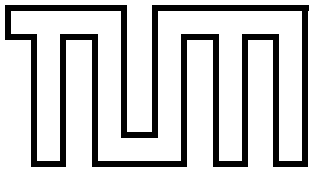


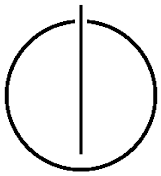
FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatik

Service Visualisation - Documentation

Michael Schott



Contents

Chapter 1

Introduction

The document describes the core functionality of the application "Service Visualisation" and gives a quick overview of the problem which the application is solving. For the java documentation please look in the repository.

1.1 Purpose of the System

The purpose of this tool is to shorten and simplify manual work. Instead of handling individual services, you can change entire service groups and their properties. In addition, the metrics and the visualization provide a better overview of the system and thus identify sources of errors and create service functions using service groups.

1.2 Current Sytem

In the current system, a lot is done manually. Services are compared individually and individual services are adjusted. Besides, there are no dependencies and no service characteristics in the current system.

Chapter 2

Requirements

A visonary scenario is presented in section 2.1.

2.1 Visionary Scenario

An existing system with several hundred or thousand services already uses the Framework Franca. The overview of the big picture has long since been lost. Based on the extension of Franca, the previous Franca Files are now extended and preset with the corresponding characteristics. Now you can visualize the architecture and get a better overview and notice that there are still some improvements needed. Therefore the characteristics of whole service groups are already adapted in the tool by means of drag and drop. Then the visualizations can be saved or a PDF can be created to share the knowledge about the services and service groups in the company. Due to the different grouping possibilities it is possible to get a good understanding also for many services with different views on the services. The application's tools help you to do this.

Chapter 3

Analysis

This chapter shows a simplified UML model, the Graphical User Interface and a Use Case Diagram.

3.1 UML Model

The UML model provides a rough overview of the structure and classes of the system. On the one hand the UI components, on the other hand the services and service groups with the grouping functionality. This UML model shows only a part of all classes and functionalities.

3.2 User Interface

TODO

3.3 Use Case Model

The use case model only gives a rough overview and does not include all use cases but is good to show that there are two types of users for the tool. Let's assume there are two types of users. A user, the service modeller, has access to all services and fidl files and has the possibility to make changes to them using the tool. The other user, the service

viewer, only needs an overview. This can be done by Service modeller by e.g. the vis. file, with which he then also sees the visualization.

Figure 3.1: UML Model - Part 1

/

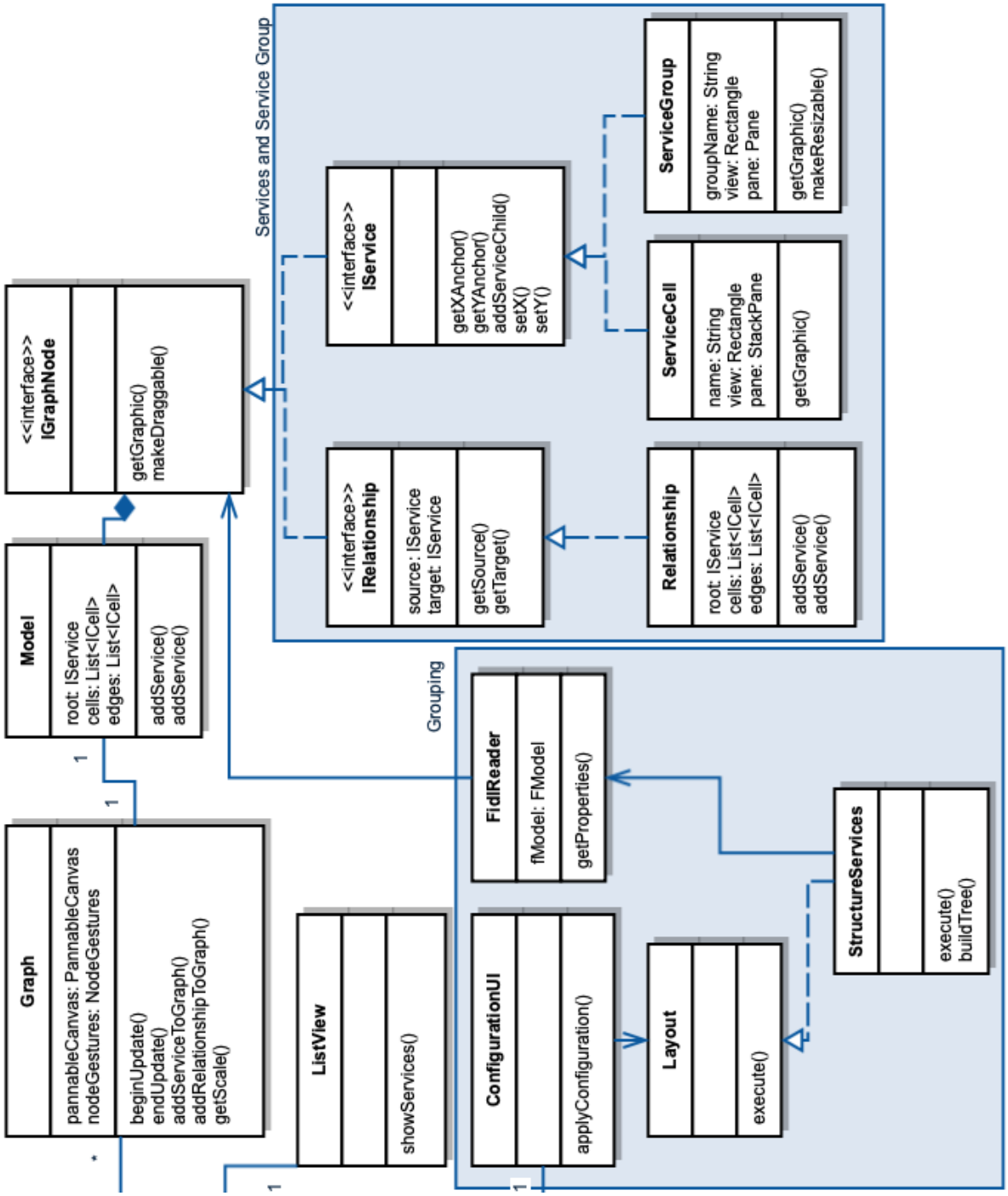


Figure 3.2: UML Model - Part 2 - UI

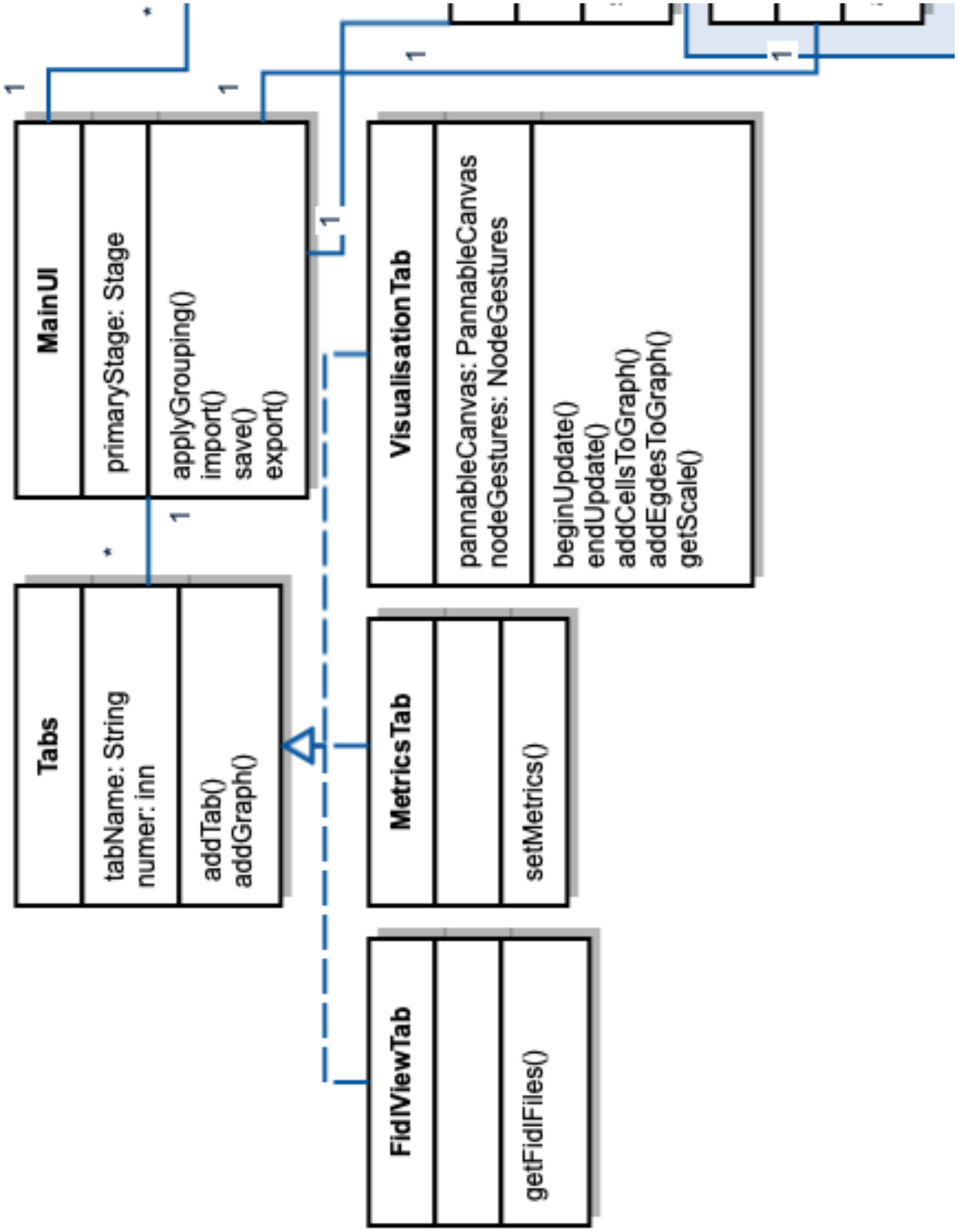
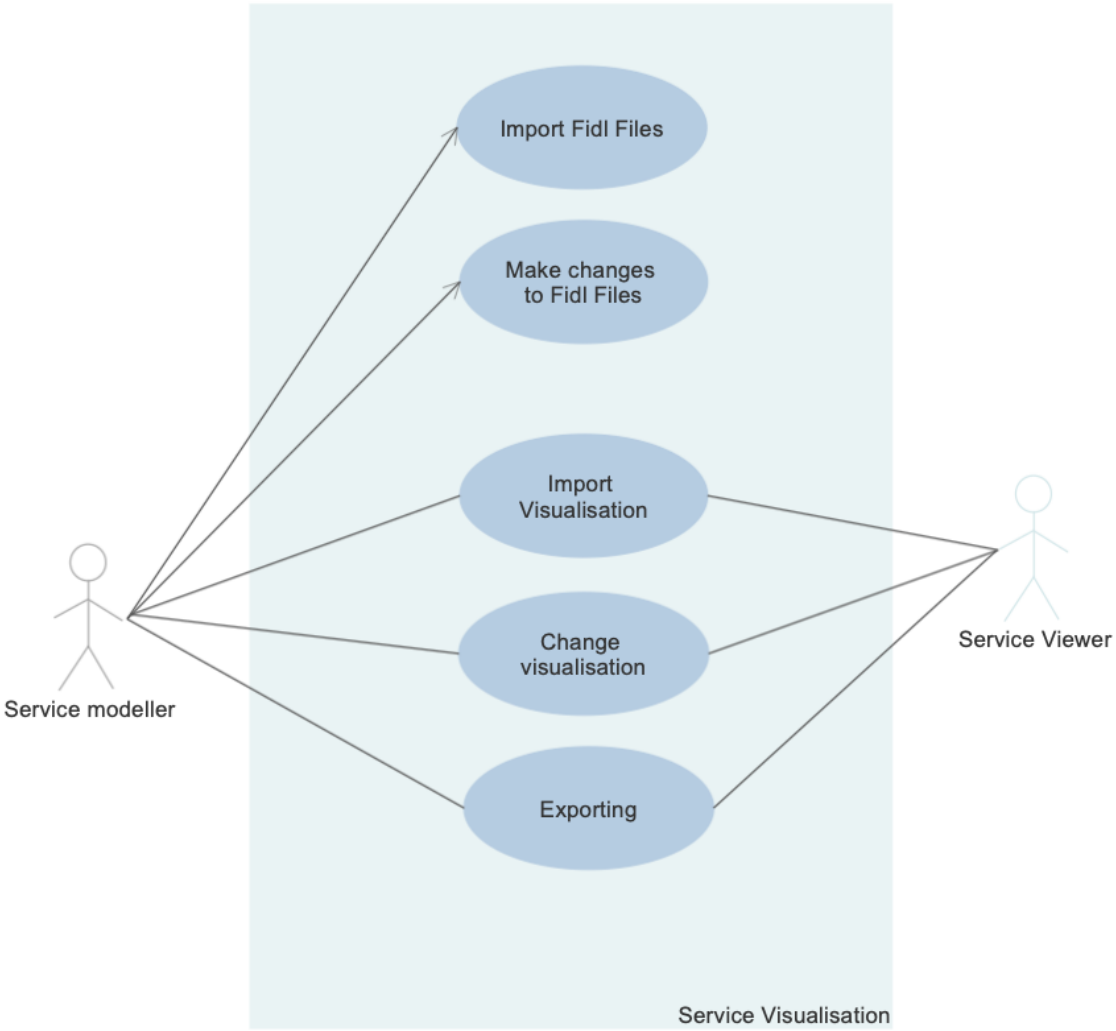


Figure 3.3: Use Case Model to visualize two kinds of users



Chapter 4

System Design

4.1 Persistent Data Management

To save the visualization, the data is serialized using Java's "serializable". The file format is self defined and .vis was chosen. The storage location is freely selectable and during import the files are deserialized and the graph with its services and service groups is created again.

Figure 4.1: Example of serializing

```
private void serialize() {

    try {
        FileChooser fileChooser = new FileChooser();
        fileChooser.getExtensionFilters().addAll(new ExtensionFilter("Visualisation Files", "*.vis"));
        fileChooser.setTitle("Save file");
        Tab tab = TabPaneSetter.tabPane.getSelectionModel().getSelectedItem();
        String string = "default";
        if (tab instanceof RenameableTab) {
            RenameableTab renTab = (RenameableTab) tab;
            string = ((RenameableTab) tab).name.get();
        }
        fileChooser.setInitialFileName("visualisation-"+ string  + ".vis");
        File savedFile = fileChooser.showSaveDialog(MainApp.primaryStage);
        FileOutputStream fileOut = new FileOutputStream(savedFile.getAbsolutePath());
        ObjectOutputStream out = new ObjectOutputStream(fileOut);
        out.writeObject(this);
        out.close();
        fileOut.close();
        System.out.printf("Serialized data is saved in " + savedFile.getAbsolutePath());
    } catch (IOException i) {
        i.printStackTrace();
    }

}
```

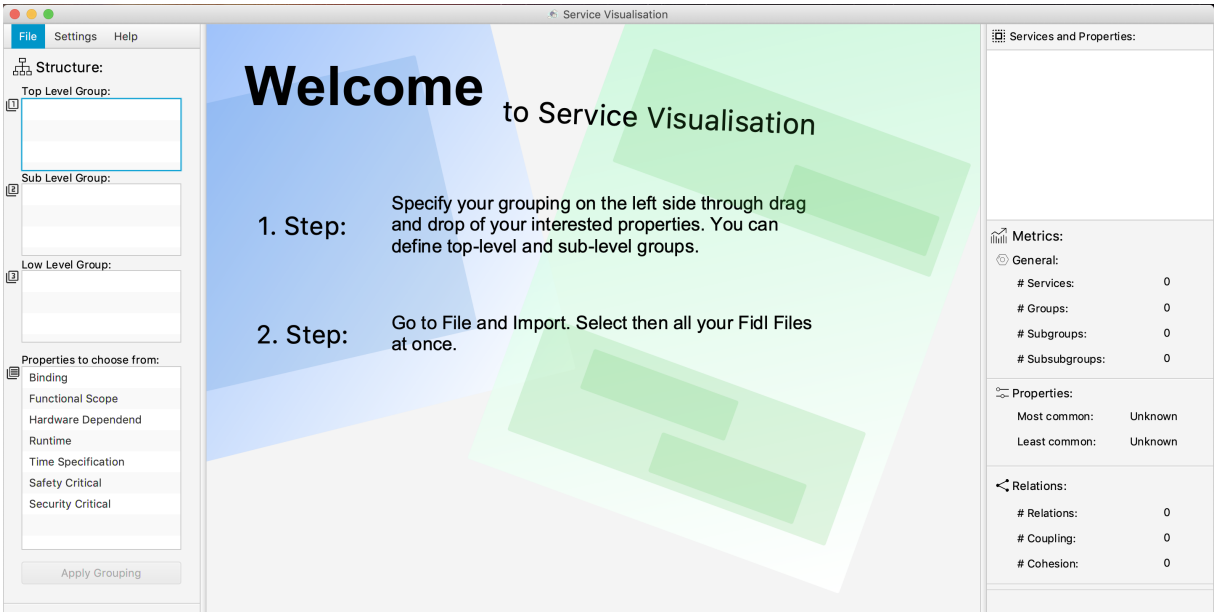
4.2 Demo Scenario

An example scenario is listed in this section.

Table 4.1: Example Scenario

Scenario Name	Scenario: Architecture creation (User wanting to review the service architecture and wants to create Service Groups to deploy on different computer nodes)	Screenshot
Participating actors	User (Architect, Sofware Developer...)	
Flow of events:	Opening the application	a)
	Importing Fidl Files	b)
	Configuring characteristics	c)
	Renaming visualisation tab	d)
	Changing groups and drag services	e)
	Compare with other visualization	f)
	Save visualization	g)
Entry conditions	Starting of the application	
Quality requirements	Everything works without any errors and feels intuitive	
Exit Conditions	User saves Visualisation	

Figure 4.2: a) Open application



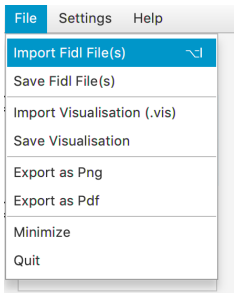


Figure 4.3: b) Import Files

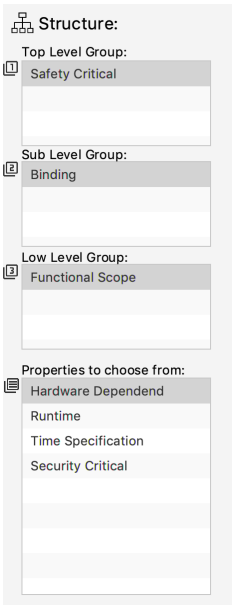


Figure 4.4: c) Configure Structure

Figure 4.5: d) Rename tab

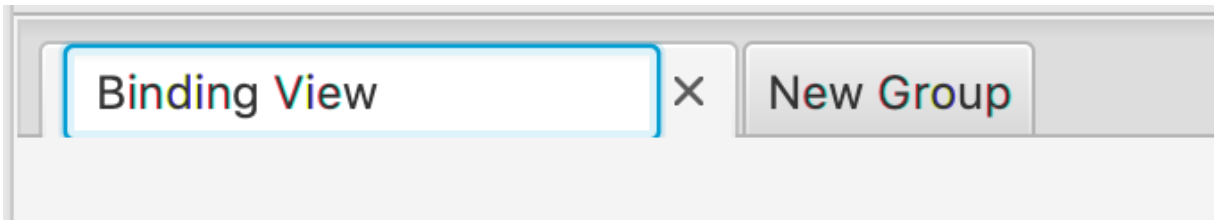


Figure 4.6: e) Rearrange services and groups

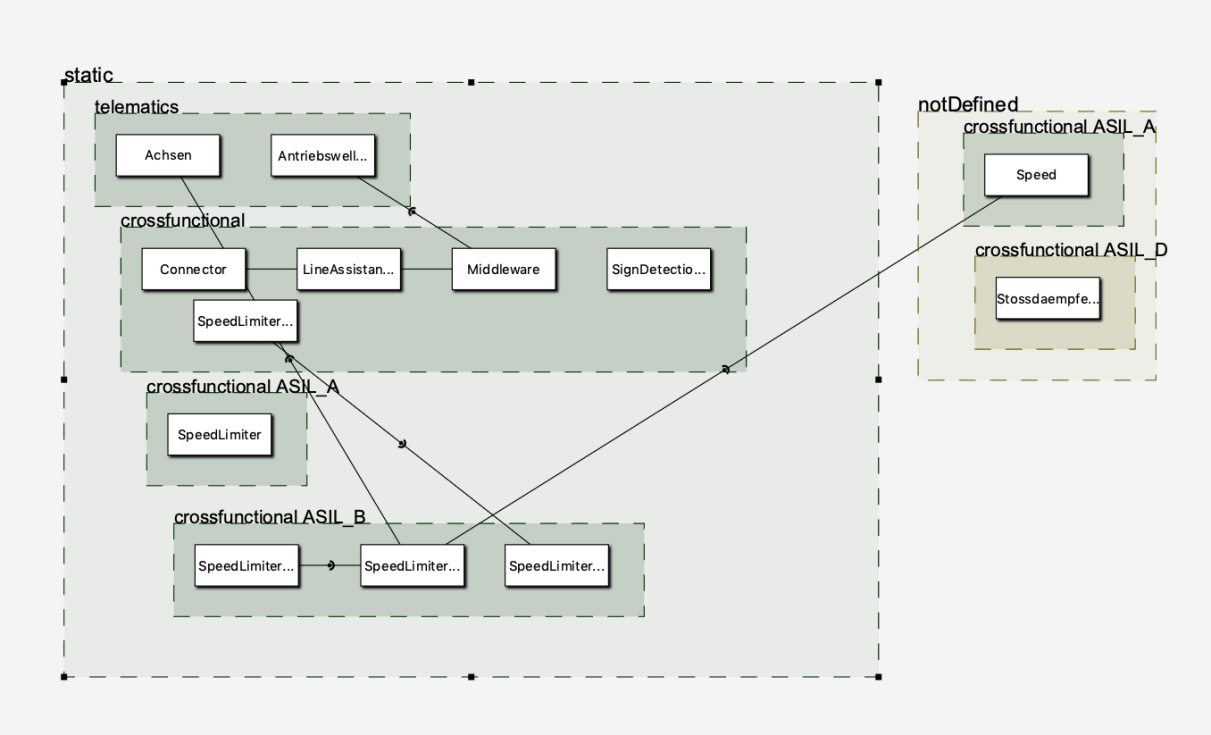
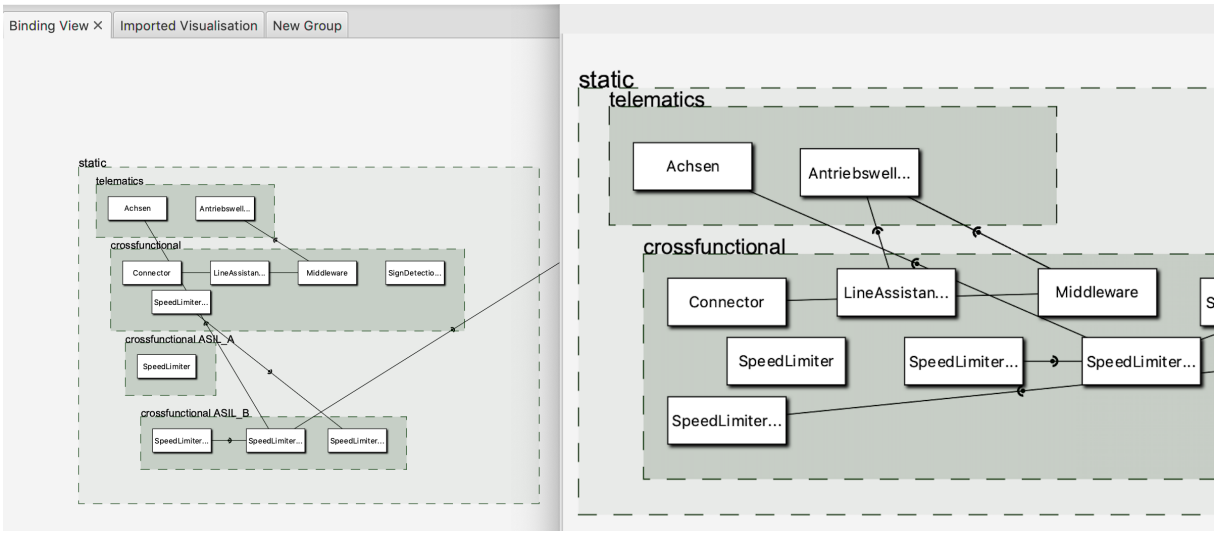


Figure 4.7: f) Compare



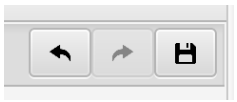


Figure 4.8: g) Save

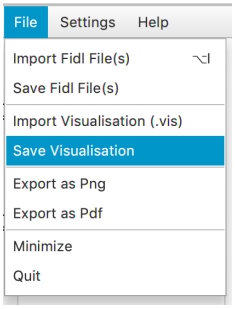


Figure 4.9: g) Save visualisation

Chapter 5

Extension and Building

This section deals with how to extend Franca and how to build the tool or the Franca extension.

5.1 Extension of new characterisitics

To see how to extend new characterisitcs to the application see the bachelors thesis document section 4.2.

5.2 Building the project

Unfortunately, both the extended Franca file and two EMF libraries are not available in Maven Central. Therefore, they must be added to the Maven local repository first. Since Maven had problems adding and executing the jars file in the pom specification, the external jar files have to be added manually. The Maven goal at org.eclipse.emf.common is as follows: *mvn install:install-file -Dfile=<location>DgroupId=org.eclipse.emf.common -DartifactId=org.eclipse.emf.common -Dversion=0.-13.1-SNAPSHOT -Dpackaging=jar* with location being the location of the jar file. This must be done for franca core, franca core dsl, org.emf.eclipse.common and org.emf.eclipse.ecore. After that should be done with *mvn clean compile assembly:single*. After that the built

and executable jar file should be in the folder.

5.2.1 Step by Step:

1. Install external jar to local repository.
   mvn install:install-file
   -Dfile=<location>
   -DgroupId=org.franca.core.dsl
   -DartifactId=org.franca.core.dsl
   -Dversion=0.13.1-SNAPSHOT -Dpackaging=jar

2. Install external jar to local repository.
   mvn install:install-file
   -Dfile=<location>
   -DgroupId=org.franca.core
   -DartifactId=org.franca.core
   -Dversion=0.13.1-SNAPSHOT
   -Dpackaging=jar

3. Install external jar to local repository.
   mvn install:install-file
   -Dfile=<location>
   -DgroupId=org.eclipse.emf.ecore
   -DartifactId=org.eclipse.emf.ecore
   -Dversion=2.12.0.v20160420-0247
   -Dpackaging=jar

4. Install external jar to local repository.
   mvn install:install-file
   -Dfile=<location>
   -DgroupId=org.eclipse.emf.common
   -DartifactId=org.eclipse.emf.common
   -Dversion=2.15.0.v20181220-0846
   -Dpackaging=jar

5. Build the jar.
   mvn clean compile assembly:single

6. Open the application.
   Open folder target and open "visualisation-jar-with-dependencies.jar".
   Done.

5.3 Building the extension of franca

To build the Franca extension, the Franca file with the extensions in EMF and xText is required. See Repository. There is the pom file under franca/releng/org.franca.parent. Execute it with the maven command *mvn clean install*. Afterwards you will find the corresponding jar files under the targets of plugins or features.

Chapter 6

Third Party Components

External components used in the application:

6.1 Main Dependency

1. itextpdf version 5.5.9
   Documentation `https://api.itextpdf.com`
   Used for exporting the pane to a pdf.

2. xtext version 2.18
   Documentation `https://www.eclipse.org/Xtext/documentation/`
   Important for franca, emf and for extending own domain specific language fidl with characteristics.

3. Eclipse EMF 2.15
   Documentation `https://www.eclipse.org/modeling/emf/`

4. Franca Core 0.13.1-SNAPSHOT
   Documentation `https://github.com/franca/franca`
   Main component of the tool which interface definition language was extended by characterisitics and service relationships.

6.2 Other Dependcies

5. Google inject 4.0
   Documentation `https://github.com/google/guice`
   Dependency necessary for Franca Core.

6. Google gson 2.8.5
   Documentation `https://sites.google.com/site/gson/gson-user-guide`
   Dependency necessary for Franca Core.

7. Maven 4.0.0
   Documentation `https://maven.apache.org/guides/index.html`
   For building the project.

8. JUnit 4.10
   Documentation `https://junit.org/junit4/`
   Dependency for testing.

Chapter 7

Known Issues and Workarounds

Sometimes, if you want to build the Franca expansion, Maven won't build it the second time and throws an error. The workaround is to let it down and build it again. When building with Eclipse, it might be that not only the file with the Pom file should be imported into Eclipse, but also the main requirements.

List of Figures