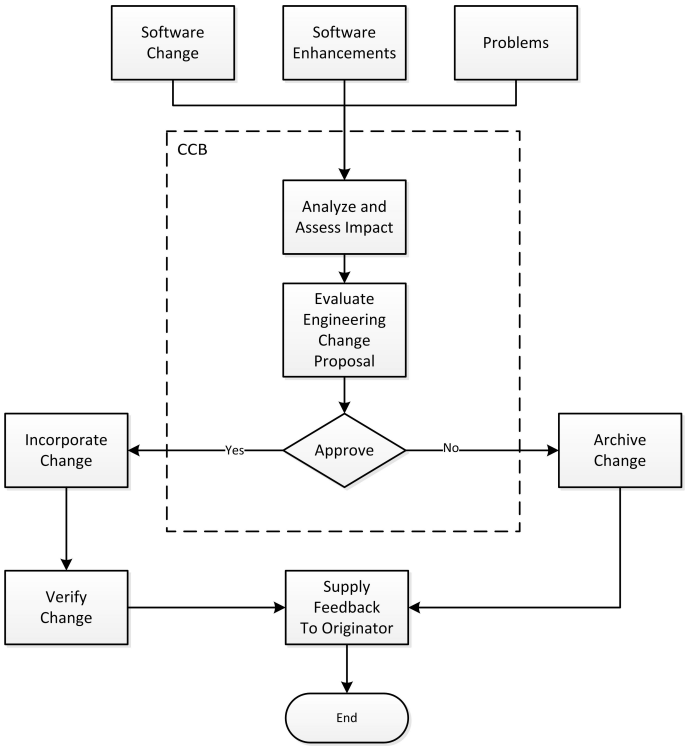


Figure 2.1: A basic change control process.

The purpose of *configuration status accounting lower-lever process* is to record and report critical information about assets to management and the project team. Status accounting can provide reports about project work during a given period and develop estimates-to-complete. The CM plan contains what type of information project members expect and how often the information is reported or can it be required on demand. Each CI shall be accounted in each stage of its life, from its initial identification to its end-of-life. At a minimum, the following data elements must be included [12]:

- The status (such as changed, stable, archived) of each of the current CIs.
- Identification of the current baseline configuration of all items comprising the product.
- The state of all the changes, whether implemented, rejected deferred, or pending.
- Relationship data from requirements to tests to implementation and vice versa.

*CM configuration auditing lower-level process* will objectively assess the integrity of the product development process and the product itself. Auditing of development process should be performed during the product development life cycle. Its purpose is to inspect the traceability between requirements, other product models (use case, design, deployment, implementation, etc.), test artefacts, and execution of the tests. The auditing of the product itself should be performed at least once before releasing the product to inspect that the right product is being properly assembled, and changes are managed on the different product artefacts. If any nonconformities are detected, it is the auditor's duty to report them to the appropriate persons, as defined in CM plan, for correction. [12]

*Interface control lower-level process* manages interfaces between CIs. An interface may be between two CIs either developed internally to the project or between a CI developed to the project and an external CI. An interface represents at least three CIs: the interface specification itself and CIs on either side of the interface. CIs can be software, hardware, or some other type of CI. For every interface, the following must be defined: the nature of the interface (data, hardware, software), the affected organizations, and the technical specifications. Specifications for each interface shall be placed in designated repository and be subject to the project's CM control, audition and accounting lower-lever processes. [12]

*Supplier configuration item control lower-level process* manages the incorporation of CIs developed outside of the project. The supplier of the CI can be a vendor, a customer, another project, or other source. This process places the CI under CM and defines how change management, auditing and accounting is handled for the CI. Also, requirements regarding the supplier of the CI and plan how the supplier is monitored are defined. [12]

*Release management lower-level process* ensure that the proper set of CIs are delivered to the designated receiving party in the designated form to the designated location. Release can mean either making the product available to the customer or internally to other projects or testing groups, and is maintained for the life of the product. After the release has reached its end-of-life, the CM authority archives it and all CIs belonging to it, making the release unavailable via normal channels,

and then marks the release as archived. This final release itself is also considered a CI. [12]

Worthy of note is that CM is not a project that has the begin and the end, but rather a process that never ends. CM must always be considered in a company, it is not suffice to organize it once and then forget it. CM must be developed and continuously monitored and implemented in all concerning projects and processes. There should be an assigned person or a group for maintaining and developing the CM process and tools. It is both a historical and auditable trail of life cycle of a product, and is thereby imperative that all responsible are actively performing the required duties and adhere to the requirements as set out in the plan.

# 3. CASE STUDY: UNIQ

UniQ is a pervasive automation platform for container terminals, developed by Cargotec since 2009. The aim of UniQ is to improve the overall performance and management of container handling equipment (CHE) in a container terminal. UniQ is designed to be a highly customizable and modular system. It has the capability to add new services later without the need to reconfigure the old system. UniQ makes it possible for customers to buy the UniQ system in piece in order to lower the threshold of starting to use the system. A standard communication interface also guarantees that any future innovation is usable and compatible with existing UniQ solutions. Figure 3.1 shows a simple physical structure of UniQ network in a container terminal.

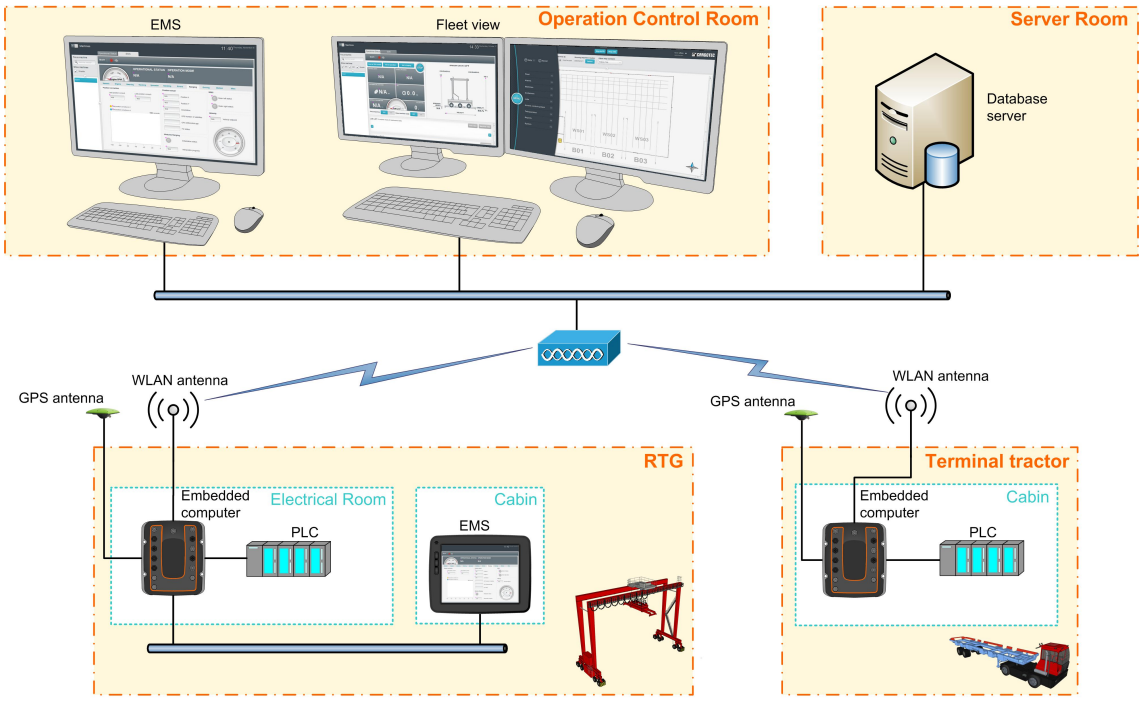


Figure 3.1: Example of possible UniQ system.

UniQ can be split into two parts; UniQ platform and UniQ graphical user interface (GUI) framework. The main feature of UniQ platform is the messaging layer, which introduces a common communication interface between independent services. UniQ GUI framework offers a generic GUI to control and monitor services. It is an application built on top of the UniQ platform. This chapter will focus on the UniQ

platform, since the configuration of UniQ GUI framework is not part of this thesis.

## 3.1 Messaging layer

The service instances of UniQ platform are called peers. Peers are usually located in separate computers, but it is possible to run multiple peers on a single computer. Every message in UniQ platform must have a global identifier, that is, a name. The naming process is called tagging, hence all the data is tagged. The term *Tag* stands for message type whereas *tagged item* is the actual tagged data in the message. Due to a limited network bandwidth, messages can not contain description, type, unit etc., as strings. To solve this problem, all messages are enumerated as numbers and delivered as such. In order for peers to be able to interpret the messages, they must have a global enumeration map, and is known as the *tag map*. Tag map is one of the most crucial configuration files in the UniQ platform system. [17]

UniQ platform implements messaging layer via a specific cross-platform software component known as the *tag facade*. Tag facade is implemented as a dynamically linked library and is written in Qt. Tag facade uses tag map for validation and interpretation of the messages. Every peer in UniQ system uses its own instance of tag facade to connect to the UniQ network. [17]

## 3.2 Peers

In the UniQ network there are many different types of peers. The most common types are Data Acquisition Service (DAQ), Equipment Monitoring System (EMS), Fleetview (FV), and gateway peers. UniQ system can also include access control of the container terminal, i.e., ports.

DAQ is a service process running in computer called CHE Controller Unit (CCU), which is an embedded computer on board of a CHE. CHE is a generic name for container handling device which is used to lift and move containers in a container terminal. DAQ's task is to acquire data from the CHE and edit it to the form defined in the tag map. The CHEs move around the container terminal while their DAQs are typically connected to the UniQ network via a wireless local area network (WLAN). However connection to the UniQ network is not reliable as container stacks and other CHEs can block the WLAN signal temporarily. Figure 3.2 shows typical CHEs.

EMS is a GUI for monitoring a single CHE. It can be located in the cockpit of the CHE or in the operation control room of the container terminal. EMS collects data from other peers and shows it to the user, it does not store old data or create new data by itself. FV is very similar to EMS, but it collects data from multiple different CHEs and shows them to the user simultaneously on a single screen. Both EMS and FV are based on the UniQ GUI framework.

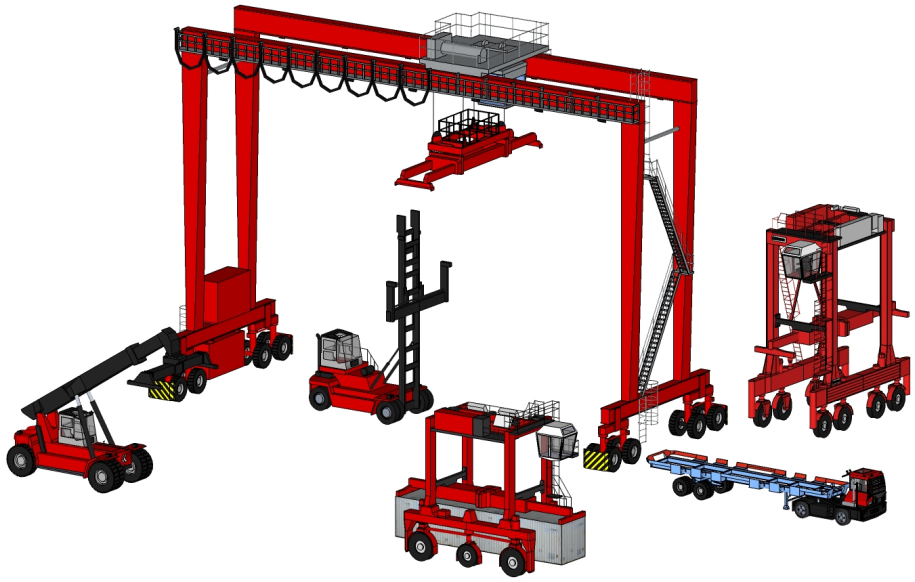

Figure 3.2: Examples of different types of CHEs.

Gateway peer acts as an interpreter between UniQ and another system. Gateway peer creates a facade that is standard UniQ interface to UniQ side while hiding the other system. Gateway peer translates UniQ messages into a form that is understandable for the other system, and vice versa. Typically, the other system is a database or another network.

A peer can act as a publisher, a subscriber, or both, of the data. Peers communicate by creating a communication channel i.e. a virtual two-way connection between each other. After the channel is created, peers can send messages to the other peers or order messages from the other peers. In order to be able to create a channel, at least one of the two peers must know other's name. To get this information, a peer needs a configuration file called *commconf*, which contains the description of the surrounding UniQ network and other peers.

Besides the tag map and commconf, peer also needs other configuration files. The most commonly needed configuration files are listed in Table 3.1. In addition to UniQ configuration files, peer might need several internal configuration files to be able to work correctly. These files depend on the type of the peer and its underlying hardware and operating system. Typically these files tells peer's IP settings and WLAN name and password.

UniQ configuration files are based on XML. Whereas non-UniQ files can in any format, typically they are configuration files for the operating system. Some of the files are exactly the same for every peer of a certain type, yet most of the files have little variance between the peers. In the initial installation, all files are installed into the peer simultaneously. For convenience all the files should be created using the same configuration tool.

| Name | Description |
|---|---|
| alarmlist.xml | List of all the alarms that are known in the system. Stored in XML file but easy to represent in table format. |
| commconf.xml | Describes the surrounding network and other peers. Single file is simple but the whole network is hard to understand without a schema picture. |
| datasourcemap.xml | Maps tagmap's data items to a local data source. Easy to represent in table format. |
| eventlist.xml | List of events which can be sent from different actions. Can be represented in table format but rows can have different amount of columns. |
| eventfactoryconf.xml | Descriptions of state machines that can create events by combining data from multiple sources. File is hard to understand in textual format, should have graphical state machine editor. |
| interfaces | Used only in CCUs. Contains the ip settings for network adapter. Not part of the UniQ system, but necessary for CCU to work. |
| parameters.xml | Parameters for Interprocess Communication. |
| tagmap.xml | Descripts every data item, its unit, structure (integer, string, floating-point number, byte array) and name. Easy to represent in table format. |

Table 3.1: Description of the most commonly used configuration files

## 3.3 Examples of configuration files

Next, the most important configuration files for UniQ system will be represented. While it is worth of note that there are also other configuration files, their structure resembles the files described here and therefore they are not described here.

### 3.3.1 Commconf

Communication configuration file (commconf) describes the surrounding UniQ network and other peers wich are available to connect. A commconf file must have at least one connection for a peer to be able to use the network. Every connection holds one or more gateway and zero or more peers for connection. Gateways are UniQ messaging servers, usually running in localhost or in a centralized server. Gateways can create networks between other gateways so that every peer does not have to connect to the same gateway.

Listing 3.1 shows a typical example of a commconf file in a DAQ type of peer. It tells that the peer's name is SC0102, it connects to a gateway located at 10.45.51.20, and has two channel peers; DATABASE and EMS. Channel to EMS peer is dynamic, meaning that the EMS peer does not list peer SC0102 in its commconf file. In

contrast, channel to the DATABASE is not dynamic, meaning the DATABASE peer has to list SC0102 in its own commconf file in order for channel to able to open successfully. It is possible to list more gateways for one connection, but only one is used at a time. The other gateways are spare gateways, made available if peer cannot connect to the first gateway.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<commconf xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <connection clientname="SC0102" >
    <gateway daemon_ip_addr="10.45.51.20"/>
    <peer clientname="DATABASE"/>
    <peer clientname="EMS" dynamic_peer="true"/>
  </connection>
</commconf>
```

Listing 3.1: A simple commconf.xml file

Although commconf is a relatively simple file, it is hard to validate on the system level. Even if every peer uses the same gateway, the IP address of the gateway depends on the logical structure of the network. It is impossible to know for sure if the IP address is correct or not without knowing the exact structure of the underlying network. Also, client names must be unique within one UniQ network, but according to commconf file it is impossible to know if two peers are connected to the same network or not.

### 3.3.2 Tag map and data source map

Tag map is crucial for the Tag Facade to be able to parse messages. Every peer must have the same tag map to avoid misprocessing of the messages. Tag map consists of tags, as shown in Listing 3.2. The tag describes a machine speed message, and the message has a double type value, which tells machine speed in kilometers per hour. Description text can be shown to the user to describe the data. Both the id and the name must be unique globally.

```
<tag id="40003">
  <name>MACHINE.SPEED</name>
  <desc>Machine speed</desc>
  <quantity>speed</quantity>
  <struct>double</struct>
  <unit>km/h</unit>
</tag>
```

Listing 3.2: One tag in the tag map

Data source map is used in DAQ peers to describe how the data should be read

from the source of the data, which can be a Programmable Logic Controller (PLC) or Controller Area Network (CAN) bus. Data source map describes how the data read from the source must be interpreted and modified before publishing it for the other peers. Listing 3.3 shows an example a data source map tag which is a pair for tag map tag from Listing 3.2.

```
1 <tag id="40003">
2   <desc>Driving speed</desc>
3   <source>CCU_siemens_if</source>
4   <specifier>DB202</specifier>
5   <extractor>DBD50</extractor>
6   <modifier>float, *0.06</modifier>
7   <timeout>M2</timeout>
8 </tag>
```

Listing 3.3: One tag in the data source map

Listing 3.3 defines that the speed should be read from Siemens' PLC and that the PLC gives the value as a float. Modifier in line 6 says that the value must be multiplied by 0.06 to convert its unit to km/h. In line 4 and 5 specifier and extractor point to the exact location of the data. The content of these fields are source specific. Id number connects data source map tags to tag map tags. [5]

### 3.3.3 Text files

The non-UniQ configuration files do not have any common type, so they are handled as pure text files. Typically non-UniQ files are Shell scripts or Unix-style configuration files, but they can be also something else. The Shell scripts are used for starting up the system and installing applications. The Unix-style configuration files are typically used by the operating system to initialize its own settings. Listing 3.4 shows a file named *interfaces* which is used to define the network interface settings for CCUs.

```
1 # The USB network interface
2 # NOTE! If set static IP, remember update /etc/udhcpd.conf too for
  DHCP-server lease area!
3 iface usb0 inet dhcp
4
5 # The primary network interface
6 iface eth0 inet static
7 address 192.168.100.101
8 netmask 255.255.255.0
9 gateway 192.168.100.1
```

Listing 3.4: A typical Unix-style configuration file

The file in Listing 3.4 is a Unix-style configuration file. Because there is no common standard for Unix-style configuration files they are defined here. Unix-style configuration file has key-value pairs that are typically separated with equal sign ('=') or with space, as in Listing 3.4. Comments start with the hash sign ('#'), meaning that the rest of the row is comment and not part of the configuration. The file in Listing 3.4 contains network interface settings, namely IP numbers. It defines two network interfaces, first one on line 3 use DHCP to obtain their IP settings and later on lines 6 to 9 use static IP settings.

Validating this type of configuration files is hard, since they do not follow any well defined schema and value types, such as string, integer, or a single character, are not defined.

## 3.4 Configuration management in UniQ customer project

This section will go through how the CM and version control is handled currently in UniQ customer projects and what kinds of improvements the process needs. The first part explains the typical workflow of UniQ customer project and how the configuration files are handled. The later part focuses on the requirements for the configuration management tool that is created in this thesis.

### 3.4.1 Project workflow

Figure 3.3 shows the basic steps in the delivery process of the UniQ system from the project engineer's point of view. In the starting meeting of the project, the project engineer receives the basic information concerning the project and the customer. After the project starts, the project engineer confirms what data the customer wants to be able to monitor, and, according to customer needs, designs the views, EMSs, and Fleetview. The customer may also at this time require changes to the views before accepting them. In this event, and following mutual concurrence between the customer and Cargotec of the view specifications, the project engineer starts to create configuration files for UniQ. This involves the need by the project engineer of information from the customer about the infrastructure, such as IP addresses and how the peers are connected to the physical network. The configuration files are tested in Cargotec's testing facilities before deploying them into the production environment. The order of the stages can vary depending upon in which order the customer provides the needed information. The basic information and issues of the projects are stored using a project management system.

In the beginning of the project, the configuration settings are stored and modified only in the project engineer's own PC. This is viewed as an acceptable process as it has not caused problems as it is the standard that every project has only one

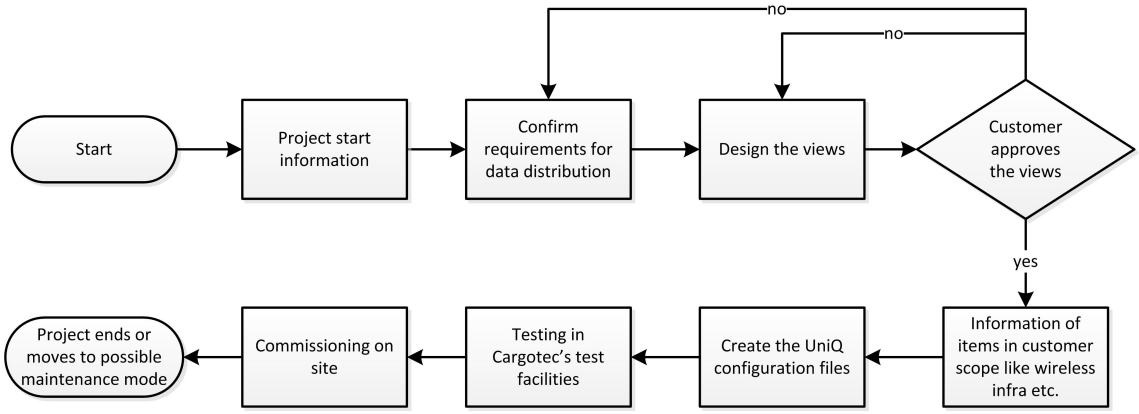


Figure 3.3: UniConf customer project workflow. [16]

project engineer in charge of creating the configuration files. Currently there is no unified way to handle changes and version control. Information collected from the customer and from other sources is stored in emails, Microsoft Excel or Microsoft Word files locally in project engineer's computer. Version control is typically handled by adding an editing date and time to the name of the project directory. After the project is finished the latest version of the related documents are copied and archived to Document Management System (DMS), which includes a simple version control system. Also, separated version control systems, such as Subversion and Git, are available for backing up and to help the development of configuration files before they are finished and added to the DMS. [19]

Since a common baseline does not exist for the configuration files, configuration files from old projects are used as a baseline for new projects. Furthermore, every project engineer has their own naming and numbering convention. The lack of common standards makes sharing of configuration files problematic, since there is no guarantee that files made by different engineers will work together. This creates excess work, as all of the configuration files must be checked every time they are used in a new project. [16]

The installation of the applications and configuration files takes place usually in the container terminal of the customer and is done by using flash drivers with automatic installation script. Installation over SSH is also possible, but it is not usually used for initial installation because of often poor network and lack of automating tools. SSH is used mainly for monitoring the peers and update them after the initial installation. The projects should end at commissioning stage, however, as some of applications may not have yet reached the mature state, changes to the applications and configuration files are often needed afterwards. Updates done after the commissioning are carried out using SSH connection from Cargotec's office to reduce traveling expenses. [19]

### 3.4.2 Requirements for configuration management tool

Requirements for the UniQ configuration management tool can be divided into two main parts. The first more acute part is configuration editor and validator. Configuration editor is used to collect all the needed configuration files together, keep them organized and automatically validate them. The second part of configuration management tool is configuration deployment and audition tool. It is needed for sharing the configuration files and other CIs to the peers and monitoring the peers.

Figure 3.4 shows the network where UniQ configuration distribution takes place. On the right side of the figure the project engineer is located in Cargotec's office, which is the normal situation. The project engineer uses configuration editor to edit the configuration files and then transports them to the configuration distribution server. Before the configuration distribution server is setup in customer's server room, project engineer can use local version control server to store the configuration files. The version control server can also act as a backup server so the configuration distribution server can be restored. After the commissioning the main repository for configuration files is in configuration distribution server and files in version control server are not guaranteed to be up-to-date.

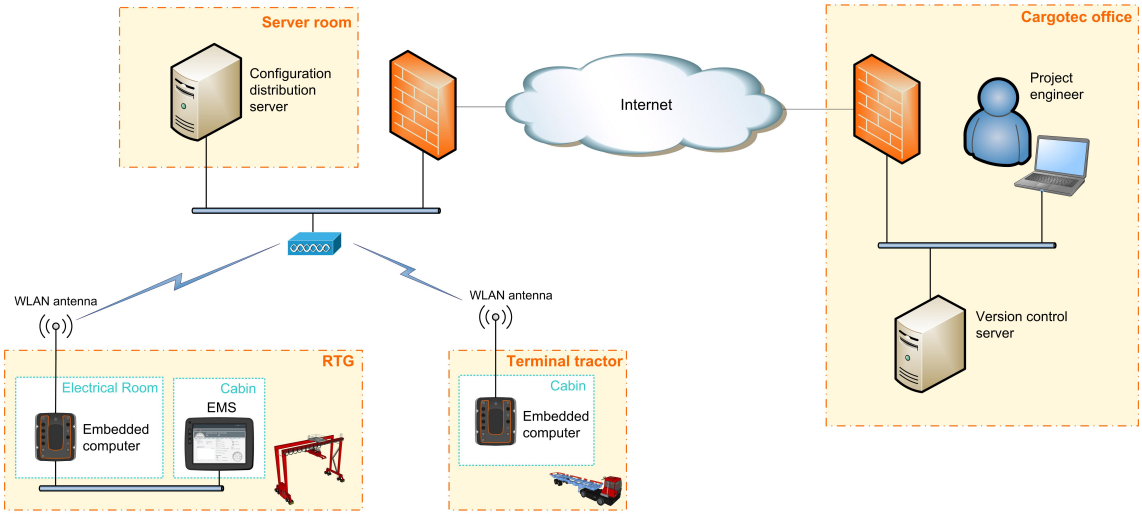


Figure 3.4: UniQ configuration network.

The peers in UniQ network will get their configurations from configuration distribution server located in the same physical network. The configuration distribution server is not a part of UniQ network so it can be used even if UniQ network fails for some reason. It should also be possible to do the initial installation of UniQ network using the configuration management tool and thus avoid unnecessary manual work. The main target for configuration distribution server is to serve Linux based embedded computers but if possible it can later be used also to configure Windows PCs.

The main direction to transfer the configuration item is from the configuration editor to the configuration distribution server. However, because also customer can edit some non-fatal configurations, it should be possible to fetch the configurations from the configuration distribution server to the configuration editor. Because the configuration distribution server does not exist in the early stage of the project, the configuration editor should not depend on it.

The configuration editor should not require internet connection, since it can be used in container terminals and it is not guaranteed that internet, or any other network, connection is available there. Also, the configuration editor should not depend on other software installed on the system and it should be capable to be run from flash driver without installing it to the computer.

# 4. CONFIGURATION MANAGEMENT OF DISTRIBUTED APPLICATIONS

Many configuration management software exists for distributed systems. This chapter will go through some of the popular configuration management software. The last section will summarise the functionalities that are implemented in every software and the methods they are implemented. It should be noticed that there are also other configuration management software available and only the ones that have passed the pre-elimination process are introduced here. The main reasons to exclude software were programming language used and inactivity of the community of associated open source project. The main focus of this chapter is to learn how configuration management is handled by popular software that are generally proven to be good. The following list shows the essential aspects that are to be taken into account while comparing software.

- Consumption of system resources, especially at the client machine.
- Software package management (install/remove/update).
- File handling (copy to/from server and remove from client).
- Management of daemonds/services on the client machine.
- Authentication of the server and the clients.
- Encryption of data transport between the clients and the server.
- Usability.

## 4.1 Local ConFiGuration system

Local ConFiGuration system (LCFG) was originally developed at University of Edinburgh around 1993. Today, it still has an active community with weekly releases. LCFG was originally developed under Solaris but is ported to Linux as Solaris version is not supported anymore. LCFG is not designed to be a monitoring system but it can collect basic information from the clients, such as are the new settings adapted correctly and if there are any errors related to them. The architecture of LCFG is shown in Figure 4.1. [2]

LCFG does not offer a configuration distribution channel. It is normally handled by an external webserver. LCFG server can automatically create an access control file and an authorization file that are compatible with Apache web server. In addition also HTTPS protocol can be used instead of HTTP. The documentation of LCFG does not mention if the LCFG client checks the validity of the SSL certificate of the server. Furthermore, LCFG uses UDP packets for communication between the server and the client. By using the UDP packets it is possibly to cause a denial of service (DOS) attack. [2]

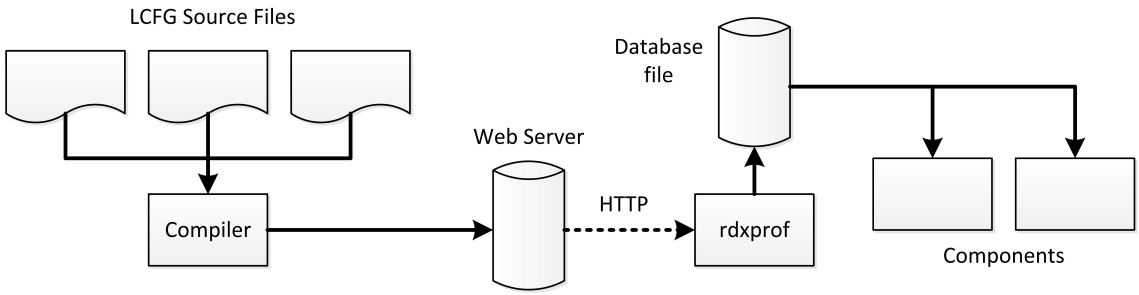


Figure 4.1: LCFG Architecture.

The most significant steps in LCFG workflow are the following [3]:

- Configuration of the entire system is described in source files. *Source files* are written in LCFG specific language. Source files consist of *resources* which are key/value pairs describing configuration parameters. Source file can include other source files, allowing easy structuring of configuration information.
- Source files are compiled into *profile files*. LCFG uses C preprocessor and its own compiler to produce the profile files. C preprocessor allows using macros in source files to ease writing. One profile file corresponds to one machine and contains all the configuration information for it. Profile files are in XML format and published on a web server.
- Client machines retrieve the profile file for the web server and stores it locally.
- Client machine's component scripts can read configuration parameters from profile file and use them to create necessary configuration files and notifies associated daemons.

LCFG supports software package management, daemon management as well as file system management. Files can be edited row by row or as a whole file. LCFG does offer a simple graphical editor for editing the source files. [9]

LCFG's language does not support an easy way for creating a list of items, each containing number of attributes. C preprocessor can also cause some problems if

source files contain C comment characters. Additional features can be added by writing a new component in Perl. [2]

## 4.2 CFEngine

The development of CFEngine started in 1993 by Mark Burgess at Oslo University. Today CFEngine is developed by a company named CFEngine AS. The current version is 3. CFEngine is a vailable both, as an open source license and a commercial version. The main differences between the open source and commercial version are that while the commercial version has a better graphical reporting system of the clients' state and native support for Windows operating systems, the open source version has only Linux support and is lisenced under GPL version 3. [7]

CFEngine is designed to be usable in both mobile and embedded devices. It is lightweight, written in C, does not have many dependencies, and aims at reducing unnecessary network usage. CFEngine clients can even continue working offline, but of course cannot then receive new information from the server. However, offline usability is important, especially with mobile devices which use unreliable networks and can often be offline for long times. [7]

CFEngine clients and the server use private protocol that is based on OpenSSH for communication. CFEngine uses RSA 2048 public key encryption for authentication. Commercial version can also encrypt data transmission using AES 256 with 256 bit random key. The CFEngine server can also be configured to allow only clients from certain IP range to create connection. [6]

CFEngine uses its own knowledge-oriented language to describe the desired state of the system. A single introduction is know as a *promise*. CFEngine offers containers called *bundles* for creating modular parts. Bundles are collections of promises and can be independent or dependent from the other bundles. The whole configuration, including all the promises and bundles, is known as the *policy*. The policy is stored on the server and individual clients pull the new policy from the server at regular intervals. The client will fetch the whole policy and determine which promises it has to fulfil. While it is not possible to push the policy into the client, it is possible to request the client to fetch the new policy from the server. A simple policy is shown in Listing 4.1. [7]

Listing 4.1 shows a simple policy that makes sure that packages *Apache2* and *Php5* are installed into the client whose IP address is 192.168.0.10. The installation will be done using *Yum* package manager. In the example, there are three different promise types, vars, classes, and packages. Vars are variables, and packages are the software packages to be controlled. Classes are used for grouping clients, so that different promises can be applied to different types of clients. Classes are evaluated to boolean values to determinate if the given promises are for the client in question.

```
1  body common control
2  {
3  bundlesequence => { "packages" };
4  inputs => { "cfengine_stdlib.cf" };
5  }
6
7  bundle agent packages
8  {
9  vars:
10  "match_package" slist => {"apache2", "php5" };
11
12 classes:
13  "server" expression => iprange("192.168.0.10");
14
15 packages:
16  server::
17   "$(match_package)"
18      package_policy => "add",
19      package_method => yum;
20 }
```

Listing 4.1: Example of CFEngine policy.

The *cfengine_stdlib.cf* on line 4 is CFEngine's standard library, providing some often used bundles. [7]. On line 10 a variable named *match_package* is defined. The variable is a string list, containing two strings. Class named *server* is defined on line 13. Server class is evaluated as true if client's IP address is 192.168.0.10. On lines 16 to 19 software packages listed in variable match_package are installed to the clients where class server is evaluated as true.

It is also possible to use CFEngine as front-end for cron to run certain jobs on a periodic basis. CFEngine allows complicated statements in order to define the time intervals. This interval definition can also contain conditional statements. For example, a job's interval can vary depending on whether it is morning, afternoon, or night. [7]

## 4.3 Puppet

Puppet is a cross-platform configuration management software developed by Puppet Labs. Puppet supports multiple Unix and Linux platforms, as well as Microsoft Windows, although support for Windows is limited when compared to the other operating systems. Puppet hides the underlying platform so that the same configuration settings can be used in different platforms without the need to rewrite them. Puppet is available via both open source and commercial version. The open source version is licensed under the Apache 2.0 license. Puppet has good online documentation

and an active community to provide new modules. Figure 4.2 presents the workflow of the Puppet system. [21]

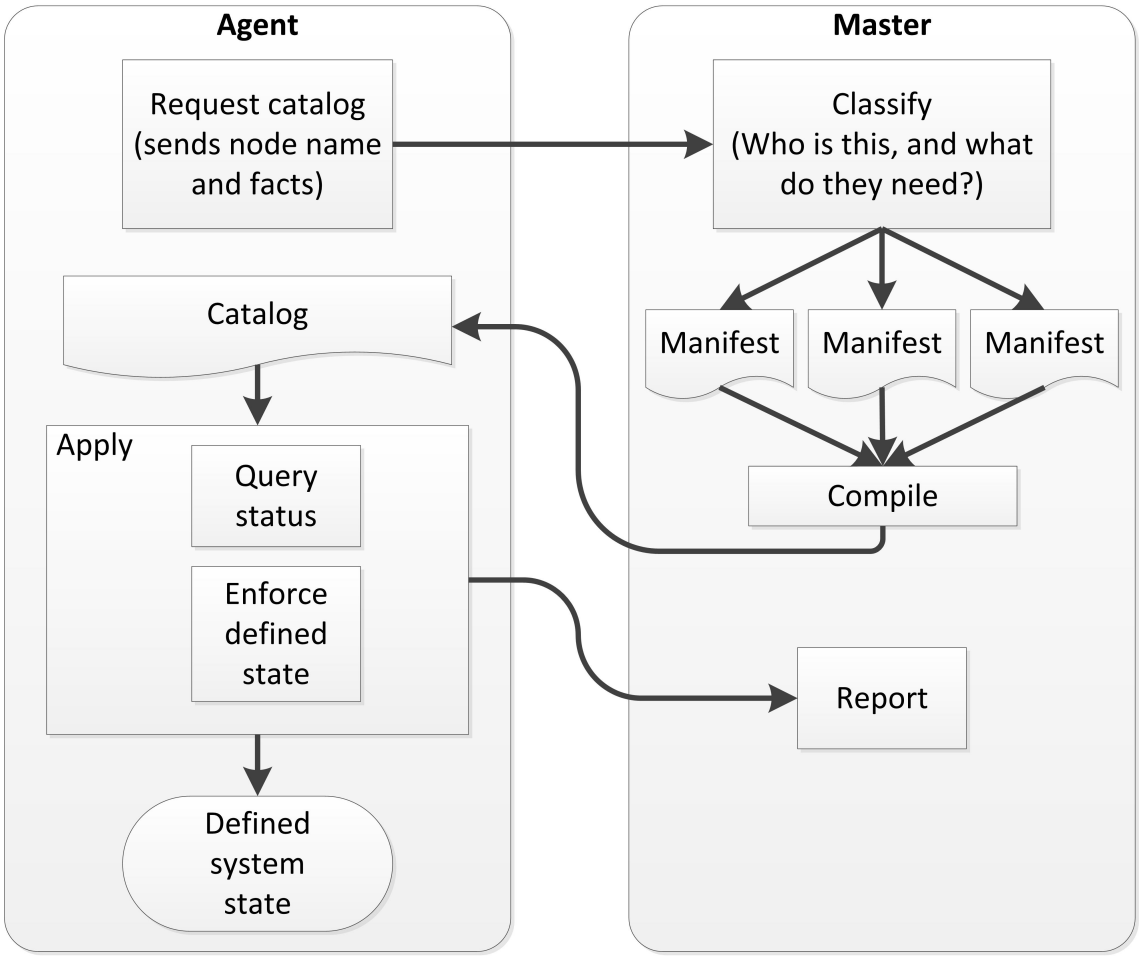


Figure 4.2: Puppet workflow.

Puppet is normally used in a server-client environment. The server is known as *master* and the clients as *agent nodes*. The administrator uses Puppet's own declarative language to write *manifest* files. Manifests contain *resources* which describe a state of a single configuration item. A configuration item can be a file, software package, a running service, or something similar. Manifests are kept in the master and resources are shipped to nodes in a *catalog* file. Puppet then compiles manifest files into a single catalog file after the node requests its configurations from the master. Puppet uses *facts* to customize manifests for the node. Facts are information about the node, such as the operating system, IP address, and hostname. After the node gets the catalog, the node applies it by using *providers*, which are platform specific implementations of resources. Listing 4.2 presents a very simple manifest file. [21]

In Listing 4.2 three resources are declared. The first resource type, declared in line 1, is a software package, that checks to whether the software in question is installed

```
1  package { 'openssh-server':
2    ensure => present,
3    before => File['/etc/ssh/sshd_config'],
4  }
5
6  file { '/etc/ssh/sshd_config':
7    ensure => file,
8    mode => 600,
9    source => '/root/learning-manifests/sshd_config',
10 }
11
12 service { 'sshd':
13   ensure => running,
14   subscribe => File['/etc/ssh/sshd_config'],
15 }
```

Listing 4.2: Example of Puppet resource

in the system and, if not, installs it. The installation must take place before the second resource, declared in line 6, is applied. The second resource is a settings file for the installed software. It ensures that the file exists and then sets its privileges and content. In line 12 the third resource guarantees that the installed service is running and is applied every time the settings file changes.

## 4.4 Ansible

Ansible is developed by Michael DeHaan. The project was published in February 2012. Ansible server is written in Python and licensed under GPL version 3. The latest version is 0.8, released in October 2012. Even though Ansible is compatively new, it already supports wide range of features. Ansible takes somewhat different approach to the configuration management than the other introduced software. It does not require any agent on the client machine, only SSH connection is required between the server and the client. Architecture of Ansible is presented in Figure 4.3. [8]

Instead of requiring agent software running on the remote machine Ansible tranfers the script or software to the remote machine when they are needed. The script or software is called *module*. After the module is transferred to the remote machine Ansible runs it with given arguments. After module is finished Ansible invokes possible callback plugins on the server side. Callback plugins can create log files, send emails, or do something else. Afterwards Ansible does not need the module on the remote machine anymore, it will delete the module. [8]

Ansible does not set any limitation for the programming language used for writing the modules, only the client machine can set limitation for the language. For example

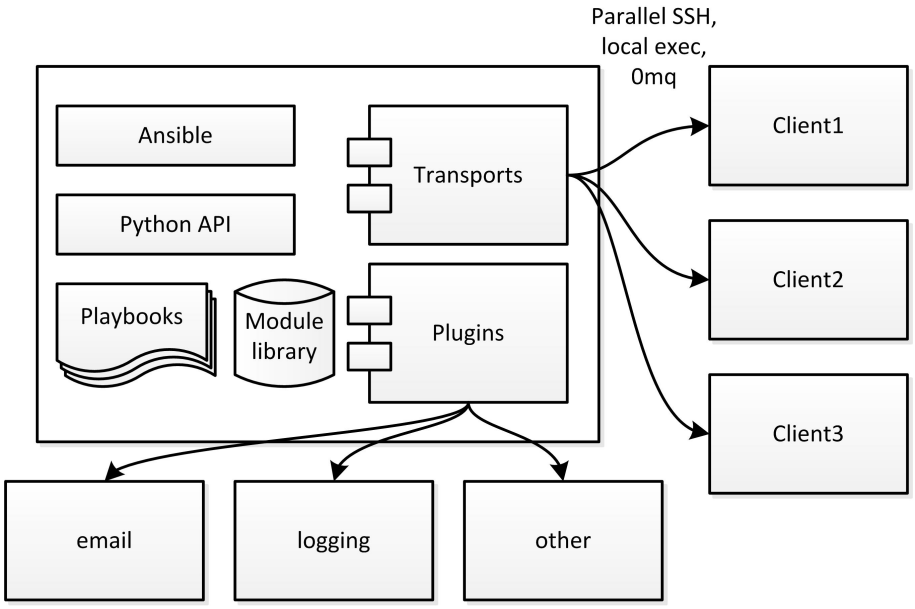


Figure 4.3: Ansible architecture.

if remote machine does not have Python, Ansible cannot run modules written in Python on it. The only requirement for module is that if it has any output, it must be printed to the standard output in JSON format. All the modules shipped with Ansible are written in Python and therefore require that Python is installed into the remote machine. Execption to this is module called Raw, which can be used to execute SSH commands on the remote machine even if there is no Python installed. [8]

By default Ansible uses Paramiko (SSH2 module for Python) to connect to the remote machines. Ansible also supports native SSH, local execution and fireball connection. In fireball connection mode Ansible launches a temporary ØMQ daemon, which by default lives 30 minutes. Fireball mode requires that its dependency Python modules are installed on the remote machine. Also other connection modes can be added to tha Ansible via connection plugins. [8]

To describe the wanted state of the remote machine Ansible uses *YAML* (YAML Ain't Markup Language). Description files are called *playbooks*. A simple playbook with one *play* is shown in Figure 4.3. Every playbook consist of one or more plays which are list of *tasks* to perform. A play defines the remote host(s) it will effect and what remote user to complete the tasks as. A task is a call to an Ansible module. The modules are executed in the remote host and they interact with the system. Modules can be written in any programming language. [8]

In Listing 4.3 the play is targeted to machines belonging in group called webservers. The host groups are defined in other file, that file is not covered here since it is only a simple list of groups and hosts. On the third and fourth lines the variables

```
1  - hosts: webservers
2    vars:
3      http_port: 80
4      max_clients: 200
5    user: root
6    tasks:
7    - name: ensure apache is at the latest version
8      action: yum pkg=httpd state=latest
9    - name: write the apache config file
10     action: template src=/srv/httpd.j2 dest=/etc/httpd.conf
11     notify:
12     - restart apache
13   - name: ensure apache is running
14     action: service name=httpd state=started
15   handlers:
16     - name: restart apache
17       action: service name=httpd state=restarted
```

Listing 4.3: Example of Ansible playbook

are defined. The remote user to be used to run the tasks is root, as defined on line 5. It would also be possible to log in as another user and then which to root user or use sudo command. There are three tasks in the tasks list. Tasks ensures that remote machine has the latest version of Apache, Apache has correct configuration file and that Apache is running. If Ansible updates the configuration file, Apache will be restarted. If configuration file's template contains variables, they will be replaced with equivalent values. Ansible uses Jinja2 templating language in templates. [8]

For configuration management of small embedded devices, Ansible could be a good choice. It can be used to configure any device that supports SSH connection. Due to the lack of agent software Ansible does not require cross-compiling the software for the client machines. The main challenge with Ansible is how it can handle unreliable networks with low bandwidth, which is often the case with mobile devices. Ansible itself is not designed for that so it must be take into account while writing modules. [8]

## 4.5 Summary

Every introduced CM software use idempotence language to describe the desired state of the system. Idempotence language makes it easier for system administrator to apply configurations, since it does not matter how many times the rules are applied, the result will always be the same. This makes it easier to recover after a failure or from unknown state of the system. Even though every software has similar type of needs for the language, they all use different language. Different languages make it harder to use another software after selecting one.

Besides using idempotence languages most of the software also share same kind of structure. The structure is shown in Figure 4.4. On the server side there is a manager software and every client machine runs an agent software. The agent software uses modules to interact with the underlying system. The manager and agent software are portable and only modules have to be rewritten for new platform. A common interface for modules makes it easy to extend the functionality of CM software and allows to use same syntax to describe operations for every module. The only exception is Ansible, which does not require agent software, but uses modules directly to interact with the client machine.

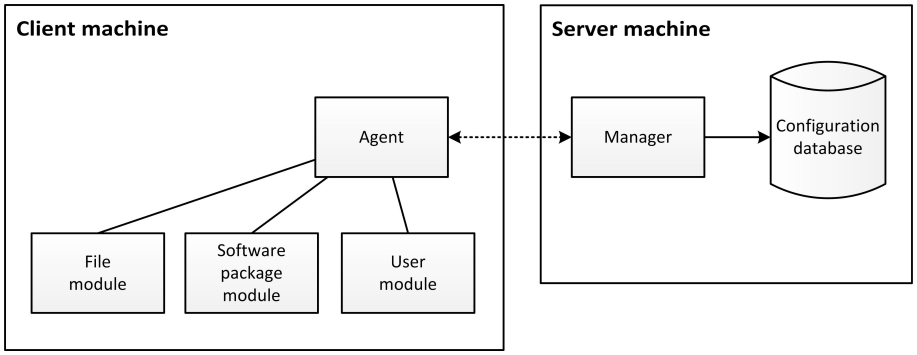


Figure 4.4: Architecture commonly used in configuration management systems.

None of the introduced software handle version control. Therefore external version control software is needed to handle change management of configuration database. Software does not restrict which version controls can be used, if any.

# 5. CONFIGURATION EDITOR, UNICONF

UniConf is a modular configuration editor designed for UniQ system configuration. It aims at helping engineer to organize and understand structure of the project and relations of peers and configuration items. Besides UniQ configuration it also supports editing configurations of other systems, often needed with UniQ. UniConf is meant to be used by project engineer in customer projects to maintain and create configuration files. UniConf is also usable in testing and product development. The most common location of UniConf in the configuration management network is marked with red circle in Figure 5.1.

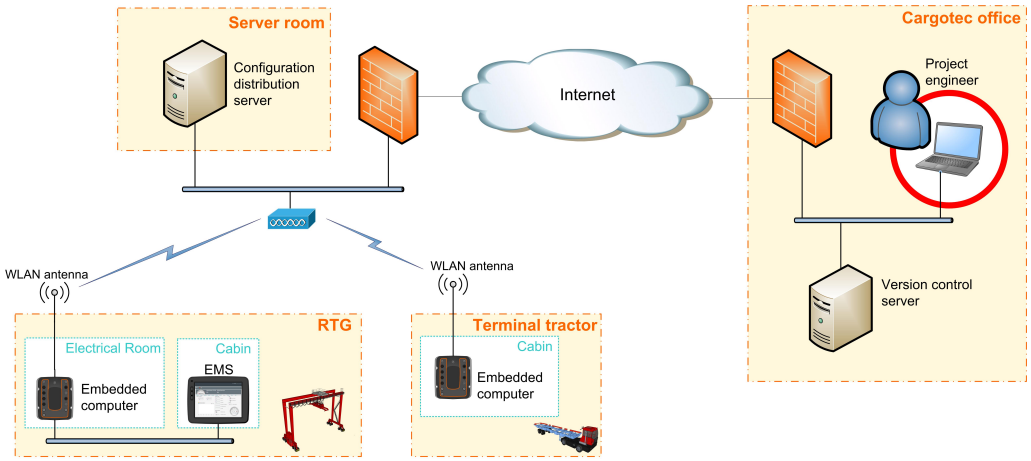


Figure 5.1: The most common location of UniConf.

UniConf is written using the Qt Framework. Qt offers powerful cross-platform support for GUI and XML handling among others. Qt also has built-in support for plugins and cross-platform file system handling capabilities. Qt is licensed under LGPL version 2.1. [20]

## 5.1 Architecture and design

UniConf introduces baselines to ease the creation of a new project. Baseline consists of two parts, the template and the plugins. The template describes the data of configuration files and the plugins the functionality of the configuration files as well as the functionality of the configuration item groups (`CIGroup`). A new baseline can change the behaviour of UniConf as well as add more ready made peer types

and configuration files. This makes it possible to add new features and support for future releases of UniQ without breaking the backward compatibility.

The Figure 5.2 shows the hierarchy of the items in UniConf. A project consist of peers, which are grouped by their type. Every peer has its own configuration files but those files can be identical for all the peers in a certain group. For example datasourcemap.xml is the same for every peer in a group and tagmap.xml is the same for every peer in the whole project. For this reason it is also convenient to edit such files as a single entity. UniConf enables to edit file only once and automatically applies the changes every peer necessary.

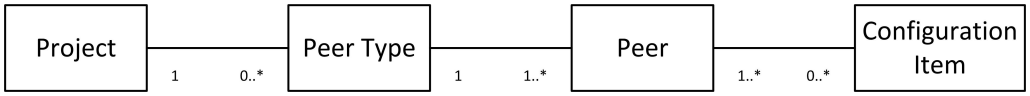


Figure 5.2: The logical structure of a project in UniConf.

Figure 5.3 shows the simplified class diagram of UniConf. The target of the design of UniConf is to keep the basic structure of the project in the main program to simplify the development of the plugins. All the data from the main program is exposed to the plugins via the standard model view architecture of Qt. This simplifies the development of new plugins, since view classes provided by Qt can be used directly to view the data to the user. Together with the main program also some convenience classes were made for plugin developers, including filter models and a table view with Microsoft Excel type of functionality.

The main program also offers the needed search functionality to the configuration data. The data of configuration file can be searched and edited not only by the configuration file plugin that owns the data but also other configuration file plugins. The benefit of this is that configuration files can be merged into the same view, if it makes editing simpler for the user. A drawback is that badly behaving configuration file plugins can break also other configuration files. However, this should not be a major problem since all the plugins are developed by a trusted party.

The main program handles saving of the data and also undo/redo functionality. Currently all the data is saved in a single XML file. The benefit of this is easy data recovery after data corruption. The configuration data is also easily readable even if UniConf software is not available for some reason. Since customers can require support after decades it is important to be prepared for this situation.

## 5.2 Plugins

Plugins work as factories to create the functional part of `CIGroup` and CI classes. The main program defines the interface the plugins must implement. For the `CIPlugin` the main functionalities are creating the view, validating, and exporting

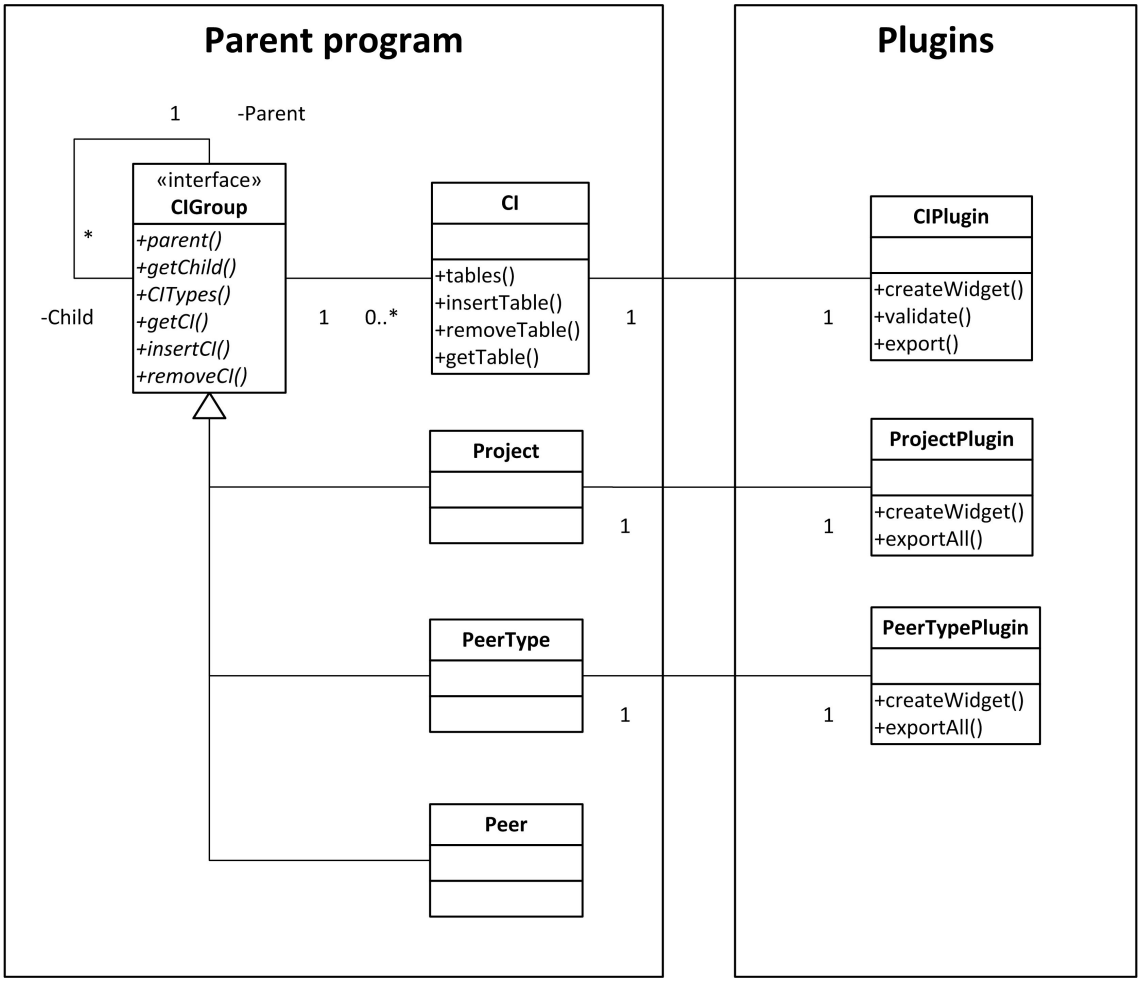


Figure 5.3: Simplified class diagram of UniConf.

the configuration file. The view is a graphical representation of the configuration file, usually a table of data. `Validate` function validates the data of configuration item and `export` function returns the file content as a string. `ProjectPlugins` and `PeerTypePlugins` also need to create the view and export all the configuration items belongin to the `CIGroup`.

The `export` function can also do some additional work, such as calculating checksums and packing the configuration files into an installation package. Export functionality is designed to be handled by `peerType`. Exporting cannot be handled solely by a configuration file plugin because exporting may need to collect multiple configuration files in the same package to create the correct output.

Plugin architecture allows various modifications of configure files and peer types, without changing the main program. With plugin architecture, it is easy to add, remove, or modify a single configuration file without an understanding of the main program. Plugin structure also allows easy outsourcing of part of the programming work without exposing the rest of the code.

The plugins are loaded at runtime from a directory specified by the user, so

new baselines can be added without reinstalling the whole software. For now new baselines are distributed via a new installation package of the software but in the future some automatic tool can be used for that.

# 6. USING UNICONF

This chapter will explain the basic functionalities of UniConf configuration editor. The following examples are based on the current main program and the plugins made in this thesis. Since the developing of new plugins is not controlled tightly, it can not be guaranteed that all the future plugins follow the same logic. However, it is recommended for plugin developers to follow same logic as example plugins.

## 6.1 The main layout

As the Figure 6.1 shows, the main window of UniConf is divided into four main items. Item 1 contains menu and tool bar which includes the actions provided by the main program. If configuration items provide some additional actions, they will be located in configuration item's view which is located in item 2.

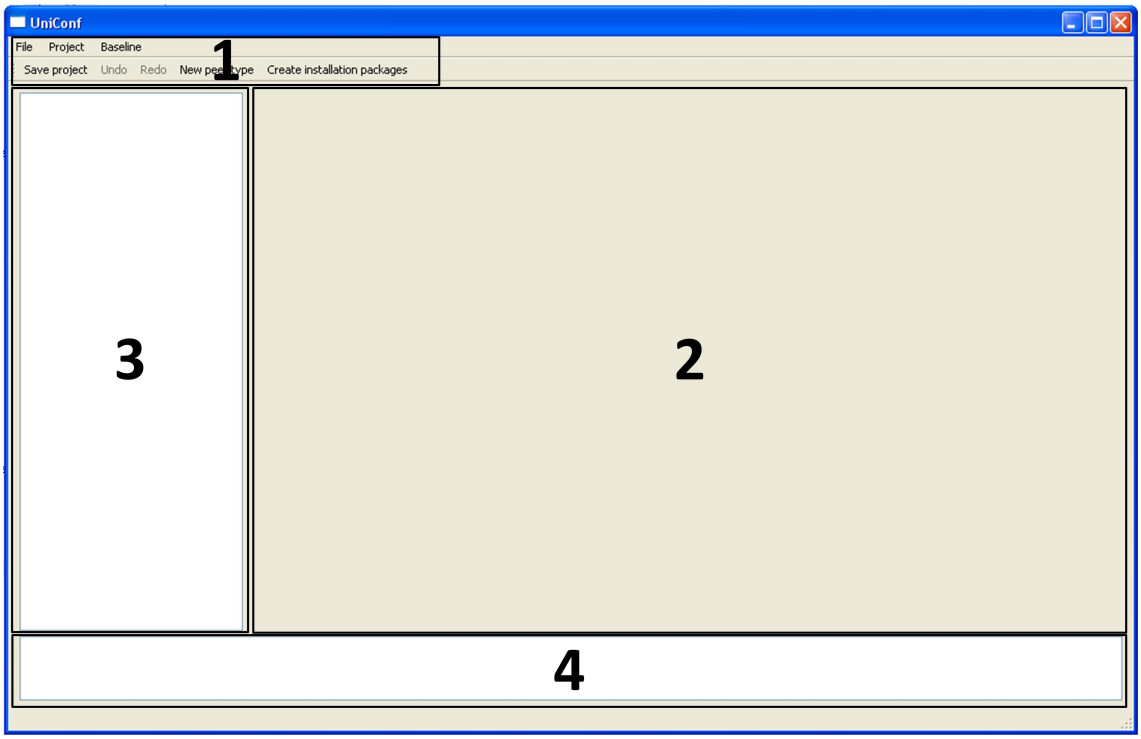


Figure 6.1: Main window of UniConf.

Item 3 is a tree view, containing all the configuration items arranged according to the peer types they belong to. From the tree view, the user can select which configuration item's view is displayed. Tree view has three levels; project, peer

type, and configuration item. Configuration item can be located under the project directly and under peer type. Modification done to a configuration item under the project will have effect to the whole project, and modifications done to configuration items under the peer type has effect only to that peer type. This is only a rough rule, as the configuration item plugin can decide the exact behaviour. Item 4 is message box which will show error messages and warnings created by the plugins.

## 6.2 Creating a new project

Creating a new project is very simple. First, the user selects to create a new project from the menu and then choose the baseline for the project. Baseline defines which configuration items and peer types are available and can also affect to the behaviour of the configuration items. The steps to create a new project are shown in Figure 6.2. After the creation, the new project will open in UniConf without any peers but the project may contain global configuration items, defined by the baseline.

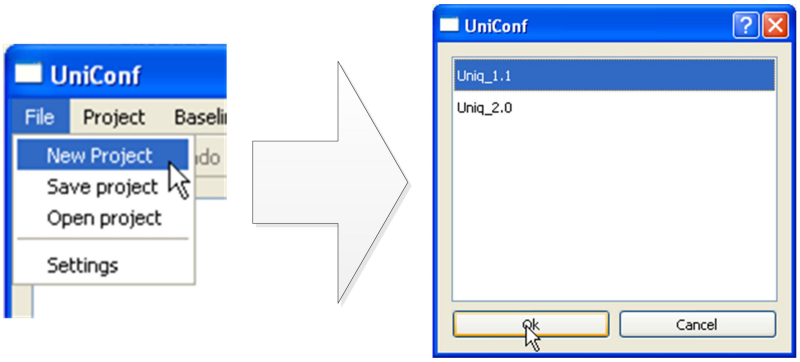


Figure 6.2: Creating a new project.

After creating a new project the user should also set correct information for the project. The user should at least change the name of the project so that projects are easy to distinguish from each other. Also filling the name of the customer and the name of the site is strongly recommended. There is also a text field reserved for notes related to the project, such as what has been done, what needs to be done, and why something is done in the way it is done.

## 6.3 Adding a new peer type

A new peer type can be added to the open project by clicking button "Add peer type" and selecting the wanted type. A new peer group will appear to the project's tree view. Peer type will have the configuration files according to the baseline. If all the configuration files are not needed in the project in question, the unnecessary file can be unchecked from file list on peer type's view.

By default, the name of the peer type is the same as its type, and, therefore, the name should be changed to be more descriptive. For example, peer type CHE should be named after the type of the CHE such as SHC, CSC, ESC. It is not possible to add two peer type with same name, so an earlier added peer name must be changed before adding a new peer type of same type.

## 6.4 Importing configuration item from csv file

Certain configuration files allow user to import them from csv file. Figure 6.3 shows the steps for importing a data source map. Importing can be done for configuration files presented as a table in UniConf.

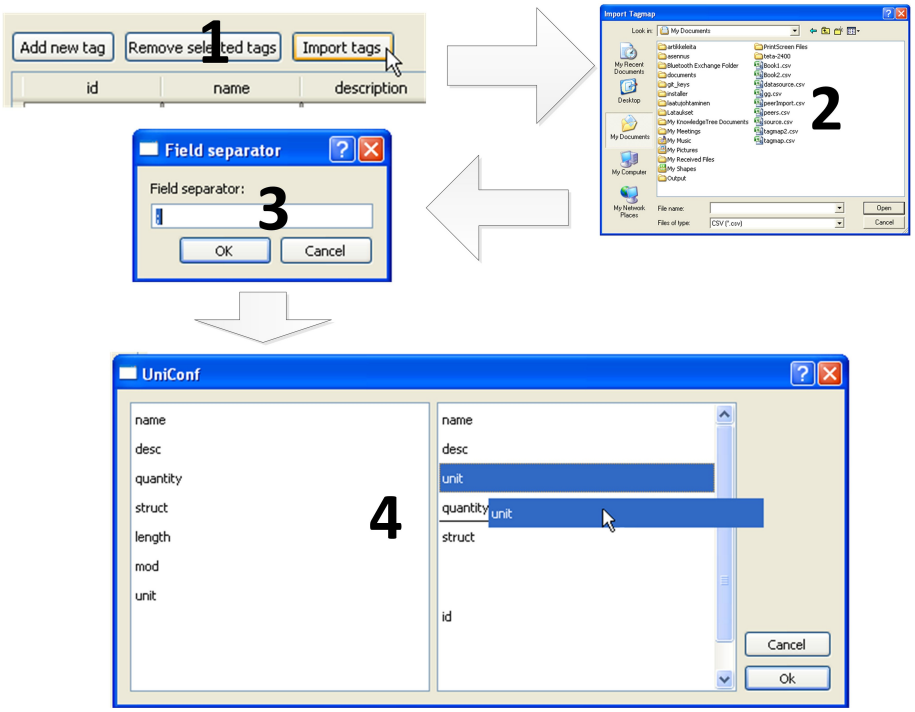


Figure 6.3: Importing a file.

Importing begins by clicking "import" button, as shown in step 1 in Figure 6.3. In step 2 user selects the csv file using normal file dialog. In step 3 user must input correct field separation character. The default value is ";" which is the same character than Microsoft Excel uses in Finnish localization. Finally, in step 4 UniConf asks the user to select which columns will be imported. On the left side UniConf shows the available columns in the csv file and right side columns that can be imported. The item on the right list can be dragged with mouse to match the column in csv file. If csv file contains columns that are not to import a blank item can be drag for their pair. Only the columns that have a pair on both lists will be imported to UniConf.

## 6.5 Editing commconf

The commconf file is very different from all the other files. It is presented as a drawing expressing the peers, connections and channels to other peers. Peers' properties can be edited both in drawing mode and in table mode. It is possible to change to table model by selecting "table" tab. Gateways and connection can be added and removed only in drawing mode, but the values of existing ones can be edited in table model. Channels can be edited only in drawing mode. Figure 6.4 shows a commconf file in a very simple project.

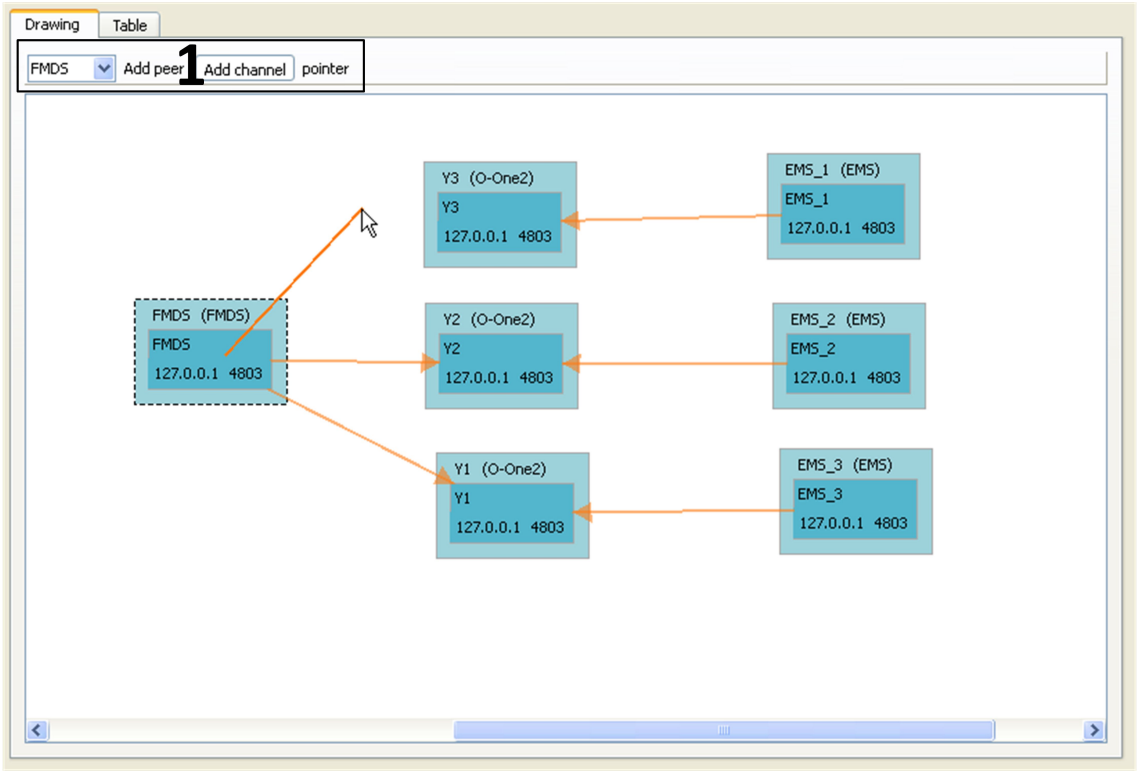


Figure 6.4: Editing commconf in drawing mode.

By default all the peers are placed on top of each others in the drawing, but they can be moved by dragging them with the cursor. In Figure 6.4 the commconf file contains three CHEs, EMS for them, and a FMDS server. Every peer has one connection but more can be added via context menu. Context menu can be opened by clicking the right mouse button on the top of the peer box. Context menu actions will vary depending on the placement of the cursor. Context menu allows user to add and remove connections and gateways. Channels between peers can be added by selection "Add channel" from tool bar, located in item 1 in Figure 6.4.

## 6.6 Editing textfile

Figure 6.5 shows the view of the text file. View is divided into three items, the first item contains the configuration data for peers, item 2 the text file itself. Item 3 contains buttons needed if XML schema validation is used. The text file shown in the view works as a template containing placeholders to be replaced with the values given in the upper table during the export. The placeholder pattern is "${NAME}", where "NAME" is the name of the placeholder. Placeholders are automatically recognized, highlighted in red, and corresponding column added to the table. A file may contain the same placeholder multiple times, in which case only one column is added in the table and every instance of the placeholder is substitute with that value. Since UniConf does not a have way to know the intended type of the placeholders, it cannot validate the values. However, UniConf will give a warning if the value is empty.

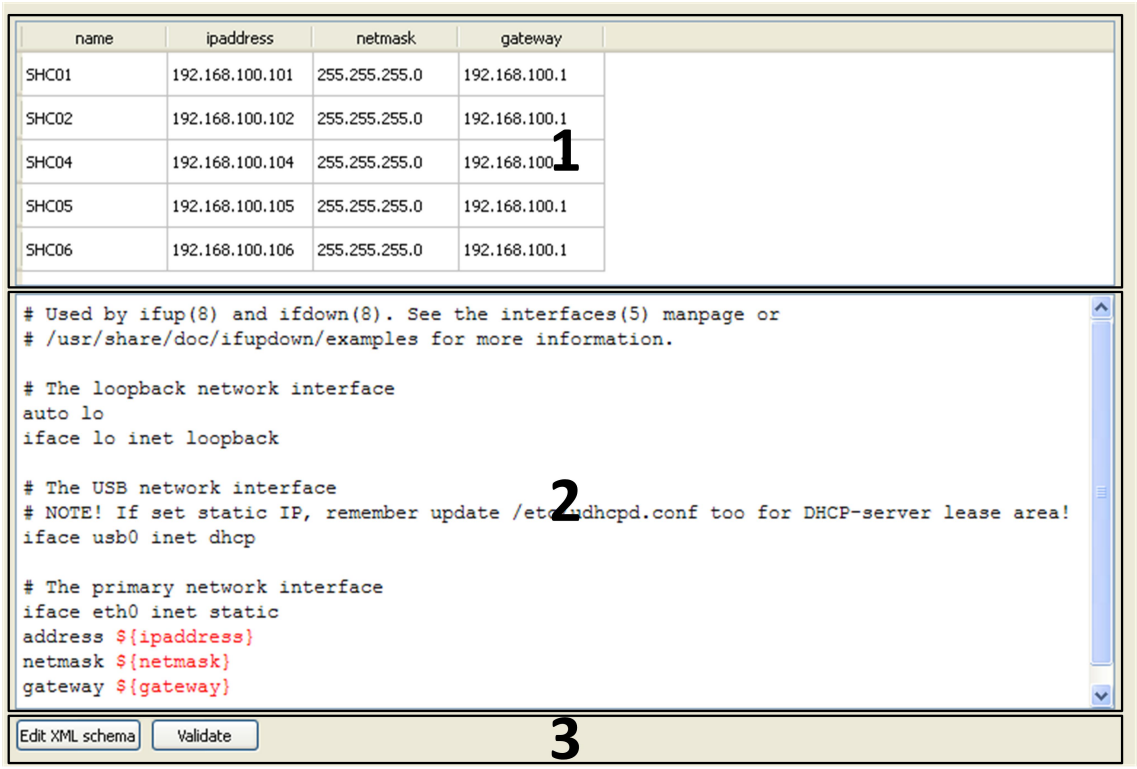


Figure 6.5: Example of generic textfile.

If the text contains XML tags, they will be highlighted. Also XML Schema can be defined for the file by clicking the "Edit XML schema" button located in item 3. If the schema is defined, then it will be used for validate the file during the export. Validation can be triggered also manually by clicking the "Validate" button in the item 3. Validation will show the first error found from the file in item 3.

## 6.7 Creating installation packages

Configuration items can be exported from UniConf in a format suitable to be used in the final product. Before exporting the files, user should check that correct files are selected for export and that file paths are correct. Figure 6.5 shows the "Files" tab of peer type, which shows all the file that can be exported. All possible files for the peer in question are listed in item 1. The files are defined by the baseline. File can be selected to be exported by checking the check box in item 2. Path for the exported file can be set in item 3. If peer requires application package, it can be selected in item 4. Application package is not copied into the project. Instead only the path to the package is saved. During the export process the application package will be copied to the user selected export directory. If installation package is needed, it can be created by selecting it in item 5 and setting a name for it. If installation package is selected all the files are placed there. The inner structure of the installation package is defined by the baseline.

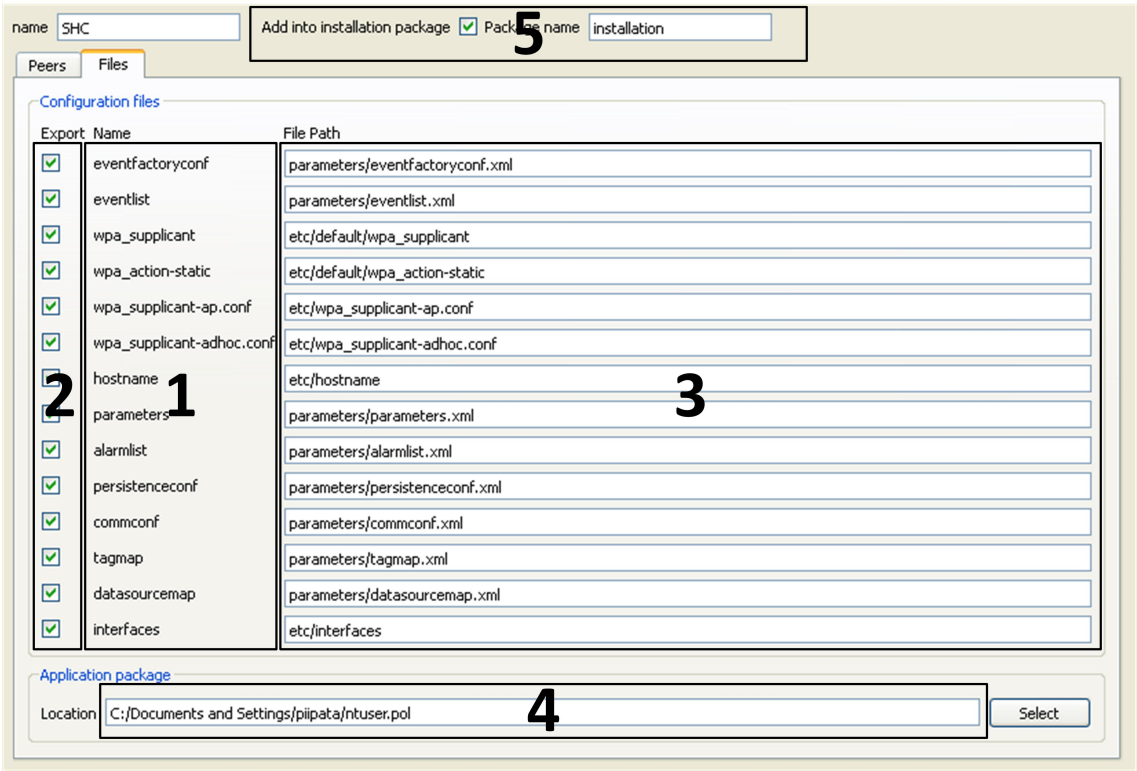


Figure 6.6: Exporting configuration items

Since file name and path can vary depending on the peer name and type it is possible to use placeholders in the file paths. Valid placeholders are listed and described in Table 6.1 File path is the path where the file is placed in export. Path must be given in Unix-style, meaning that file separator is "/", path can contain spaces. File name must be included in the path, the name can be different from the

| Placeholder's name | Description |
|---|---|
| ${projectName } | Name of the project |
| ${peerType} | Peer's type (E-One, SMC, CSC, etc.) |
| ${peerName} | Name of the peer |
| ${MAC} | MAC address of the CCU, applicable only for the CCU peers. |
| ${SN} | Serial number of the CCU, applicable only for the CCU peers. |

Table 6.1: Possible placeholders to be use in file paths.

name shown in item 1.

UniConf will automatically validate the configuration items before exporting them. Errors will not prevent exporting, but error messages are shown in the message area. Errors are allowed since during the project it is convenient to test the configurations even if they are not completely ready yet.

## 6.8 Creating a new template

UniConf has baseline menu in the main windows menu bar. Menu allows user to add and remove configuration files and peer types of the currently open project. The project can be saved as a baseline from baseline menu by selection "save as baseline". Saving the project as baseline will create a template file containing peer types and configuration files of the current project. Created template file can be added to the baseline library by designated baseline manager. A normal project engineer should have no need to ever create a new baseline and therefore they shall ignore this menu.

# 7. CONCLUSIONS

The configuration editor developed in this thesis, the UniConf, is designed to be only a demo version and a base for development of the actual configuration editor. The demo version is used for research on how configuration files can be edited user friendly and efficiently. Also new things were learnt about CM in general that can be beneficial in further development of configuration editor and other related systems.

## 7.1 Further development of UniConf

Currently there are two main concerns related with UniConf. The bigger issue is how to integrate UniConf with the chosen configuration distribution server. Since the server software has not yet been chosen, the detailed integration design is impossible to do. Only the basic outlines can be given here.

UniConf should be able to read the configuration files from the format that configuration distribution server is using for storing them. This should not be hard since all the studied server software stored the files using normal file systems, not database. The files stored in the configuration distribution server should be organized in the same kind of hierarchy as is used in UniConf. Then it would be easy for UniConf to read the existing configuration files from the server and restore the project back to UniConf.

The minor issue is the file format that UniConf uses internally. Currently it is XML file, that stores all the configuration data in a single file mimicking a database. Until now it has worked well; however, if the size of the project increases significantly then XML can turn to be too slow and memory consuming. Instead of XML it would be possible to use ready-made relation database to store the data. Other option could be to use archive type of file, containing the same directory structure as the configuration distribution server uses.

The integration with configuration distribution server is essential, since otherwise the configuration information would be stored and edited in multiple locations simultaneously causing incompatible configurations.

## 7.2 Integration of the configuration management processes

UniQ customer project has multiple interfaces to other CIs, such as product developing projects and customer. Product developing projects are typically done by

Cargotec or by its subcontractor, but it is possible that one or more of CIs are developed by some other company not related directly with Cargotec. The most important interfaces are presented in Figure 7.1.

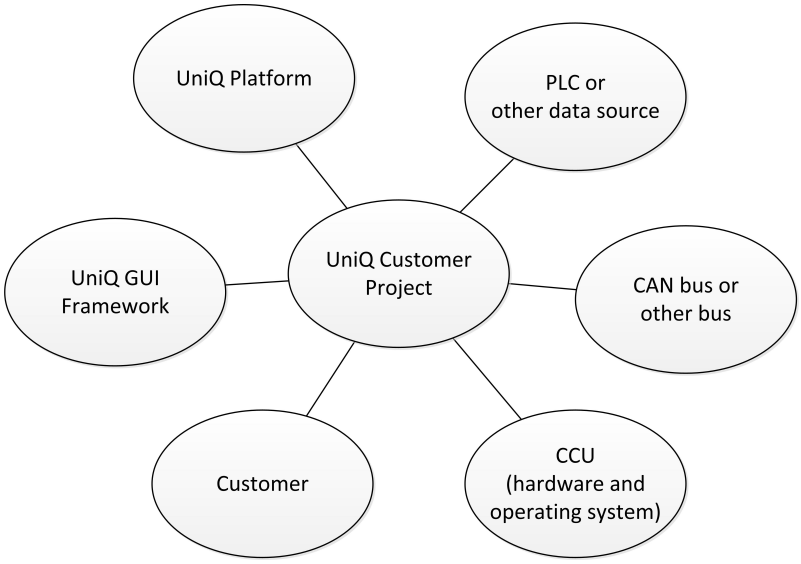


Figure 7.1: UniQ customer project interfaces to other projects.

In order to achieve a comprehensive CM system, it is not enough to implement just the configuration file management and distribution of UniQ, but the process should be integrated with other CM processes in Cargotec. Configuration file management depends on multiple application development projects as well as CHE developing projects. All these should be bring together and managed under unified CM process and rules. Currently especially the interfaces between different projects are problematic, since all of them are not documented well, or at all, and information is provided by emails.

A common CM environment and tools should be chosen and implemented into use. A common set of tools would ease the personnel to learn and understand the CM process and would make information sharing between different projects easier. Also monitoring that correct practise and procedures are applied would be easier.

## 7.3 Summary

UniConf has been tested in one customer project. Although the project has not yet been finished, the output of UniConf has been tested and proven to be correct. By using UniConf, it is possible to create all the UniQ framework configuration files and installation packages for CCU peer needed in a typical customer project. Creating the configuration files using UniConf is faster and simpler than using the older tools.

# REFERENCES

[1] V. Ambriola, L. Bendix, and P. Ciancarini. The evolution of configuration management and version control. *Software Engineering Journal*, 5(6):303 –310, November 1990.

[2] P. Anderson. The complete guide to LCFG. `http://www.lcfg.org/doc/guide.pdf`. June 2005. University of Edinburgh. Internal documentation.

[3] P. Anderson and A. Scobie. LCFG: The Next Generation. `http://www.lcfg.org/doc/ukuug2002.pdf`. January 2002. UKUUG.

[4] BitMover, Inc. BitKeeper - The scalable Distributed Software Configuration Management System. `http://www.bitkeeper.com`. November 2012.

[5] Cargotec Finland Oy. UniQ Framefork Architecture Specification. December 2010. Internal documentation.

[6] CFEngine AS. CFEngine Architecture and Security, A CFEngine Special Topics Handbook. `https://cfengine.com/manuals_files/SpecialTopic_Security.pdf`. July 2012.

[7] CFEngine AS. Open source configuration management - CFEngine. `https://cfengine.com`. December 2012.

[8] M. DeHaan. Ansible - Advanced System Orchestration. `http://ansible.cc`. November 2012.

[9] C. Devlin. A GUI interface for LCFG. `http://www.lcfg.org/doc/devlin.pdf`. June 2003. University of Edinburgh. Undergraduate project report.

[10] Git project. Git. `http://git-scm.com`. November 2012.

[11] A. Hass. *Configuration Management Principles and Practice*. Agile Software Development Series. Prentice Hall, 2003.

[12] Institute of Electrical and Electronics Engineers. IEEE Standard for Configuration Management in Systems and Software Engineering. *IEEE Std 828-2012 (Revision of IEEE Std 828-2005)*, March 2012.

[13] International Organization for Standardization. Systems and software engineering – Vocabulary. *ISO 24765:2010*, 2010.

[14] J. Keyes. *Software Configuration Management*. Taylor & Francis Group, 2004.

[15] R. Krikhaar, W. Mosterman, N. Veerman, and C. Verhoef. Enabling system evolution through configuration management on the hardware/software boundary. *Systems Engineering*, 12(3):233–264, 2009.

[16] P. Kylliäinen. Engineering Manager, Projects. Cargotec Finland Oy. Interview. November 2012.

[17] A. Lehtonen. Data distribution protocol for container handling equipment interface. Master's thesis, Tampere University of Technology, November 2009.

[18] Mercurial community. Mercurial SCM. `http://mercurial.selenic.com/`. November 2012.

[19] J. Niva. Project Engineer. Cargotec Finland Oy. Interview. November 2012.

[20] Nokia Oyj. Qt - cross-platform application and UI framework. `http://qt.nokia.com`. August 2012.

[21] Puppet Labs. Puppet Labs: IT Automation Software for System Administrators. `http://puppetlabs.com`. October 2012.

[22] M. Rochkind. The source code control system. *Software Engineering, IEEE Transactions on*, SE-1(4):364 –370, dec. 1975.

[23] T. Roponen, A. Roponen. Avoimen lähdekoodin versionhallintaohjelmat. Master's thesis, University of Jyväskylä, November 2007.

[24] F. Watts. *Engineering Documentation Control Handbook: Configuration Management and Product Lifecycle Management.* William Andrew Publishing. Elsevier Science, 2011.