CHALMERS
UNIVERSITY OF TECHNOLOGY

Visualization of Configurations

Degree Project, BSc in Computer Engineering

Magnus Dannerstedt
Niklas Helgegren

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2014

Visualization of Configurations

Magnus Dannerstedt, Niklas Helgegren

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Tel: +46-(0)31-772 1000
Fax: +46-(0)31-772 3663

Department of Computer Science and Engineering
Gothenburg, 2014

Visualization of Configurations

Abstract
There exist multiple configurations for an *RBS (Radio Base Station)* in a telecommunication network. The combinations between equipment configurations and functional configurations are numerous. Due to this mapping between functional and equipment the validity of a configuration is complex. When developing or deploying an RBS it is important to understand the configuration.
This degree project investigates the ability to visualize this domain with the environment *Intentional Domain Workbench*. The main method of doing this is creating two kind of type catalogues for equipment and functional. Then one can use the type catalogues to create configurations that then are able to map between each other.
A basic prototype in the environment was developed which lets two types of users interact with the prototype. The two types of users have different views. That is for letting the specific type of user use the program in the best way possible. The prototype developed was tested by a group of prospective users. They did see potential in the prototype and gave suggestions of improvements for the future. Future improvements of the prototype would be to increase the validation making sure it would be hard make mistakes while using the prototype, and increase usability.

Sammanfattning
Det existerar många olika konfigurationer för en *RBS(RadioBasStation)* i ett telekommunikationsnätverk. Kombinationerna mellan hårdvarukonfigurationer och funktionella konfigurationer är mångtaliga. Därför är validering av mappning mellan hårdvara och funktionalitet komplex. När man utvecklar eller placerar ut en *RBS* är det viktigt att förstå konfigurationen.
Detta examensarbete undersöker möjliheten att visualisera denna domän med miljön *Intentional Domain Workbench*. Den huvudsakliga metoden att göra detta är att skapa två sorters typkataloger för hårdvara och funktionella delar. Sedan kan man använda typkatalogerna för att skapa konfigurationer som kan användas till att mappa emellan.
En enkel prototyp utvecklades som låter två typer av använder interagera med prototypen. De två användarna har olika vyer. Detta är så att en specifik typ av användare ska kunna nyttja programmet på det bästa möjliga sättet. Prototypen testades av en grupp tilltänkta användare. De såg potential i prototypen och gav förslag om framtida förbättringar. Framtida förbättring av prototypen kan vara att öka valideringen för att minska risken för misstag och öka användarvänlighet.

Acknowledgments
We would like to thank Staffan Ehnebom, Lars Boström, Chongchi Phung and Mikael Krekola for their support and assistance to this project. We would also like to thank Mats Helander for the support and time he spent helping us with the programming and Magnus Almgren for his guidance with the report and feedback on our workflow.

Acronyms, Abbreviations and Terms

Functional	A term for the logical parts in the configuration
IDW	Intentional Domain Workbench
DSL	Domain Specific Language
RBS	Radio Base Station
View	A workbench with a specific purpose

Table of Contents

1. Introduction

1.1 Background

Ericsson is an international telecommunications company. Ericsson was founded in 1876 by Lars Magnus Ericsson [1]. Today Ericsson products exist in more than 1,000 networks in over 180 countries and at least 40% of the world’s mobile traffic passes through these networks [2].

The department where the degree project is done is focused on software for Ericsson's radio base stations.

An *RBS (Radio Base Station)* in a telecommunication network can be configured in many ways. There are numerous combinations of equipment configurations (hardware and software capabilities) and logical configurations (functional demands). Because of the complexity of an *RBS* configuration, it is sometimes hard to understand the configuration, i.e. the mapping of the logical configuration onto the equipment configuration. Furthermore, it can also be hard to understand whether a configuration is a valid. The need for this understanding is of high importance when working in development and deployment of an *RBS*. Therefore, it is interesting to investigate how to edit, validate and visualize large configurations in an efficient way. In other words, to investigate how to make efficient *domain specific languages* for editing and exploring large configurations. One level of visualization is just for manual validation. Another, more advanced, level of visualization is to support automatic validation and/or proposal(s) of configurations.

When there is just a few kinds of equipment items and functions it is realistic to document them, but when more items are added, the complexity grows [3]. To improve this, another way to show configurations is needed and providing a visualization is a good way to do it [4].

Visualization is used in wide range of subjects and for many different reasons. It can range from how a tree structure is best visualized [5] to finding errors in computer networks [6]. When you have information it is important to find a good way to visualize it [7]. If the visualization of information is represented in a rather large graph it could be good to figure out how to navigate in large graphs [8].

1.2 Purpose

The purpose of this degree project is visualizing large *RBS* configurations using a *Domain Specific Language (DSL)*. The created program should have the possibilities to show which different layers of configurations that the product has. The *DSL user* should be able to see how different parts are interconnected. Together this should make it easier for the *DSL user* to get a grasp of what the product consists of and how different parts and layers are related to each other.

1.3 Goal

The goal of this project is to develop a prototype/workbench with the following functionality. One *view* with type catalogues where functional and equipment types will be defined and placed in the catalogues of the domain experts. There will be both an equipment type catalogue and a functional type catalogue (described further in chapter 4.2).

Another *view* to be defined will be the *configuration view*. This is a *view* where the *DSL user* can create and connect instances. These instances are of the types that are defined in the *type catalogues view.* When created they are placed in a functional or equipment diagram. Both works in similar ways when a *DSL user* does the connections between equipment or functional items. This *view* will also show the mapping between functional and equipment. The *view* will show what functional instances an equipment instance may have or on which equipment instance a functional instance is deployed.

Both the *configuration view* and the *type catalogues view* should have validation controls that can suggest appropriate values for a field. This will be used when adding ports for a unit or connection. The suggested validation will also be applied when a *capacity demand* or *capability* is selected (described in chapter 4.2). The choices of which resources the domain expert can choose from will be included in the schema.

1.4 Restrictions

The environment platform that will be used is the *"Intentional Domain Workbench"* .The intention is to evaluate this *projectional editor* for visualization. No comparison with other visualization tools will take place. *Intentional Domain Workbench* is used to develop the prototype. The prototype comprises only basic validations. However, more enhanced ones are possible to add later on. The prototype at this time is limited to two layers, one equipment layer and one functional layer. The prototype will have the ability to add more layers later on with ease but for the specific version that will be used for demonstration within this project only two layers will be used.

2. Method

The method that will be used is *scrum* [9] like, where there will be a meeting in the beginning of each week with supervisors to show what has been done and plan what the focus will be until next meeting. A task that will be ongoing but also done last is to polish the code by removing unnecessary code and making it more readable.

The workflow will be to have a progress meeting each week with both supervisors on Ericsson and help from Intentional Software Corporation. Furthermore there will be additional Skype meetings with Intentional Software Corporation, for help with the coding. In each week it will be decided what will be the focus, what can be scratched or added. In addition to this every other week there will be a meeting with the Chalmers supervisor. The following phases of the project have been identified and will be described below; preparations, creating *workbenches* and evaluation.

2.1 Preparations

The first thing that is done is to learn about the problem domain. When that is done it is time to use one week to learn the platform that is going to be used, in this case the *Intentional Domain Workbench.* Then it is time to define the schema for the specific domain. As a scrum-like method is used, there will not be much further detailed planning for each week; instead there is going to be a scrum-board with backlog where we can choose what to focus on each week.

2.2 Creating Workbenches

After the schema is defined the plan is to develop a *workbench* where the domain expert can edit and see the different unit types, connection types and port types for both equipment and functional. Other tasks that are planned are the *configuration view* for equipment and functional units and the *mapping view*.

There will be a set of starting points to choose from for both equipment and functional so that the *DSL user* does not start from scratch each time when building a new configuration, even if that is an option.

2.2.1 Creating a Configuration

There will be validation for mapping and connections between units to ensure that no invalid configurations are created. Another thing that is planned is to calculate how much an equipment unit is used and color-code it if overused. Programming or choosing a graphical library to represent the functional and equipment graph is another task to be done.

2.3 Evaluation

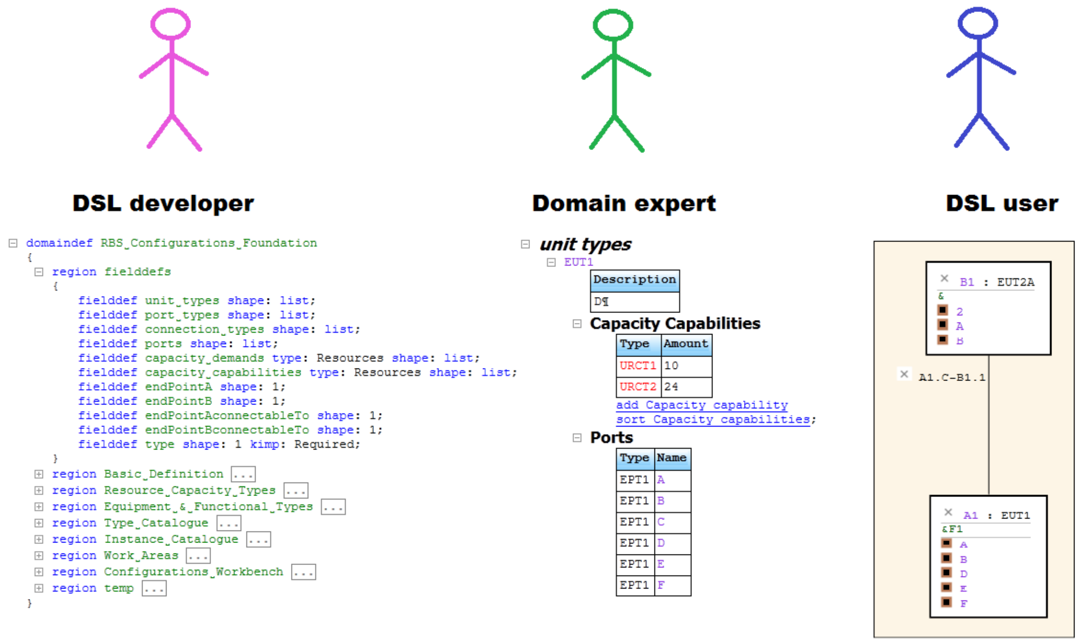
To ensure that the prototype has a practical usage, an evaluation will be done at the end. The group evaluating the created prototype should be persons working in the field. In that way the evaluation gives the best information if the prototype will be of practical usage. The evaluation will let the group of people test the prototype doing a specific task. After the task is done they will answer a questionnaire. The questionnaires will be analysed. In addition to the questionnaire, the functionality will be constantly tested during the development of the prototype.

3. Technical Background

The platform that is used in this degree project, the *Intentional Domain Workbench (IDW)*, is an example of a *language workbench.* The following sections will describe what a *language workbench* is and then focus on *Intentional Domain workbench*. When *projections* are mentioned in the following text, it refers to how the data in a *language workbench* is presented graphically.

3.1 Language Workbenches

Language workbenches are built upon a tree structure which means in *IDW* that everything needs to be defined where it is in the tree before any compilation can be done. The workbench is used to define knowledge of a specific field in a concrete way based mostly on business knowledge. This is what is called a *Domain Specific Language (DSL)*. This means that instead of a language as java or another programming language, a *DSL* is specifically tailored to one small domain [10]. The method is to define something known as the *meta model* or *schema*. It is important to think of every detail due to how the *DSL user* and domain expert want it in the schema. After the *schema* or *meta model* is done, it is possible to project the language in a comfortable *projection* for a *DSL user* or domain expert to use it. A *projection* is what it is called when the program visualizes the data in some way such as tables, text, diagram, images etc. The *language workbench* has three different kinds of persons that interact with it [11]. The domain experts that have detail knowledge about the types, the *DSL developer* who interprets the domain expert and translates his knowledge into the *type catalogues* and the *DSL user*, who will use the finished product to create solutions for different situations as illustrated in *Figure 3.1*.



```
⊟ domaindef RBS_Configurations_Foundation
  {
   ⊟ region fielddefs
     {
        fielddef unit_types shape: list;
        fielddef port_types shape: list;
        fielddef connection_types shape: list;
        fielddef ports shape: list;
        fielddef capacity_demands type: Resources shape: list;
        fielddef capacity_capabilities type: Resources shape: list;
        fielddef endPointA shape: 1;
        fielddef endPointB shape: 1;
        fielddef endPointAconnectableTo shape: 1;
        fielddef endPointBconnectableTo shape: 1;
        fielddef type shape: 1 kimp: Required;
     }
   ⊞ region Basic_Definition ...
   ⊞ region Resource_Capacity_Types ...
   ⊞ region Equipment_&_Functional_Types ...
   ⊞ region Type_Catalogue ...
   ⊞ region Instance_Catalogue ...
   ⊞ region Work_Areas ...
   ⊞ region Configurations_Workbench ...
   ⊞ region temp ...
  }
```

⊟ ***unit types***

⊟ EUT1

Description
D¶

⊟ **Capacity Capabilities**

Type	Amount
URCT1	10
URCT2	24

add Capacity capability
sort Capacity capabilities;

⊟ **Ports**

Type	Name
EPT1	A
EPT1	B
EPT1	C
EPT1	D
EPT1	E
EPT1	F

Figure 3.1 The Different Users

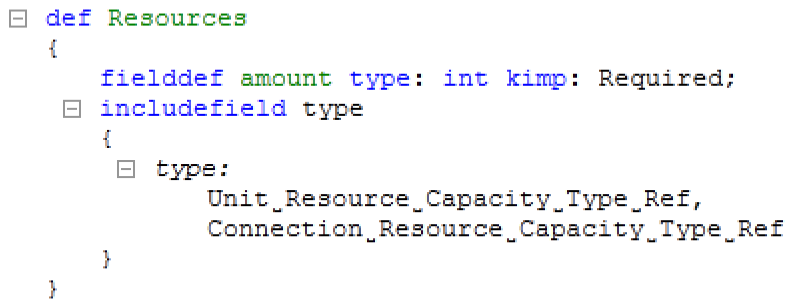
3.2 Intentional Domain Workbench

The environment used for this project is *Intentional Domain Workbench*. The reason for this was that it is possible to project the model visually in several different ways without doing changes in the domain. The possibility to have several *views* but only one model can make it possible for different users to understand the model, and modify it in the most appropriate way.

Intentional Domain Workbench is an environment platform that can be used to make other tool platforms known as *workbenches*. In this project the *type catalogues view* and the *configuration view* are examples of new *workbenches* created by *IDW*. The *views* have a specific purpose but much of the original *workbench* functionality remains.

3.2.1 Meta Model or Schema

The *meta model* or *schema* is the building blocks of the program. This is where the *DSL* is defined. In *Intentional Domain Workbench* this is done by using the keyword *def* followed by a name which can be seen in *Figure 3.2*. In the *def* it is possible to include different fields with different types and decide if it should be a list or not. After the *schema* is defined the *DSL developer* can start building the *projections*. If the standard *projection* is not wanted, then for each item that needs a different *projection*, a so-called *projectdef* is defined.



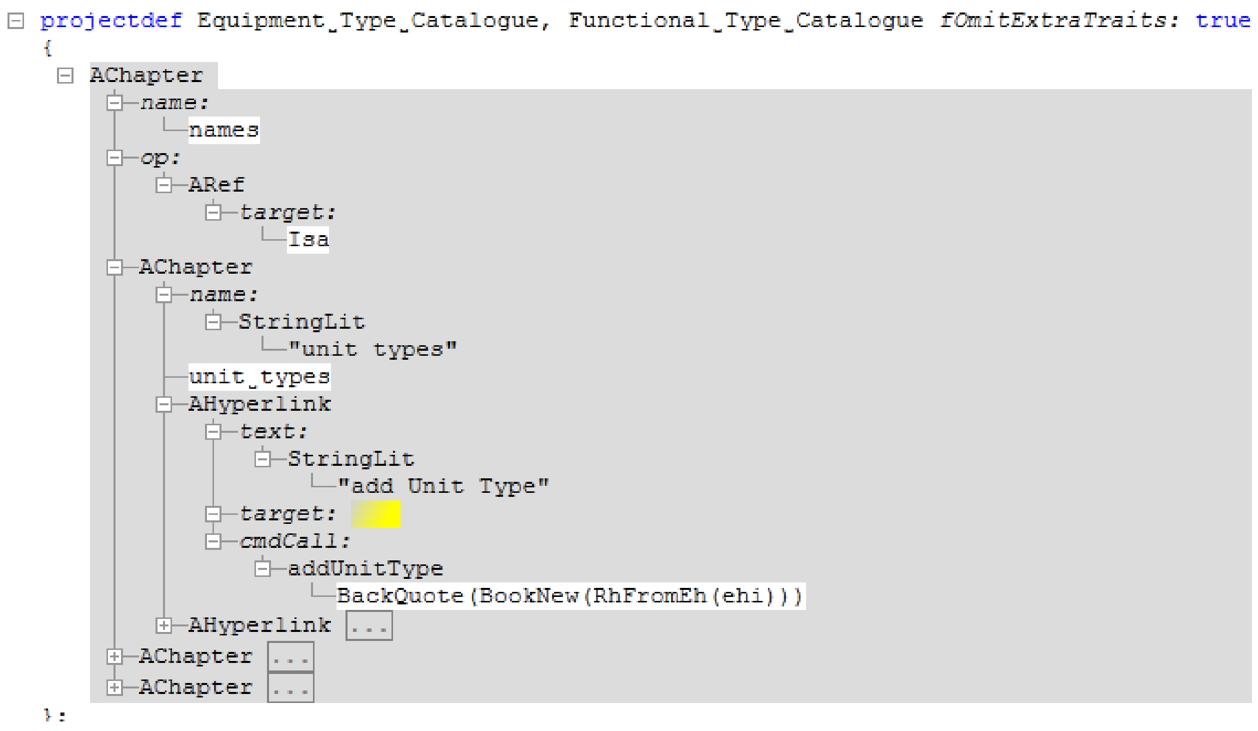
```
⊟ def Resources
  {
     fielddef amount type: int kimp: Required;
   ⊟ includefield type
     {
      ⊟ type:
            Unit_Resource_Capacity_Type_Ref,
            Connection_Resource_Capacity_Type_Ref
     }
  }
```

Figure 3.2 A def example

3.2.2 Projections and Projectdef

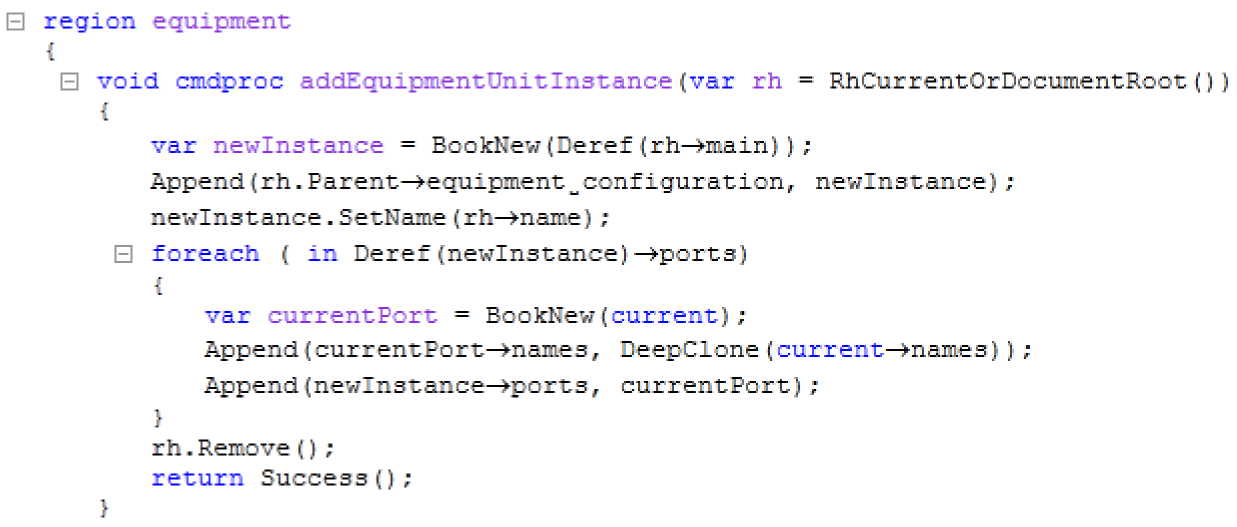
A *projectdef* (*Figure 3.3*) is what determines how a specific *def* is projected if another *projection* than the standard *projection* is wanted. In the *projectdef* it is decided which *def* it will determine the *projection* for. They are listed after the declaration that the following code is a *projectdef*. One *projectdef* can determine the projection for several *defs*. *Figure 3.3* is the *projectdef* for the *defs*, *Equipment Type Catalogue* and *Functional Type Catalogue*.

The *projectdef* in *Intentional Domain Workbench* is built by the *Alanguage. Alanguage* is specific to *Intentional Domain Workbench* and allows the *DSL developer* to build very specific *projections*. Options include *AChapter*, for the ability to close and open, *Atable* to project as tables, *Agraphs* for a diagram and more. The *DSL developer* then uses these keywords to build the structure for the projection. An example of using *Alanguage* is shown in *Figure 3.3* If more interactivity is required for example pressing a button to add a new item defined in the schema or sorting a list this can be programmed in *CL1*. *CL1* is an extension of C# with added features so it is possible to easily incorporate nodes and trees in the programming. *Figure 3.4* is an example of how to add a node to a tree, and in this case used for creating new instances in the *Configuration view*



```
projectdef Equipment_Type_Catalogue, Functional_Type_Catalogue fOmitExtraTraits: true
{
   AChapter
    ├─name:
    │   └─names
    ├─op:
    │   └─ARef
    │       └─target:
    │           └─Isa
    ├─AChapter
    │   ├─name:
    │   │   └─StringLit
    │   │       └─"unit types"
    │   ├─unit_types
    │   ├─AHyperlink
    │   │   ├─text:
    │   │   │   └─StringLit
    │   │   │       └─"add Unit Type"
    │   │   ├─target:
    │   │   └─cmdCall:
    │   │       └─addUnitType
    │   │           └─BackQuote(BookNew(RhFromEh(ehi)))
    │   └─AHyperlink ...
    ├─AChapter ...
    └─AChapter ...
};
```

Figure 3.3 Projectdef for the type catalogues

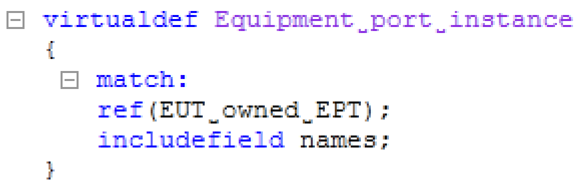


```
region equipment
{
   void cmdproc addEquipmentUnitInstance(var rh = RhCurrentOrDocumentRoot())
   {
      var newInstance = BookNew(Deref(rh→main));
      Append(rh.Parent→equipment_configuration, newInstance);
      newInstance.SetName(rh→name);
      foreach ( in Deref(newInstance)→ports)
      {
         var currentPort = BookNew(current);
         Append(currentPort→names, DeepClone(current→names));
         Append(newInstance→ports, currentPort);
      }
      rh.Remove();
      return Success();
   }
```

Figure 3.4 CL1 program example

3.2.3 Virtualdef

A *virtualdef* is a *def* that matches on a reference to another *def*. This can be used to make certain *projection* for parts of the *schema* or validation. So instead of creating a *projectdef* or validation for all fields of the type, instead the *DSL developer* can set just a couple of fields to the *virtualdef* which means that the *projectdef* or validation will only apply to those fields. Another improvement is the ability to add fields in the *virtualdef* with extra information which can be seen in *Figure 3.5.*

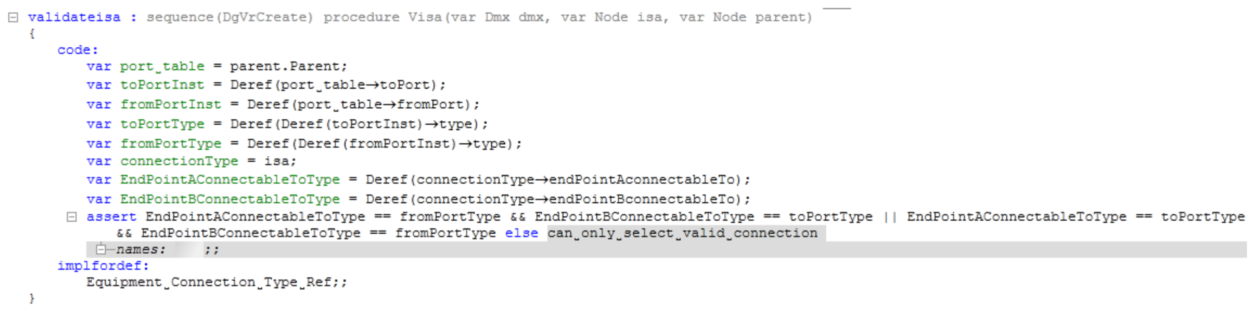


```
⊟ virtualdef Equipment_port_instance
  {
   ⊟ match:
      ref(EUT_owned_EPT);
      includefield names;
  }
```

Figure 3.5 virtualdef

3.2.4 Validation

In *Intentional Domain Workbench* there are two different kinds of validations, *validateisa* and *validateNode*. *ValidateNode* is used to check the relevant nodes after committing, while *validateisa* limits choices before the domain expert or the *DSL user* selects a new node to add to the tree. *Valiadateisa* is implied for one *def*, a *virtualdef*.
In the *validateisa* (*Figure 3.6*) node the programmer write code and in the end there is one or several assert statements. If all asserts evaluate to true the nodes are allowed to be chosen but if at least one evaluates to false that node will not be shown when the domain expert or *the DSL user* is selecting from the dropdown menu.



```
⊟ validateisa : sequence(DgVrCreate) procedure Visa(var Dmx dmx, var Node isa, var Node parent)
  {
     code:
        var port_table = parent.Parent;
        var toPortInst = Deref(port_table→toPort);
        var fromPortInst = Deref(port_table→fromPort);
        var toPortType = Deref(Deref(toPortInst)→type);
        var fromPortType = Deref(Deref(fromPortInst)→type);
        var connectionType = isa;
        var EndPointAConnectableToType = Deref(connectionType→endPointAconnectableTo);
        var EndPointBConnectableToType = Deref(connectionType→endPointBconnectableTo);
     ⊟ assert EndPointAConnectableToType == fromPortType && EndPointBConnectableToType == toPortType || EndPointAConnectableToType == toPortType
           && EndPointBConnectableToType == fromPortType else can_only_select_valid_connection
        ⊟─names:     ;;
     implfordef:
        Equipment_Connection_Type_Ref;;
  }
```

Figure 3.6 Validation

3.2.5 BackQuote, BookQuote, Cmd_proc and Classes

A *BackQuote* in *IDW* is when the *DSL developer* switches from the normal context to a programming context for example writing some code in a *projectdef*. *BookQuote* is the opposite which is when the programmer wants to go from code to be able to add a new *Book*. A *Book* is a free tree that has not been inserted into the main tree yet. But it is easily inserted with append or another appropriate command. Each new node or subtree that any of the users wants to insert to the main tree is done by defining a new *Book*.

A *cmdproc* is a function written in *CL1* code that is executed when the domain expert or the *DSL user* clicks a hyperlink that links to that *cmdproc*. In addition to this the *DSL developer* can also write *CL1* classes in *IDW*, to define public classes with helper functions that can be comparators, name creators etc.

3.3 Descriptions of Terms Used

It is hard to visualize so much complex data and using the same language consistently. The method that was used in this project is to create several equipment and functional types. The types that exist are *equipment connection types, functional connection types, equipment port types, functional port types, equipment unit types, functional types, Connection Resource Capacity Types (CRCTs)* and *Unit Resource Capacity Types (URCTs).*

The *URCTs* and *CRCTs* are of a fixed number that will not be changed. All the other types can vary by adding new types in the *type catalogues workbench*. Functional and equipment types are different types but they have the same structure.

Each type except for the *URCTs* and *CRCTs* also has an instance. The following instances exist; *equipment unit instance*, *functional unit instance, equipment connection instance, functional connection instance*, *equipment port instance* and *functional port instance.* An instance is either referred to as the previous names or similar but without mentioning instance. What makes it an instance is that all instances have a reference to a type but are not the same item as the type and each instance is unique.

In this report *types catalogues view* or *configuration view* is often mentioned. A *view* in this case is a *workbench* created by the *Intentional Domain Workbench*. In a *workbench* it is not only possible to see the *types catalogues* or the configuration but it is also possible to edit and the visualization will adapt instantly.

4. Implementation

The following sections are going to describe how the prototype was developed. Instead of describing it in a chronological order, the different sections are instead divided into appropriate logical building blocks of the prototype. The focus will be on *the meta model or schema*, *projections*, *validation and cmdprocs* and *helper classes*.

4.1 Meta Model or Schema

In the *schema* the *unit types, connection types* and *port types* are defined. The same pattern was used for both equipment and functional types. A *unit type* contains references to *port types.* A *connection type* also contains references to what *port types* it can be connected to. In the *schema* a certain number of *CRCT* and *URCTs* are defined. *URCT* stands for *Unit Resource Capacity Type* and *CRCT* stands for *Connection Resource Capacity Type.* An *equipment unit type* has a field called *capacity capability* while a *functional unit type* has a similar field named *capacity demand*. This is also true for the *connection types*. The field is a list of type *Resources*. The *def Resources* have two fields named *amount* and *type* as can be seen in *Figure 3.2*. The *type* field shows what kind of *URCT* or *CRCT* it is and the *amount* field shows how much of it that is required. Each *unit type* has a certain amount of *CRCT* and *URCT types*. *Capacity capability* tells how many functional items can be deployed on an *equipment unit* and what kind of functional items it supports. The *capacity demand* on the functional item on the other hand tells what kind of capability is needed to support that function. The two *type catalogues* are defined as each containing the same fields but with different types. The fields in a *type catalogue* are *unit types, port types* and *connection types.*

The basics for the configurations and mapping are that *virtualdefs* are created that match references to *types* in the *type catalogue*. This means one *virtualdef* each for the *functional port type, the equipment port type, the equipment unit type, the functional unit type, the equipment connection type* and *the functional connection type*.

The *def* that represents configurations is named *config* and contains the following fields: *equipment connections, functional connections, equipment configuration, functional configuration, hide ports* and *hide connection names.*

In the *schema* a few different *wizards defs* are designed, one *wizard* to add new *equipment units*, one *wizard* to add connections between ports on equipment units. Two more *wizards* are defined that have the same functionality for the functional configuration. One *wizard* to map between a *functional unit* and an *equipment unit*. The last *wizard* is the mapping between *functional connection* to *equipment unit* or *equipment connection.*

4.2 Projections

In this project two different *views* are created. Each *view* is meant to be used by different users, and each *view* is projected so that the intended user has all information needed for editing and understanding. The different *views* are, the *"type catalogues view"* and the *"configuration and mapping view"*. Also in general a color coding is used where equipment is set to blue in the *type catalogues* and functional is set to green. In the *"configuration and mapping view"* there is a slight difference, while functional is still green the equipment matching color is black instead. Since the default style for names in *IDW* is green this style changed to violet to avoid misinterpretations. The following sections will explain the *projectdefs* for each view in more detail.

4.2.1 Type Catalogues View

Both the functional and equipment *type catalogue* will be displayed in a similar way. The *projectdef* for a *unit* is an *Adecl* for the ability to open and close. Every field in *unit* except description is projected with an *Achapter* which means it gets a black headline and can open and close. Furthermore so is the content of each field displayed with the help of *Atables* where the first row will be colored blue or green depending on if it was the equipment or functional unit. Under each table except the description table there are two hyperlinks for adding a new item of that definition or sort the relevant table.

Port types are simple since they are only an *Adecl* and then the description field is defined to be projected as an *ATable*. The *connections* are using the same principle as *units*.

The *type catalogues view* will display all different *types* of a configuration. Here the domain expert will be able to add new *types* and modify old *types*. The other *views* will use the definitions in this *view*. In this *view* all data regarding a specific *type* is typed in so this is the only *view* where the *types* can be modified. The *type catalogue* is defined with one *Achapter* and then each field is also defined as an *Achapter*.

4.2.2 Configuration and Mapping View

This *view* starts with defining the hyperlinks to different *cmdprocs* which are the following: *Add Equipment Unit, Add Equipment Connection, Add Functional Unit, Add Functional Connection, map Functional Unit to Functional Connection* and *map Functional Connection to Equipment Unit or Connection.*

In the *projectdef* for *Config* there are two main *BackQuotes* that create the two different diagrams, one for equipment and one for functional. It is in those *BackQuotes* the flags are checked to see if *connection* names are hidden or shown. The *connections* are defined as *Agraphedges* with no direction. In the *projectdefs* for *units* both the functional and the equipment are defined as an *Agraphnode*. Also in each *projectdef* in a *unit* there is some code to check the hide/show unused ports flag. The *projectdef* for the different *wizards* was designed using *ATables* and following the color coding with blue for equipment, green for functional and grey for the mapping table.

4.3 Validation

A certain amount of *validateisas* were created for the *configuration view*. A validation for *equipment units* were created so that the *DSL user* can only choose *units* from the current configuration when using the *wizard*. Validations for both from and to equipment port were created that checked that the *DSL user* could only choose a *port* from the selected *unit* and a *port* that was free. Another validation rule was created for *equipment connection* so that when the ports are chosen the *DSL user* can see which connections match those ports and there is no direction on the connection. Naturally all the validators for the equipment side were matched with analogous validators on the functional side. Except for the equipment and functional validators, mapping validators were also added.

The validators make sure the *DSL user* can only map to items in the same configuration. There is one validation for mapping *functional unit* to *equipment unit*. If both fields are empty, all *units* in the configuration will be valid but as soon as one of the fields is filled in, the validator will check the *capacity demand* or *capacity capabilities* and sort out only the *units* that match. The same principle is used in the validator for mapping *functional connection* to *equipment unit* or *equipment connection*.

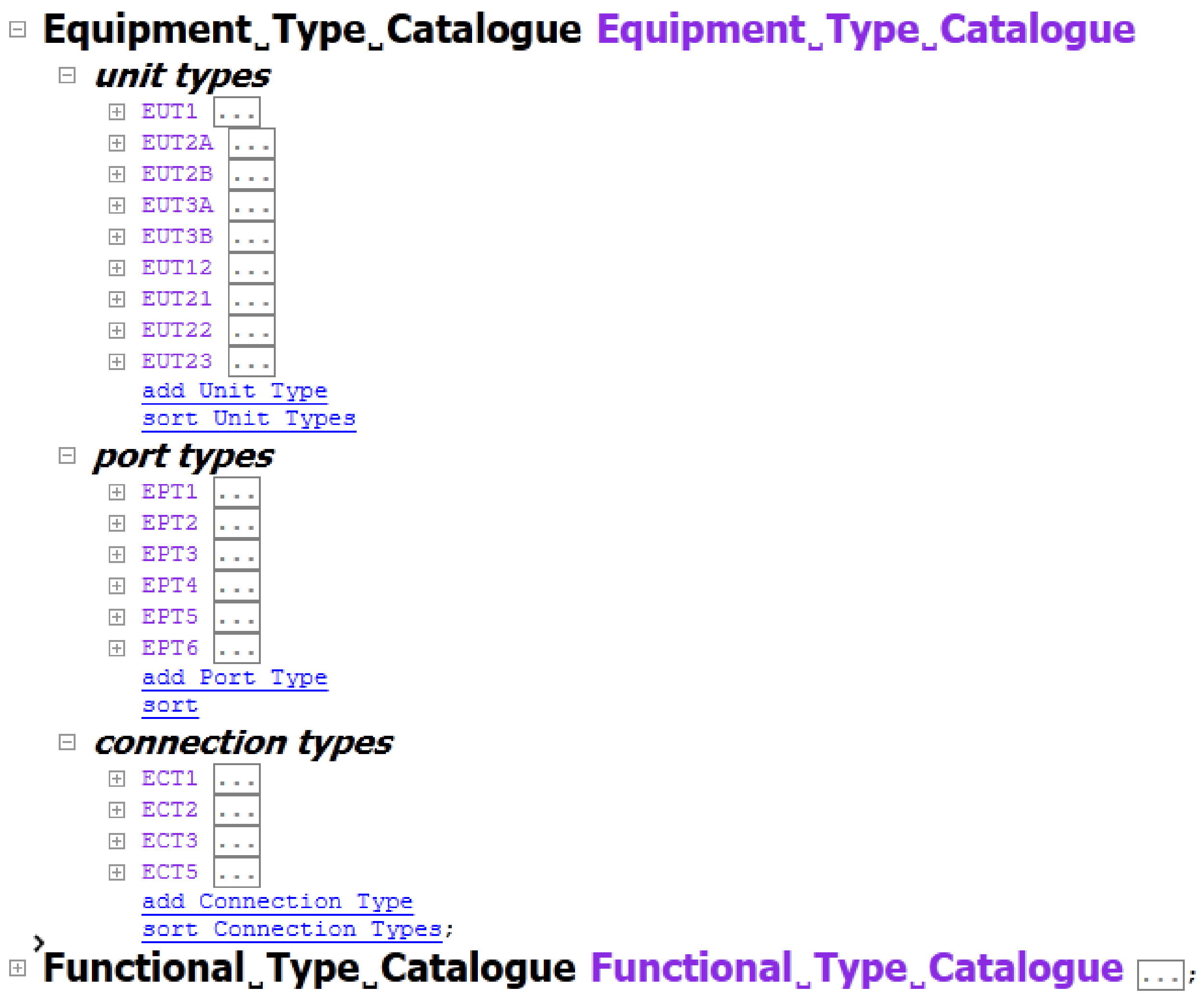
4.4 Cmdprocs and Helper Classes

Cmdprocs have been created for the different sort links so there exists a sort command for each group of nodes that should be sorted. There also exist *cmdprocs* for adding new items into the tree such *as Units, Connections, Ports, Instances* of the same and more. In addition to the *cmdprocs* there exist public helper classes with static methods and a couple of comparators for the sorting methods. The other static methods that exist are for creating connection names and swap values on flags. A connection name is created by taking the *unit* name from the from port then adding the *port* name and then bind it together with the to side which results in a name of the form Uf.pf-Ut.pt where Uf stands for the *unit* name for the from *unit*, pf the portname in the from *unit* and Ut and pt are equivalent for the to part of the name.

5. Result

5.1 Type Catalogues

In the *type catalogues view* there exist two *catalogues*, the *Equipment Type Catalogue* and the *Functional Type Catalogue*. Since both *catalogues* work similarly except that *capacity capabilities* are named as *capacity demands* in the *functional type catalogue*, only the *equipment type catalogue* will be explained here. The domain expert can collapse and expand the *catalogue* as seen in *Figure 5.1* where one *catalogue* is collapsed and the other expanded.

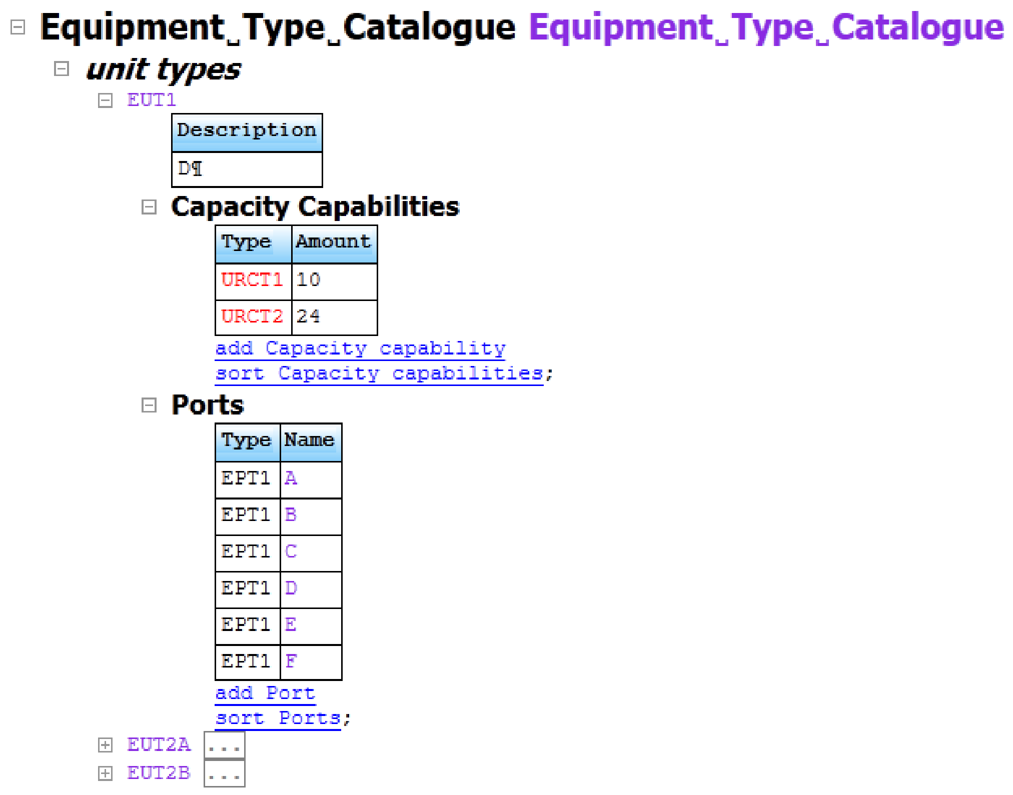


```
⊟ Equipment_Type_Catalogue Equipment_Type_Catalogue
  ⊟ unit types
    ⊞ EUT1 ...
    ⊞ EUT2A ...
    ⊞ EUT2B ...
    ⊞ EUT3A ...
    ⊞ EUT3B ...
    ⊞ EUT12 ...
    ⊞ EUT21 ...
    ⊞ EUT22 ...
    ⊞ EUT23 ...
      add Unit Type
      sort Unit Types
  ⊟ port types
    ⊞ EPT1 ...
    ⊞ EPT2 ...
    ⊞ EPT3 ...
    ⊞ EPT4 ...
    ⊞ EPT5 ...
    ⊞ EPT6 ...
      add Port Type
      sort
  ⊟ connection types
    ⊞ ECT1 ...
    ⊞ ECT2 ...
    ⊞ ECT3 ...
    ⊞ ECT5 ...
      add Connection Type
      sort Connection Types;
⊞ Functional_Type_Catalogue Functional_Type_Catalogue ...;
```

Figure 5.1 Equipment Type Catalogue

In a *catalogue* there are three sections *Unit Types*, *Port Types* and *Connection Types*. All of these can be collapsed and expanded. Under each section there are hyperlinks for adding a new item or sort the existing items under the category. Opening up a *unit* reveals the description, *Capacity*

Capabilities and *Ports* (*Figure 5.2*). Each of these sections in the *unit* can be closed and expanded except for the description. The description is a table that contains the description of the *unit* and can be modified by the domain expert. *Capacity capability* is a table that contains the kind of *functional demand* that can be mapped onto the *unit*. Under the table there are links to add new *capabilities* and to sort the current ones. The last table in the *unit* is the *Ports* table and in this table the *ports* for the *unit* exist. This one also have an add and sort link as above. When the domain expert first creates a *unit*, *Description*, *Capacity* and *Ports* are empty. The *Capacity capabilities* when added can only be chosen from the predefined options in the program while the *ports* must be chosen from the *ports* section of the *catalogue*. The *ports* section is basically *add Port* and *sort Ports* hyperlinks and each new *port* only has a description that can be modified and do not require anything more except the name. The connection section also has *add connection* and *sort connection* hyperlinks. In addition to a description it is like a *unit* in that it has *capacity capabilities*. Instead of *ports* though it has a *connectable to* a section and in it there is a table which shows which port *EndPoint A* can connect to and which port *EndPoint B* can connect to.


Equipment_Type_Catalogue Equipment_Type_Catalogue

unit types

EUT1

Description
D¶

Capacity Capabilities

Type	Amount
URCT1	10
URCT2	24

add Capacity capability
sort Capacity capabilities;

Ports

Type	Name
EPT1	A
EPT1	B
EPT1	C
EPT1	D
EPT1	E
EPT1	F

add Port
sort Ports;

EUT2A ...
EUT2B ...

Figure 5.1 Units in type catalogue

5.2 Configuration

Configurations are created in the *"configuration and mapping view".* A configuration consists of several *equipment units, functional units* and *connections,* of which each is an instance of *types* from the *type catalogue*. If another *type* is wanted, the domain expert has to add that *type* in the *type catalogue view*. After that the *DSL user* can go back to the *configuration and mapping view*, creating an instance of the new *type*.

Figure 5.3 shows an example of an equipment graph. At the top left in each node there is a grey cross which is a delete button. To the right of the cross the name of the *unit* is displayed followed by the *type* of the *unit*. Below the name a list of green names appear. These are the names of all *functional units* and *connections* that are mapped on to the specific *unit*. If the list is empty it is not displayed, like *B3* in *Figure 5.3*. Under the list of *functional units* there is another list which is the *ports* that are unused. It is possible to hide the unused *ports* by pressing the link *"hide/show Unused ports".* If the *DSL user* wants to see the unused ports again the *DSL user* can press the link again. The link is shown in *Figure 5.4.*

Each edge in the graph represents a connection between two units. All edges have a name that is close to them, and similar to the nodes a grey cross is used for deleting a connection. It is possible to hide the name of the connections by pressing the *"Hide/Show Connection names".*

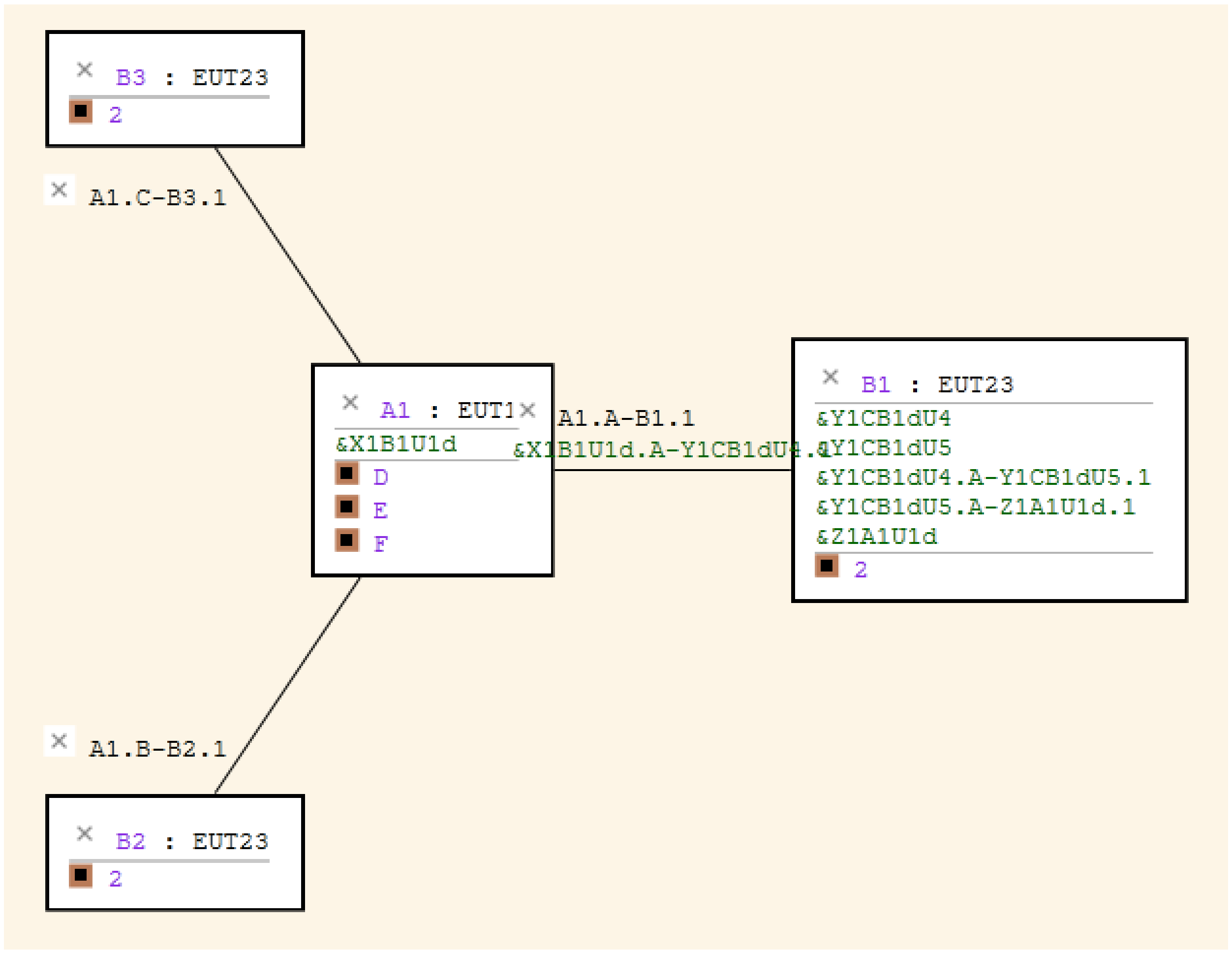
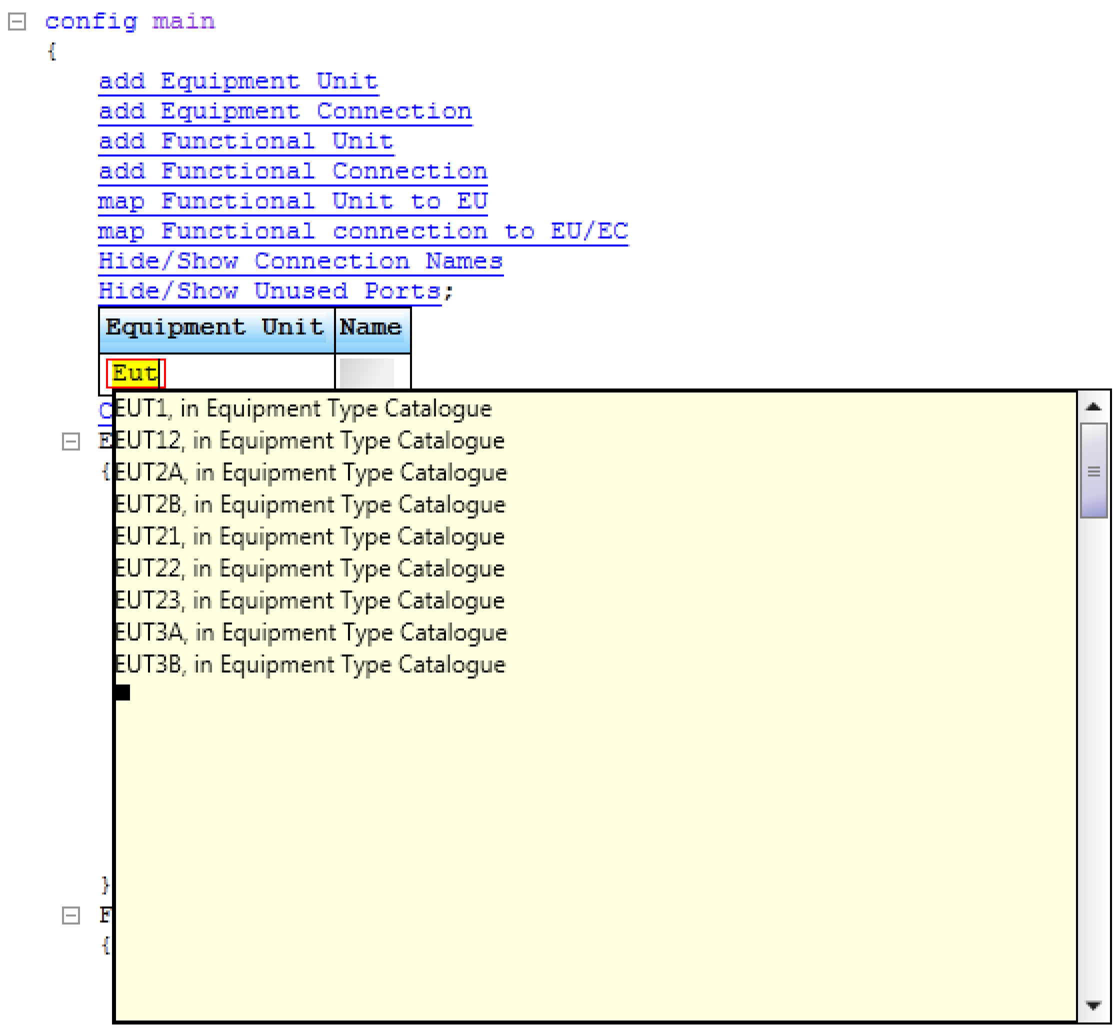


Figure 5.3 Units in configuration view

In order to add a *unit instance* to a configuration, the *DSL user* has to press the *"add new unit"* link above in the graphical visualization of the configuration. When the link is clicked a table appears between the link and the graph. In the table the *DSL user* will be able to enter what *type* of instance that is wanted. If the *DSL user* press ctrl+space a list of all available *types* will appear. The table also require the *DSL user* to type in a name for the new instance. After the fields in the table have been filled the *DSL user* may press the *"ok"* link below the table. This will create a new *unit instance* of the given *type* in the graph. The table and the list of suggestions are shown in *Figure 5.4* with the table for adding new equipments, and the list with suggestions of different *equipment types*.



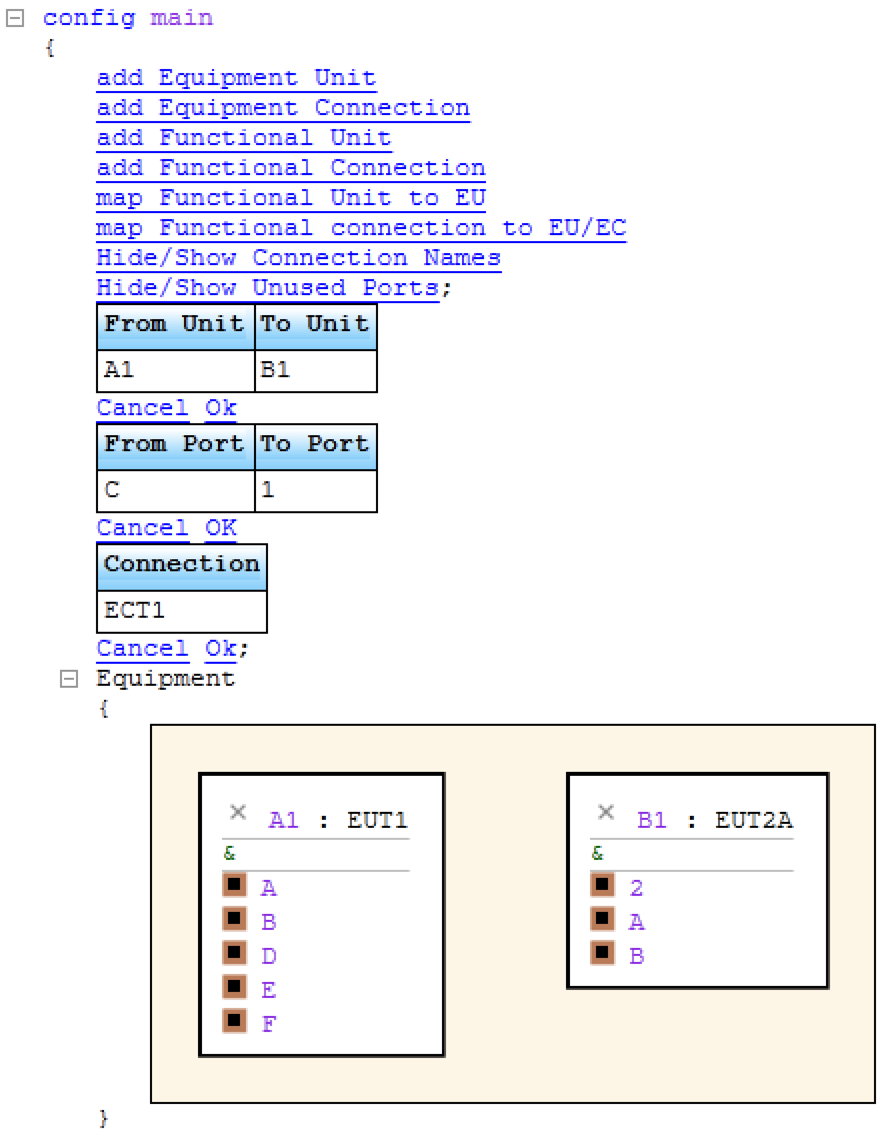
```
config main
{
    add Equipment Unit
    add Equipment Connection
    add Functional Unit
    add Functional Connection
    map Functional Unit to EU
    map Functional connection to EU/EC
    Hide/Show Connection Names
    Hide/Show Unused Ports;
```

Equipment Unit	Name
Eut	

```
EUT1, in Equipment Type Catalogue
EUT12, in Equipment Type Catalogue
EUT2A, in Equipment Type Catalogue
EUT2B, in Equipment Type Catalogue
EUT21, in Equipment Type Catalogue
EUT22, in Equipment Type Catalogue
EUT23, in Equipment Type Catalogue
EUT3A, in Equipment Type Catalogue
EUT3B, in Equipment Type Catalogue
```

Figure 5.4 Adding Equipment Unit Wizard

To link two *equipment unit instances* together there exists an *"add Equipment Connection"* link above the graph. When it is pressed a table is opened where the *DSL user* has to enter the two nodes that the *DSL user* wants to connect. Below the table there will be an *"ok"* link which will open a table below the table with nodes. If the *DSL user* selects one field and hit ctrl-space a list with suggestions of only valid *ports* will appear. If there exists no valid *ports* the list will be empty. If the *DSL user* knows the name of the *port* the *DSL user* can enter the name of the *port* and hit enter. After two *ports* are selected, one for each node, the *DSL user* can press the link *"ok"* below this table. Then the last table appears where the *connection type* can be entered. In this field like the ones above the *DSL user* can either hit ctrl-space or write directly in the field what *type* of *connection* the *DSL user* wants. After *connection type* is selected the *DSL user* presses a link that says *"ok"* and a *connection* will be established between the two nodes. The *ports* allocated for the *connection* will be removed from the pool of free *ports*. An example how all tables look like when they are filled is shown in *Figure 5.5*. Once a *DSL user* is done adding different types to the graph and connecting them it will save all the nodes and connections between them by pressing ctrl-s.



```
config main
{
    add Equipment Unit
    add Equipment Connection
    add Functional Unit
    add Functional Connection
    map Functional Unit to EU
    map Functional connection to EU/EC
    Hide/Show Connection Names
    Hide/Show Unused Ports;
```

From Unit	To Unit
A1	B1

```
    Cancel Ok
```

From Port	To Port
C	1

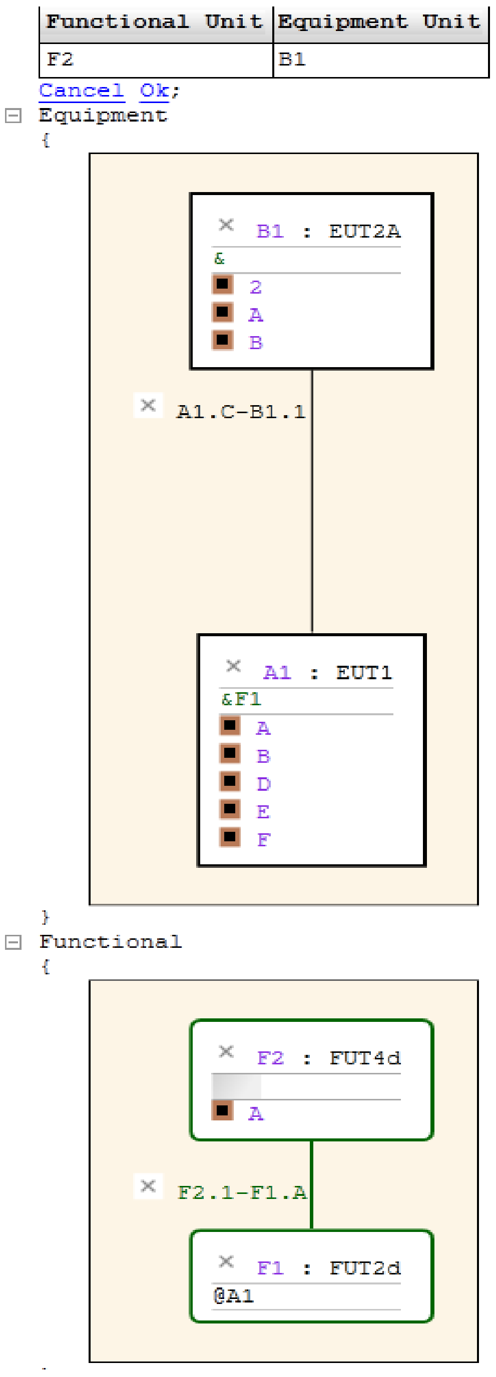
```
    Cancel OK
```

Connection
ECT1

```
    Cancel Ok;
    Equipment
    {
```

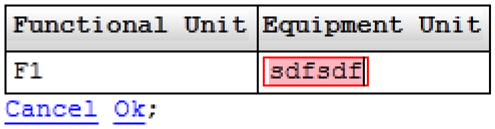
```
    }
```

Figure 5.5 Adding Equipment Connection Wizard



Functional Unit	Equipment Unit
F2	B1

```
Cancel Ok;
Equipment
{
    B1 : EUT2A
    &
    2
    A
    B
  A1.C-B1.1
    A1 : EUT1
    &F1
    A
    B
    D
    E
    F
}
Functional
{
    F2 : FUT4d
    A
  F2.1-F1.A
    F1 : FUT2d
    @A1
```



Functional Unit	Equipment Unit
F1	sdfsdf

```
Cancel Ok;
```

Figure 5.6 No matching Unit in mapping

Figure 5.7 Mapping

5.3 Mapping

In order to map a *functional unit* on to an *equipment unit* the *DSL user* has to press the link *"map Functional unit to EU"*. The link will create a table where the *DSL user* can enter an *equipment unit* and a *functional unit*. To ensure that *a functional unit* really can be deployed on the equipment there exist a validation for the second *unit* that is entered. If the second *unit* that is entered do not match on the first *unit* a red field will appear around the name (*Figure 5.6*) and if the *DSL user* presses enter the text will turn purple. If it is a match it will first be bold and when enter is pressed it will appear as normal black text as seen in *Figure 5.7*.

6 Evaluation

This chapter describes how the testing of the prototype created was performed. The evaluation consists of two parts. First, we describe the practical testing which contains description of how each view was tested and evaluated. The other part is a questionnaire handed out to a group of people after they have tested the prototype. After each part a paragraph describing the result of the evaluation is given.

6.1 Practical Testing

First the *type catalogue view* is evaluated since a full *type catalogue* is needed to test the *configuration view*. The evaluation is done by filling up the *type catalogue* in the same manner as a domain expert would do it by using the add and sort hyperlinks when appropriate.

A similar approach was taken when evaluating the configuration view instead of having four finished configurations and see how they are displayed. Instead each of the four configurations is built the same way a *DSL user* would do it, clicking *Add Unit*, *Add Connections* and *Add Mapping* when appropriate. Doing it in this way, usability is also evaluated and not just the finished configuration.

6.1.1 Result of Practical Testing

Testing both the *type catalogue view* and the *configuration and mapping view* showed that both work as intended. The only exception is sorting the tables in the *type catalogue view*. The sorting requires that no configurations *in configuration and mapping view* reference to the table that is sorted. If any configuration is referring to the table the references stops working and is incompatible with any new instances in the configuration.

6.2 Evaluation Questionnaire

When evaluating the practical usage of the prototype created, a group of people were given a task which they should do using the prototype. Afterwards they were asked to answer a questionnaire. The questionnaire had the following questions:

1. What is missing in the program?
2. What is good about the program?
3. How easy does it seem to be to use? (scale 1-5)
    i. (1 really hard , 2 hard, 3, okay, 4 easy, 5 really easy)
4. In comparison to the tools you know of now, how does this program compare?
    i. (1 harder, 2 slightly harder, 3, about the same, 4 slightly easier, 5 easier)

These questions were handed out to four persons who after having tested the program got to answer the questions.

6.2.1 Result of Questionnaire

The evaluation of the program showed that in question 4, 75% of the testers said that the program is okay in functionality compared to the tool they are using now. 25% said it is better. The functionality that the users would like to add the most according to question 1 is the possibility to rearrange the mapping, and validation so that a function cannot exist on two equipments at the same time. The good thing with the prototype is that it is model based and not only a painting tool.

50% said in question 3 that the prototype was easy to use while the other half said it was okay in the matter of usability. The usability that the testers appreciated according to question 2 was the ability to choose from a list of alternatives when entering values in the tables, and that the layout matches well the *DSL*. Improvements that could be made according to the questionnaire are drag and drop, pictures of real equipments for easier identifications, and a simpler way to add and change data in the graph.

7. Conclusion

7.1 Recap

In this degree project, we investigated and then implemented two separate *views*. The two different *views* are the *type catalogues view* and the *configuration and mapping view.* The *type catalogues view* have two *catalogues*, one for functional and one for equipment. Both *catalogues* consist of the following*: chapters unit types*, *port types* and *connection types.* For each *chapter* the domain expert can add a new item of that *type*. And there is also an option to sort the items. The other *view* is the *configuration and mapping view*. Here the *DSL user* has two diagrams: one for functional and one for equipment. The *DSL user* has the ability to add, connect and delete items in each diagram. The mapping is shown by text on the item it is mapped to. It is also possible to hide characteristics to make a cleaner diagram. The standard view is a circular layout for equipment and a standard layout for functional. Validation is also included for the ability to choose the correct nodes.

7. 2 Critical Discussion

7.2.1 Sustainability Concerns

Ericsson wants to build energy effective networks with good energy performance [12]. If the ability to visualize configurations exists, it might be easier to identify parts and make improvements which mean a configuration can be optimized to be more environment friendly. When the configurations are visible and editable it should be easier to validate how to optimize a certain configuration to make it more energy efficient. It should be possible to get a better picture and then a person might be able to decide how to make different configurations more environment friendly for example.

7.2.2 Configuration and Type Catalogue

One of the project´s strengths is the *type catalogue* where it is easy to add new complex *types* and being able to make instances of the new *types* in the *configuration view.*
We think that the *type catalogue* is fairly clean and easy to use for any new domain expert.

While building the configurations in the *configuration and mapping view*, we used the help of the validation, which were helpful to create different configurations. The help we received from the validation was that when creating new *units* in a configuration we could chose *units* from a list that popped up when typing in the *type* field. In a similar way we used the list of valid *types* when we created all the connections between *unit instances.* It saves time getting suggestions in a list of which *type* that could be used, instead of writing the entire name of the *unit instance*. It also makes it easier, since we did not have to look up the name in the configuration when we forgot which name we could choose from. The validation did also ensure that only valid configurations were made.

The validation was also a help while building the *type catalogues*. When creating the different *types*, it was helpful when defining *ports* and *capacity/demand* in *units* to get a list of which *types* of *ports* that existed, and also what kinds of *capacities/demands* that existed. In that way we could make sure that all *units* used the same set of *ports* and *capacities/demands.*

One of the project´s weaknesses is the inability to display names for connections without any overlapping parts. For example when showing the name for connection A1.A-B1.1 in *Figure 5.3* A1 and B1 overlaps making all three harder to read. That is one reason why the *hide/show connections names* hyperlink is good because then the *DSL user* have the ability to remove the connection name and mapping to be able to see a cleaner diagram.

7.2.3 Intentional Domain Workbench

At the beginning it was difficult to work with *IDW* since it was a new type of language for us. The new way of thinking and structuring was the hardest part to learn. After spending time with the program and starting to learn about the way of using it, *IDW* showed promising potential. The code was simple to understand once getting a grasp of IDW, and the code readability was good. It was also faster than if we would have used something like Eclipse and Java due to it being on a higher abstraction level.

7.2.4 Differences in Plan Versus Execution

The plan was to find or create our own graphical engine but after discussions it was decided due to the time constraint and our domain that the auto layout of the *IDW* would be appropriate for this degree project. Having preconstructed configurations as starting points was also scratched due to the time constraint. Another feature that was scratched due to time constraints was the ability to see how much of an equipment unit was used. One of our greatest problems was the lack of time, the initial plan for this project was too large in scope, as this was a rather big project.

7.3 Future Development

The following is suggested future work.

- As was seen with the hide/show connections names and unused *ports* the ability to make a cleaner diagram was appreciated which suggests to continue in that direction by adding flags for hide/show mapping.
- Functionality that can be added is the ability to start with a prefabricated configuration instead of having to start from scratch.
- It would be good to be able to have the ability to add several *units* of the same type at once.
- It would be useful for the *DSL user*, having an option to have the *wizard* open so the *DSL user* can add connections for an example until he is done, could be added.
- To give the *DSL user* better overview. It would be useful to adding more layers so that the *functional units* and *connections* are grouped together and can be mapped together.
- It would be good if an *equipment unit* is showing if it is over or under utilized by color-coding the unit.
- It would be good if when clicking on a *port* it opens the appropriate *wizard*. That saves time, avoiding the multiple steps needed to take for connecting units.
- It would be good to solve the problem with the sort link in the *Type Catalogue view*. So that when sorting the references to the *types* sorted will still be correct.
- For usability testing, it would be good to create a test case with a large configuration to see how *Intentional* will handle that.
- It would be good if a delete popup was implemented, that shows what the user are about to delete. So that the *DSL user* does not delete the wrong part or delete anything by mistake.
- The possibility to zoom in and out to get better overview could be added. To make it easier for the *DSL user* to navigate in a configuration.

7.4 Conclusion

A basic prototype was developed which lets two types of users interact with the prototype. One user is in charge for defining and implementing the specifications for each *functional* and *equipment unit type*. The other user takes instances of the *unit types* the first user created, connecting them and mapping the functional layer on to the physical layer. In that way the two users are creating configurations which are displayed visually in graphs.

We think that the developed prototype is a good prototype for visualization and editing of configurations. It also has good validation controls which we noticed when building the example configurations. We are well aware that there are many improvements that can be done on the product. But since we had a limited amount of time, we had to decide what parts should be prioritised. We hope that the work done can be the foundation for future work on visualization and editing of configurations.

References

[1] http://www.ericsson.com/thecompany/company_facts/history (accessed 23 April 2014)

[2] http://www.ericsson.com/thecompany/company_facts (accessed 23 April 2014)

[3] http://www.ericsson.com/ourportfolio/telecom-operators/heterogeneous-networks (accessed 23 May 2014)

[4] A.Pleuss and G.Botterweck Visualization of variability and configuration options (Springer-Verlag 2012)

[5] Kobsa, A.: *User experiments with tree visualization systems.* In: INFOVIS '04: Proceedings of the IEEE
Symposium on Information Visualization, page.9-16 IEEE Computer Society, Washington, DC, USA (2004)

[6] Zhicheng Liu et al, Interactive visualization to enhance automated fault diagnosis in enterprise networks. Visual Analytics Science and Technology (VAST), 2010 IEEE Symposium on

[7] Lam.H, et al, Empirical Studies in Information Visualization: Seven Scenarios. In Visualization and Computer Graphics IEEE Trans. 12 juli 2012

[8] Herman. I et al : *Graph visualization and*
navigation in information visualization: a survey. IEEE Trans. Vis.
Comput. Graph. Vol 6 No1, page 24–43 (2000)

[9] http://en.wikipedia.org/wiki/Scrum_(software_development) (accessed 5 June 2014)

[10] M. Voelter et al, *DSL Engineering Designing, Implementing and Using Domain-Specific Languages* [E-book] (2013)

[11] http://www.intentsoft.com/intentional-technology/intentional-platform/ (accessed 23 April 2014)

[12] http://www.ericsson.com/thecompany/sustainability-corporateresponsibility/reducing-our-environmental-impact/energy-efficient-portfolio (accessed 29 May 2014)

Appendices

Appendix A

1. Vad saknas i programmet?

2. Vad var bra?

3. Hur lätt verkar programmet vara att använda?

mycket svårt	**svårt**	**okej**	**lätt**	**mycket lätt**

4. I jämförelse med nuvarande metoder hur verkar detta verktyg?

mycket sämre	**sämre**	**okej**	**bättre**	**mycket bättre**