CHALMERS

Franca IDL C Code Generator

Development and Evaluation of New Tools for Franca IDL

Master of Science Thesis in the Programme
Computer Science – Algorithms, Languages and Logic

JESPER LUNDQVIST

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, October 2015

Franca IDL C Code Generator
Development and Evaluation of New Tools for Franca IDL

JESPER LUNDQVIST

Examiner: PATRIK JANSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden October 2015

Abstract

Franca IDL is an interface description language, commonly used in the automotive infotainment industry to define the interfaces of software components in a language-independent way. Code generators for the language are available, which convert Franca IDL interface files to server stub and proxy code in an implementation language, using the Remote Procedure Call paradigm and the D-Bus Inter-Process Communication system to enable communication between them.

The present code generators available for Franca IDL have several problems which affect the productivity of companies using them. These problems consist of heavyweight software dependencies, both in regard to the code generators and to the implementation code generated by them. The code generators also give very low-quality build feedback when used in automated build systems, making it difficult to find errors in the code.

In this thesis, FrancaCCG, a set of open source prototype code generators developed as part of this thesis work, is presented and evaluated. The prototypes successfully solve the identified problems of the old code generators, due to giving build feedback of better quality and having a minimum of external software dependencies. The code generated by FrancaCCG is runtime compatible with the code generated by the old code generators. FrancaCCG currently supports a subset of the Franca IDL and can be further developed to include more features of Franca IDL, further increasing its usefulness.

Keywords: Franca IDL, code generation, Interface Description Language, D-Bus, Inter-Process Communication, build feedback, automotive infotainment.

Acknowledgements

The author would like to express his sincere gratitude to his supervisor at Chalmers University of Technology, Mary Sheeran, for all her help and quality feedback given during the creation of this thesis. In addition, the author would like to thank his examiner, Patrik Jansson, for the valuable feedback provided. Furthermore, the author wish to express his utmost appreciation to his industry supervisors at Pelagicore, Jonatan Pålsson and Erik Botö, for all their invaluable help and advice given during the development and evaluation process of the software created as part of this thesis work. Finally, the author would like to give a heartfelt thank you to the rest of the employees at Pelagicore for all their support.

Jesper Lundqvist, Gothenburg, October 10, 2015

Contents

1 Introduction **1**
1.1 Current Franca IDL tooling 1
1.2 Aim of this thesis work 2
1.3 Report structure 2

2 Background **4**
2.1 Interface Definition Languages 4
2.2 Advantages and disadvantages of using an IDL 6
2.3 IDL annotations 6
2.4 Franca IDL 7
2.5 The D-Bus IPC 9
2.6 D-Bus XML Introspection 10
2.7 CommonAPI 11
2.8 Other code generators using Franca IDL 12
2.8.1 Default D-Bus XML Introspection 12
2.8.2 ipc-quartztime 13

3 Problems with current tools **14**
3.1 Software dependencies of the tool 14
3.1.1 Proposed metric 15
3.2 Error feedback in automated build systems 16
3.2.1 Proposed metric 18

4 Development of FrancaCCG **22**
4.1 Case study Franca IDL files 23
4.2 Development of first code generator 25
4.2.1 Annotations added to D-Bus XML Introspection 26
4.2.2 Output of first code generator 27
4.3 Development of second code generator 28
4.3.1 Software base of the code generator 28
4.3.2 D-Bus implementation choice 28

4.3.3 Enumerations in the second code generator 29
4.3.4 Output of second code generator 30
4.4 Compatibility with the CommonAPI tool 31
4.5 Resulting software suite 31
4.6 Code structure of FIDLtoXML 32
4.7 Code structure of XMLtoC 36
4.8 Testing of generated code 39

5 Results 41
5.1 P1: Software dependencies 41
5.1.1 Runtime dependencies of code generated by CommonAPI 41
5.1.2 Runtime dependencies of code generated by FrancaCCG 42
5.1.3 Runtime dependencies of the CommonAPI code generator 44
5.1.4 Runtime dependencies of the FrancaCCG code generator 45
5.2 P2: Error feedback 46
5.2.1 Summary 46
5.2.2 Test case 1: Missing or additional curly bracket 48
5.2.3 Test case 2: Import of non-existing file 49
5.2.4 Test case 3: Franca file structured wrong 49
5.2.5 Test case 4: Misspelled language construct 50
5.2.6 Test case 5: Usage of non-existing data type 50
5.2.7 Test case 6: Extension loop 51
5.2.8 Test case 7: More than one "out" section 51
5.2.9 Test case 8: No version defined 51
5.3 Compatibility with CommonAPI generated code 52
5.4 Assessment of FrancaCCG by Pelagicore 52

6 Discussion 55
6.1 Evaluation of the software dependencies of the tools 55
6.1.1 Runtime dependencies of the generated code 55
6.1.2 Runtime dependencies of the code generators 56
6.2 Evaluation of the error feedback of the tools 57
6.3 Evaluation of the compatibility with CommonAPI generated code 58
6.4 Future work 58

7 Conclusion 60
7.1 Resulting code generators 60
7.2 Personal experience 61
7.3 Availability and future of FrancaCCG 61

References **66**

A Source code for the test suite **I**
A.1 Original, error-free files II
A.2 Case 1: Missing curly bracket III
A.3 Case 2: Import of non-existing file III
A.4 Case 3: File structured wrong III
A.5 Case 4: Misspelled language construct III
A.6 Case 5: Usage of non-existing data type IV
A.7 Case 6: Circular dependency in extensions IV
A.8 Case 7: Several “out” sections in method definition IV
A.9 Case 8: No version defined V

B BNF grammar for a subset of Franca IDL **VI**

C Error feedback from test suite **X**
C.1 Test case 1: Missing curly bracket X
C.2 Test case 2: Import of non-existing file XII
C.3 Test case 3: File structured wrong XIII
C.4 Test case 4: Misspelled language construct XIV
C.5 Test case 5: Usage of non-existing data type XV
C.6 Test case 6: Circular dependency in extensions XVII
C.7 Test case 7: Several “out” sections in method definition XVIII
C.8 Test case 8: No version defined XIX

D Code generated by FrancaCCG from case study Franca IDL interface **XX**
D.1 MediaTypes.fidl XX
D.2 MediaIndexer.fidl XXI
D.3 MediaIndexer.xml XXII
D.4 MediaIndexer_common.h XXIII
D.5 MediaIndexer_proxy.h XXIV
D.6 MediaIndexer_proxy.c XXV
D.7 MediaIndexer_stub.h XXVIII
D.8 MediaIndexer_stub.c XXIX
D.9 MediaIndexer_proxyImplementation.c XXXII
D.10 MediaIndexer_stubImplementation.c XXXIV

Chapter 1

Introduction

An Interface Definition Language (IDL) is a type of specification language which is used to describe the interfaces of a software component, in a way that is independent of the actual implementation language used. Code generators can then be used to generate implementations of the interface detailed in the IDL files. There exist a large number of IDLs, both general and domain specific. In the automotive industry, the Franca IDL is widely used to describe component interfaces of infotainment systems. It has been proposed as an industry standard by the GENIVI Alliance, a grouping of over 150 companies working together to create an open-source, standardized, infotainment system platform [1]. GENIVI groups companies from the entire production chain of infotainment systems; from Original Equipment Manufacturers such as the BMW Group and the Volvo Car Corporation, to software developers such as Pelagicore and silicon manufacturers such as Intel.

1.1 Current Franca IDL tooling

Currently, the tooling of Franca IDL consists of an Eclipse toolkit installation in which C++11 code can be generated from Franca IDL files, using the CommonAPI tool [3]. This code corresponds to a server stub and proxy based on the Remote Procedure Call (RPC) concept of inter-process communication [9], using D-Bus [19] as Inter-Process Communication (IPC) system.

While Franca IDL is widely used by GENIVI Alliance members, its present tooling makes the language cumbersome to use due to several factors. It has very heavy-weight software dependencies due to requiring a specific installation of the Java Eclipse toolkit [7] as well as the CommonAPI tool. The requirement of having to use the Eclipse toolkit also makes it very difficult to use Franca IDL in automated build environments. The current automated building solution gives build feedback, such as compiler errors or warnings, of very low quality, making it hard for the user to find errors in the code.

1.2 Aim of this thesis work

The work conducted in this thesis aims to make Franca IDL more useful and easier to use, thus increasing productivity of GENIVI companies that make use of the language and its tooling. To accomplish this, a new set of tools for the Franca IDL will be developed, which should be affected as little as possible by the current problems described in this thesis. This set of tools will consist of new prototype code generators, runnable from the Linux terminal with a minimum of external dependencies. They should output C stub and proxy code using the D-Bus IPC. To preserve compatibility, the resulting code should be compatible at the D-Bus level with the code from the present CommonAPI code generator.

To evaluate the extent to which the tools developed as part of this thesis solve the identified problems of the CommonAPI tool, methods of evaluation of each of the problems will be defined. The new tools and the currently existing CommonAPI tool will then be evaluated using these methods.

This thesis work was conducted at Pelagicore, a Gothenburg/Munich company developing infotainment systems for the automotive industry and a member company of the GENIVI Alliance. Pelagicore employees use Franca IDL and its present tools on a daily basis in their workflow.

1.3 Report structure

The rest of the report is organized as follows. In the *Background* chapter following this introduction, the technical background of this thesis work is given. The following *Problems with current tools* chapter details the identified problems with the current tools and describes methods to evaluate potential new tools against these problems. The subsequent chapter named *Development of FrancaCCG* details the development of a new set of tools for Franca IDL. The *Result* chapter describes and evaluates the resulting code generators developed during this thesis work, as well as the old code generators, in regard to the identified problems. The following *Discussion* chapter provides further thoughts about the usefulness of the code generators and suggests potential future courses of action. Finally, the *Conclusion* chapter sums up this thesis work and its result.

Chapter 2

Background

2.1 Interface Definition Languages

Interface Definition Languages, also known as Interface Description Languages, are a family of specification languages used in software development to describe an interface independently of any implementation language or target system. They do not compile to executable code, in contrast to regular programming languages such as C or Java. Instead, there often exist several different compilers or code generators for a particular IDL, which produce server and client stub code corresponding to the interface file in the chosen implementation language [12]. These stubs enable communication over an Inter-Process Communication (IPC) system according to the interface specifications.

The notion of IDLs is not new; the first IDL was proposed in 1981 [27]. Since then, a multitude of IDLs have been developed. Today there exist many different IDLs, ranging from very general ones, such as Apache Thrift [14] and OMG IDL [22], to IDLs specific to a certain domain or industry.

The workflow of using interface definitions to generate server and client stub code using the Remote Procedure Call paradigm has also been well established for a long time [9] and is widely used. While the semantics of the generated stubs differ between different IDLs and IPCs, the general idea is the same. The same interface definition file (IDL file) is used to generate a server stub and server proxy. These use an Inter-Process Communication (IPC) system to communicate with each other. On the client side, a remote procedure call initialized by the client application will be directed to the local server proxy generated from the IDL file. This local proxy will then make use of an IPC to remotely communicate with the generated stub code on the server side. The stub on the server side will then relay the procedure call locally to the server application.

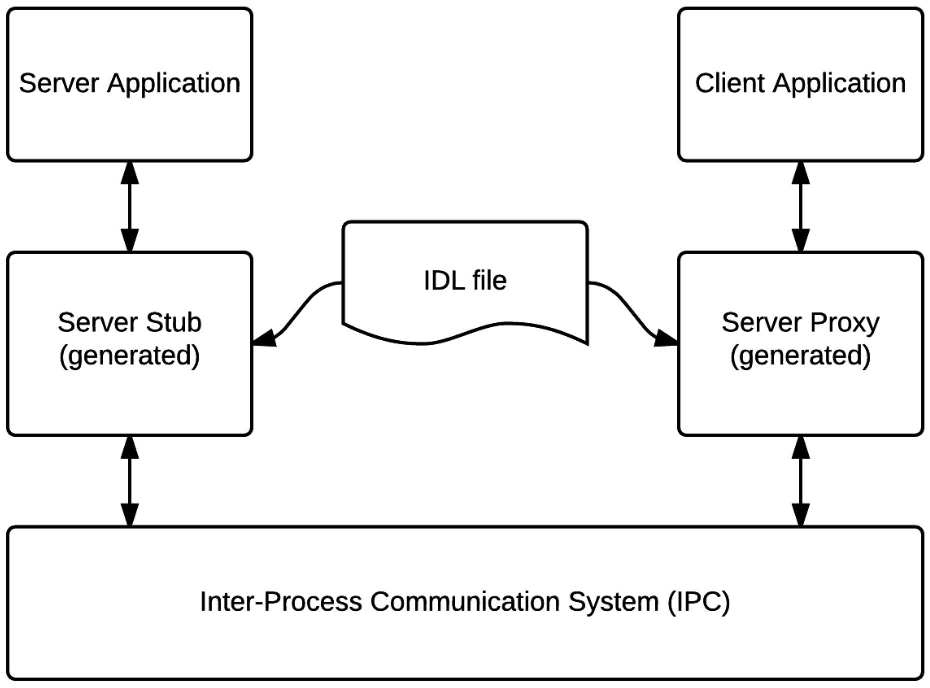


Figure 2.1: Flowchart depicting a typical Remote Procedure Call setup with server stub and proxy code, generated from an IDL file, using an IPC system to communicate. The server application and the client application will only communicate with the local server stub and proxy, respectively, and will thus be unaware of the IPC system.

2.2 Advantages and disadvantages of using an IDL

The main reason to use an IDL in your workflow is to allow for easier inter-process or inter-system communication, since an IDL allows an interface to be described independently of any implementation language or target system.

A major advantage of the server and client stub code generated from an IDL file is that both the client and the server application will be unaware of the remote nature of a remote procedure call. They will only communicate locally with the server proxy and stub, which then in turn handle all remote communication details. Thus, one could for example switch server implementation language without having to make any changes in any of the clients, as long as the new implementation of the server corresponds to the same IDL specification.

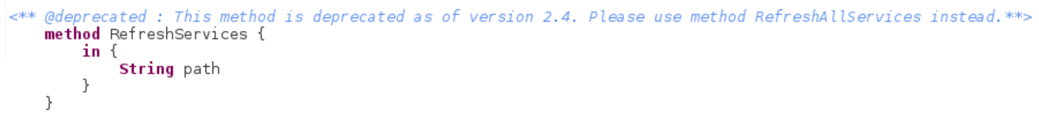
While there are many advantages of using an IDL in the workflow of software development, it imposes a significant increase in tool chain complexity [13]. It is thus important that the tools related to the IDL, such as the code generators, are easy to use and that they can be implemented in the workflow in a natural and non-obtrusive way.

The generality of IDLs can also be a source of problems [24]. While this generality is the main reason IDLs enable inter-system communication, it also places heavy limitations on the features that can be included in the IDL. Any interface specified in the IDL needs to be able to be properly expressed in one or more target implementation languages. Thus a more feature rich IDL puts greater constraints on which target languages it can potentially support.

2.3 IDL annotations

To preserve generality but still allow language-specific features, some IDLs are extended in their functionality by adding annotations or structured comments to the IDL code. These annotations can then be processed during code generation, and they essentially add functionality not present in the non-annotated form of the IDL. A similar approach is to "hide" extra information in the identifiers of IDL constructs such as parameters or methods. Both these approaches have the advantage that old code generators, unaware of the meaning of the annotations, can still process the code and maintain some degree of compatibility [23]. Such code generators will simply ignore these annotations, structured comments, and identifier names when processing the IDL code. Code generators that support the annotations will recognize them and process them during code generation.

Features such as documentation or version information are often added as annotations. Such annotations do not add any concrete functionality to the code, but rather help the user with tasks such as making the resulting code compatible with old versions of the interface, or generating documentation for the code. An example of such an annotation is presented in Figure 2.2. Here, a method definition in a Franca IDL file is marked with the `@deprecated : <explanation string>` annotation. Code generators aware of the meaning of the annotation can, for example, print an error message based on the explanation string each time the method is used, letting the user know the method is deprecated.



```
<** @deprecated : This method is deprecated as of version 2.4. Please use method RefreshAllServices instead.**>
    method RefreshServices {
        in {
            String path
        }
    }
```

Figure 2.2: A simple example of an annotation added to a method definition in the Franca IDL.

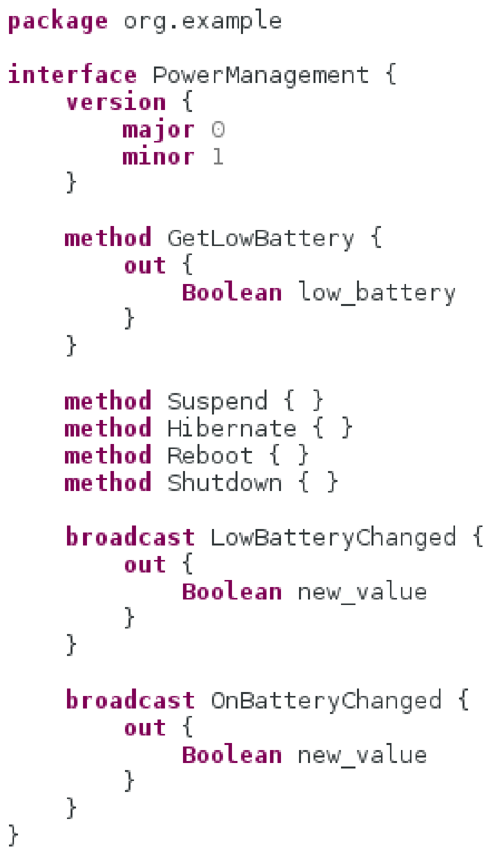
Other annotations add or change functionality of the resulting code. An example of this is the `org.freedesktop.DBus.Method.NoReply` annotation used in the D-Bus XML Introspection [30]. If this annotation is set to `True` in a method definition, the resulting method generated by a code generator aware of the annotation will not produce a reply when called.

2.4 Franca IDL

Franca IDL is primarily used in the automotive infotainment industry, where it has been proposed as a standard by the GENIVI Alliance [8]. However, it is a general IDL and its features are not domain specific. It can thus be used for a multitude of different projects and on many platforms. Developing interfaces in Franca IDL is primarily done in a special version of the Eclipse

Java toolkit. The file extension for Franca IDL interface files is `*.fidl`.

The initial release of Franca IDL was made in November 2011 [6]. As of August 2015, the language has been continuously developed from its initial release. Today, Franca IDL contains many advanced features, such as support for advanced data types and external type collections, as well as contracts describing the dynamic behavior of interfaces. The latest version of the language, 0.10.0, was released in July 2015 [5], fixing bugs and adding new features.



```
package org.example

interface PowerManagement {
    version {
        major 0
        minor 1
    }

    method GetLowBattery {
        out {
            Boolean low_battery
        }
    }

    method Suspend { }
    method Hibernate { }
    method Reboot { }
    method Shutdown { }

    broadcast LowBatteryChanged {
        out {
            Boolean new_value
        }
    }

    broadcast OnBatteryChanged {
        out {
            Boolean new_value
        }
    }
}
```

Figure 2.3: A simple example of a complete Franca IDL interface, containing several methods and broadcasts for a Power Management component.

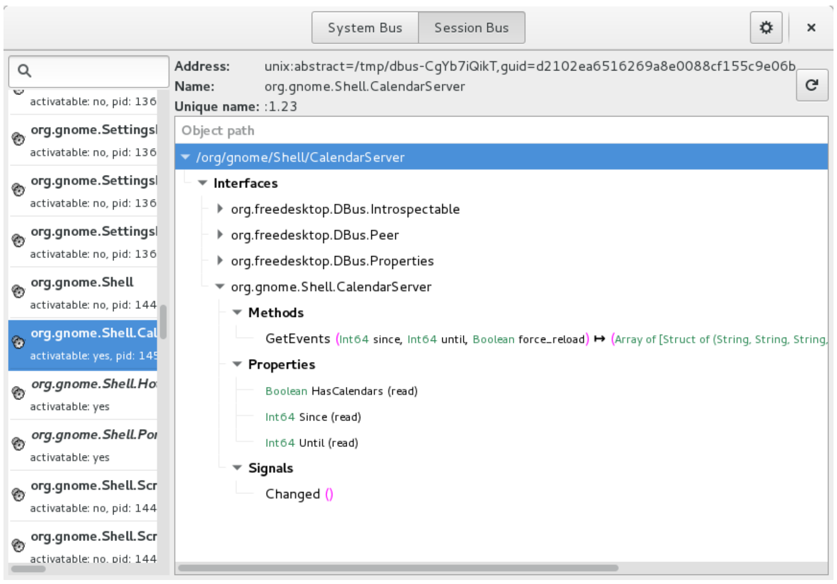
An example of a complete Franca IDL interface file can be seen in Figure 2.3. Here, an interface for a Power Management component is defined. The interface, belonging to the `org.example` package, begins with its current major and minor version, which is required to be stated in all Franca

IDL interface definitions. Afterwards, the five methods of the interface are detailed. None of these methods take any in-parameters, and all but one return no out-parameters. The `GetLowBattery` method has a boolean out-parameter. Finally, two broadcasts (or signals) are defined for the interface. These both broadcast a boolean value.

2.5 The D-Bus IPC

Inter-Process Communication systems, IPCs, are used to facilitate communication between two processes in a computer system. These processes can either both exist on a local machine, or they can communicate with each other over a network. There are a number of fundamentally different ways an IPC can work. For example, a process writing to a file in memory which another process subsequently reads from is one simple type of IPC system. Another option is to let the processes communicate using network sockets or using shared memory [41].

The D-Bus IPC is a popular IPC used in a multitude of projects, such as the GNOME project [35]. It uses several message buses to facilitate inter-process communication [26]. Daemon processes are used to control the buses. By contacting the daemon process, applications on the system can connect to the buses and create services on the bus in question, which other applications then can connect to. D-Bus services currently running on the system can be explored and listed by using the `d-feet` tool in Linux, shown in Figure 2.4. This can be used as a debugging tool when developing software using the D-Bus IPC, to make sure that a D-Bus service is running correctly.



```
activatable: no, pid: 136
org.gnome.SettingsI
activatable: no, pid: 136
org.gnome.SettingsI
activatable: no, pid: 136
org.gnome.SettingsI
activatable: no, pid: 136
org.gnome.Shell
activatable: no, pid: 144
org.gnome.Shell.Cal
activatable: yes, pid: 145
org.gnome.Shell.Ho
activatable: yes
org.gnome.Shell.Por
activatable: yes
org.gnome.Shell.Scr
activatable: no, pid: 144
org.gnome.Shell.Scr
activatable: no, pid: 144
```

```
Address:      unix:abstract=/tmp/dbus-CgYb7iQikT,guid=d2102ea6516269a8e0088cf155c9e06b
Name:         org.gnome.Shell.CalendarServer
Unique name:  :1.23
Object path
/org/gnome/Shell/CalendarServer
  Interfaces
    org.freedesktop.DBus.Introspectable
    org.freedesktop.DBus.Peer
    org.freedesktop.DBus.Properties
    org.gnome.Shell.CalendarServer
      Methods
        GetEvents (Int64 since, Int64 until, Boolean force_reload) ↦ (Array of [Struct of (String, String, String,
      Properties
        Boolean HasCalendars (read)
        Int64 Since (read)
        Int64 Until (read)
      Signals
        Changed ()
```

Figure 2.4: Screenshot showing the d-feet tool, used to inspect running D-Bus services.

2.6 D-Bus XML Introspection

The D-Bus IPC protocol includes D-Bus XML Introspection, which is a way to describe the behavior of a D-Bus interface. D-Bus Introspection can be seen as an IDL of sorts, since it specifies a D-Bus interface without any implementation details. There are several code generators for D-Bus XML Introspection available, for example `nih-dbus-tool` [40] and `gdbus-codegen` [34]. These output code for D-Bus stubs and proxies corresponding to the interface detailed in the introspection data, in different target implementation languages depending on the code generator used.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="introspect.xsl"?>
<node name="org.example">
    <interface name="PowerManagement">
        <version>0.1</version>
        <method name="GetLowBattery">
            <arg direction="out" name="low_battery" type="b">
            </arg>
        </method>
        <method name="Suspend"/>
        <method name="Hibernate"/>
        <method name="Reboot"/>
        <method name="Shutdown"/>
        <signal name="LowBatteryChanged">
            <arg direction="out" name="new_value" type="b">
            </arg>
        </signal>
        <signal name="OnBatteryChanged">
            <arg direction="out" name="new_value" type="b">
            </arg>
        </signal>
    </interface>
</node>
```

Figure 2.5: A simple example of D-Bus XML Introspection data, corresponding to the same interface as the Franca IDL interface in Figure 2.3

2.7 CommonAPI

CommonAPI is a software library developed by GENIVI, working as a communication middle layer between the IPC and the application using the library. Letting the application communicate with the CommonAPI library and not with the IPC directly makes the application IPC agnostic; you can dynamically switch the IPC used, without having to recompile the application.

In addition to this software library, CommonAPI also refers to a code generator for Franca IDL. This code generator is runnable from the Eclipse toolkit, and can produce both IPC agnostic code and code using the D-Bus IPC. Both versions produce C++11 code which uses the CommonAPI library. For the purposes of this thesis, the IPC agnostic version is not considered, since the D-Bus version of the code generator is used at Pelagicore. While the CommonAPI code generator supports most features of the Franca IDL language, advanced Franca IDL features such as contracts are not presently supported.

The CommonAPI code generator produces a set of C++ files corresponding to server stub and proxy code for the interface detailed in the Franca IDL file. This code can then be used to implement servers and clients for the interface. An example of a simple server implementation using this generated code can be found in Figure 2.6.

In this thesis, *CommonAPI* will refer to the *CommonAPI D-Bus code generator*, unless explicitly stated otherwise.

```
#include <iostream>
#include <thread>
#include <CommonAPI/CommonAPI.h>
#include "IndexerStubDefault.h"

int main(int argc, char** argv) {

    std::shared_ptr<CommonAPI::Runtime> runtime = CommonAPI::Runtime::load();
    std::shared_ptr<CommonAPI::Factory> factory = runtime->createFactory();
    std::shared_ptr<CommonAPI::ServicePublisher> servicePublisher = runtime->getServicePublisher();
    const std::string& serviceAddress = "local:org.genivi.mediamanager.Indexer:org.genivi.mediamanager.Indexer";
    std::shared_ptr<org::genivi::mediamanager::IndexerStubDefault> myService = std::make_shared<org::genivi::mediamanager
          ::IndexerStubDefault>();

    servicePublisher->registerService(myService, serviceAddress, factory);

    while(true) {
        std::cout << "Waiting for calls... (Abort with CTRL+C)" << std::endl;
        std::this_thread::sleep_for(std::chrono::seconds(60));
    }
}
```

Figure 2.6: Sample C++ code showing a simple server implementation using CommonAPI generated files.

2.8 Other code generators using Franca IDL

In this section, the most widely used code generators currently available for Franca IDL, other than the CommonAPI code generator described above, will be briefly presented.

2.8.1 Default D-Bus XML Introspection

The standard Franca IDL Eclipse installation includes a code generator capable of converting a Franca IDL file to a D-Bus XML Introspection file. This can be useful, since D-Bus is much more broadly used than Franca IDL and thus there are more code generators available for the language, capable of generating stubs and proxies in many different target languages. However, D-Bus XML Introspection only supports a subset of the Franca IDL language; many of Franca IDLs advanced features (such as contracts) are not available in D-Bus XML Introspection, and they are ignored by the code generator when processing a Franca IDL file. This, in combination with the code generator being runnable only from within the Eclipse toolkit, makes it have rather limited practical use.

2.8.2 ipc-quartztime

`ipc-quartztime` is an IPC library based on shared memory, mutual exclusion and signals [42]. It includes a code generator capable of parsing a subset of Franca IDL, producing client proxy and server stub code using the included IPC library. This resulting code is written in C. Due to using another IPC than D-Bus, the resulting code is not compatible with the code from the CommonAPI tool.

Chapter 3

Problems with current tools

In the pre-study of this thesis, two major problems with the CommonAPI code generation tool were identified by users of CommonAPI at Pelagicore: Its software dependencies are large, and it gives build feedback of very low quality in automated build systems. These problems affect the usability of the tool as well as the usefulness of the resulting code generated. Since there are no relevant alternatives to the CommonAPI tool currently available, the productivity of companies using Franca and the CommonAPI tool in their workflow can be negatively influenced [10]. Therefore, an alternative to the CommonAPI tool, such as the software developed as part of this thesis work, should be as little influenced by these two problems as possible, to increase productivity.

In this chapter, the two major problems identified in the pre-study will be presented, along with methodology for evaluating Franca IDL code generators against the problems.

3.1 Software dependencies of the tool

The first problem of the CommonAPI tool identified in the pre-study is defined as follows:

P1. The runtime dependencies for the CommonAPI code generator and its generated code have large sizes in kB.

Like most software, the CommonAPI tool uses and requires other software libraries to install and to use. However, in the case of CommonAPI, this set of additional software needed has a very large installation size, due to several factors. One such factor is the dependency on a heavy-weight Java and Eclipse toolkit installation. In some companies using the CommonAPI tool, such as Pelagicore, Java is not used for any other projects. This leads to a Java installation being required for the CommonAPI tool only.

While the problem P1 only states the size in kilobytes of additional software dependencies as a problem, it is also worth noting the general difficulty of installing the additional software needed. If the software dependencies of a tool are very difficult to install, it will impact productivity and in some cases might make the user choose another, inferior, tool instead. It is, however, very hard to measure the difficulty of installing a particular software, due to differences in the experience of the user installing the software as well as hardware and software differences of target computer systems. For the purpose of this thesis, only a general description of the difficulty of installing the software dependencies of a particular tool will be given when discussing said tool.

Related to the run-time software dependencies of the actual code generators is the required run-time dependencies of the output code generated by the code generators. Since Franca IDL is mainly used in the automotive infotainment industry, it is often used in embedded systems. This places heavy limits on the size of the additional software needed, since embedded systems often have limited physical memory. Thus, it is important that the additional software required by the code produced by a code generator for Franca IDL is lightweight and small.

3.1.1 Proposed metric

Finding the dependencies of a Linux binary file can be done with the `ldd <filename>` command, which produces a list of all additional libraries needed by the binary in question. This list can then be studied in detail and the command `du <filename>` can be used to find out the size of each software library.

The total number of kilobytes of software dependencies can be found by summing the size of all individual libraries in this way. However, there are some libraries that might be very large in size, but are universally used by a large number of applications, such as the `libc` library. Since it is likely that such a library will already exist on the target system due to some other already installed software using it, its inclusion in the metric can make such code seem more heavyweight than it actually is on the target system. Therefore, in addition to stating the total size of all necessary libraries, the larger ones should be separately discussed to determine if they are likely to be common to other applications running on the system. This should then be considered when comparing two different code generators.

However, this approach can only be taken on binary files, making it suitable to measure the dependencies of the code generated by the code generators in this way. The dependencies of the actual code generators can not always be measured in this way, though. This is due to some of them not being available as standalone binaries, but rather as applications written in interpreted languages such as Python, or available as plugins to other software, such as the CommonAPI code generator available as an Eclipse plugin. In these cases, the installation size of this software can be measured, and whether it is likely or not that the software is already available on the target system should be noted.

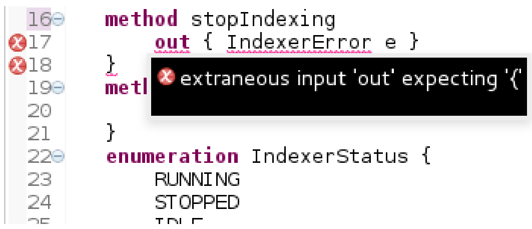
3.2 Error feedback in automated build systems

The second problem identified in the pre-study of this thesis is defined as follows:

P2. Automated building of Franca interfaces with the CommonAPI code generator is very difficult, and gives very low-quality feedback.

Writing code in any programming language is error prone. Thus it is important that any syntax errors in the code are caught by compilers or code generators. In contrast to compilers, code generators also need to handle run time errors such as a source file referring to a non-existing external file. When an error is found by a code generator, code generation is normally aborted, and an error message is given. To help the programmer identify the error, the error message given should contain enough information about the error so that it can easily be found and corrected by the programmer. For example, if some piece of code is missing a semi colon or a curly bracket, this should be indicated by the error message along with the location (e.g. line number) of the error.

These types of syntax errors are easily found and corrected when using the Eclipse toolkit to define Franca IDL interfaces, since they are clearly described to the user in a graphical way, as seen in Figure 3.1. However, it is not possible to use the graphical Eclipse toolkit to generate code in an automated build system, such as Yocto [38] or Jenkins [37]. In such a system, the tools used must be runnable from the command line, since the entire build process is automated. It is then important that any errors occurring during the build process can be easily found and logged.



```
method stopIndexing
    out { IndexerError e }
}
metl   extraneous input 'out' expecting '{'

}
enumeration IndexerStatus {
    RUNNING
    STOPPED
```

Figure 3.1: Error feedback given in the Eclipe Toolkit when a bracket is missing from a method declaration. Note that both the nature of the error and its exact location are shown.

The CommonAPI code generation tool is designed to be used from within the Eclipse Editor, and is normally not runnable at the command line. Since a command line interface is required to include CommonAPI in an automated build environment, a command line "hack" for CommonAPI has been developed by Pelagicore which is able to generate code from Franca IDL files [29]. However, this software is difficult to set up, and still requires the Eclipse toolkit installed on the machine. Furthermore, the build feedback given when using this command line version of CommonAPI is of very low quality, due to error messages and warnings normally visible in the Eclipse Editor not showing in the command line interface. Often, only a Java stack trace is given, which does not contain any useful information about the error. See Figure 3.2 for an example of such an error message.

It is therefore very hard for a programmer using the CommonAPI command line tool to debug the faulty code, since often the only information given is that there is at least one error somewhere in the code, but not its location or exact nature. When developing complex interfaces, finding such an unidentified error can be time consuming and impact productivity. For some errors, no error feedback at all is given, and faulty output files are generated. In this case, it is likely that the programmer does not even realize that an error has occurred during code generation, which can have disastrous consequences, unless the resulting code is properly tested.

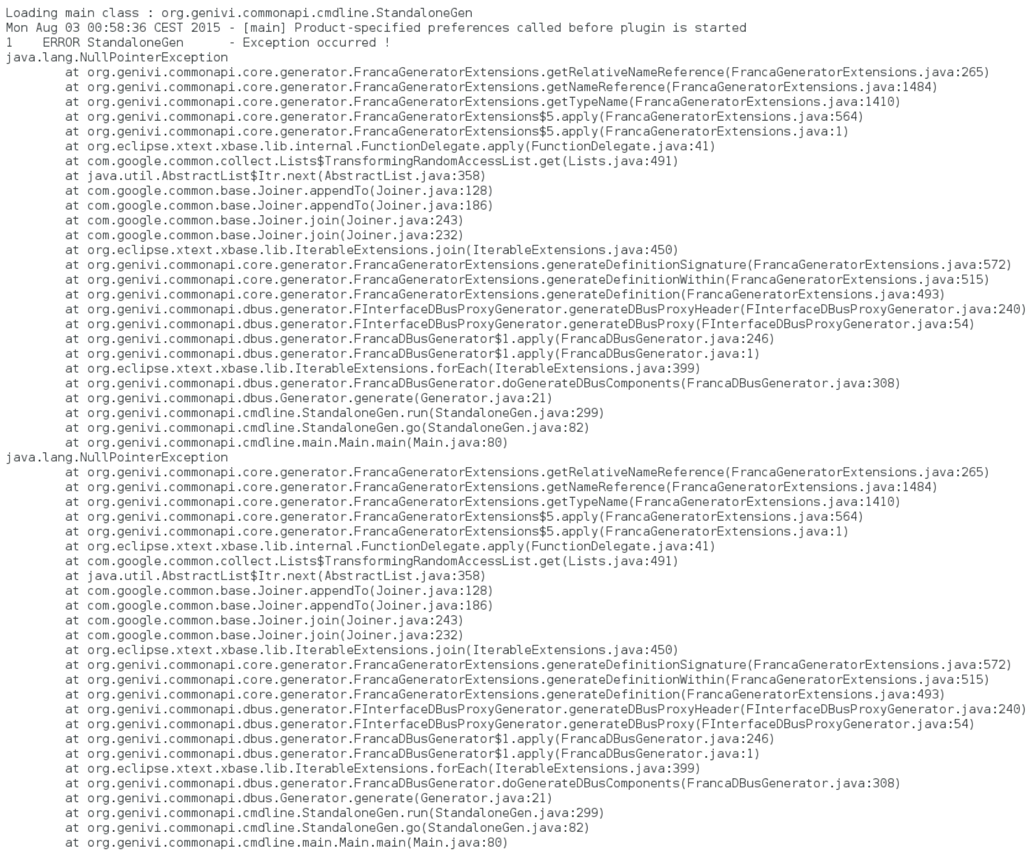
For a Franca IDL code generator to be usable in an automated build environment it should be runnable from the command line, and it should output useful build feedback. This feedback should be accessible on the `stderr` and `stdout` streams, which often correspond to the terminal window the code generator was run from [21].

3.2.1 Proposed metric

To measure the quality of the error feedback of a Franca IDL code generator, a test suite of eight faulty Franca IDL interface files was created. These files contained common syntax and run time errors. They were chosen based on common mistakes made when designing interfaces in Franca IDL, identified in collaboration with an employee at Pelagicore proficient in the use of Franca IDL [39]. For a complete list of these errors and a brief description of each, see Table 3.1. Each error was added to the case study interface defined in the *Development of FrancaCCG* section and saved separately, producing a set of eight Franca IDL interface files. The complete Franca IDL source code for each of these eight test cases is given in Appendix A.

What constitutes good build feedback is subjective. For the purpose of this thesis, the error message produced by a code generator when processing a faulty file is considered good enough if the nature of the error (e.g. "syntax error") is stated, along with the line number of the error, if applicable. The location of some errors in the code, such as *"missing external file"*, is easily found due to the reference to the file being in one location only. Other errors, such as *"missing curly bracket"*, have many possible locations in the code, and therefore the line number of the error should be given. Many syntax errors affect the subsequent line of code and not the actual line of the error. Thus, a fault marginal of ± 1 lines of code was allowed on the line number given, to allow for such an error message to still be considered good enough.

The total number of good error messages reported when running the test suite on a particular code generator for Franca IDL is proposed as a metric of the quality of the error feedback given for that particular code generator.



```
Loading main class : org.genivi.commonapi.cmdline.StandaloneGen
Mon Aug 03 00:58:36 CEST 2015 - [main] Product-specified preferences called before plugin is started
1    ERROR StandaloneGen      - Exception occurred !
java.lang.NullPointerException
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getRelativeNameReference(FrancaGeneratorExtensions.java:265)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getNameReference(FrancaGeneratorExtensions.java:1484)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getTypeName(FrancaGeneratorExtensions.java:1410)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions$5.apply(FrancaGeneratorExtensions.java:564)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions$5.apply(FrancaGeneratorExtensions.java:1)
        at org.eclipse.xtext.xbase.lib.internal.FunctionDelegate.apply(FunctionDelegate.java:41)
        at com.google.common.collect.Lists$TransformingRandomAccessList.get(Lists.java:491)
        at java.util.AbstractList$Itr.next(AbstractList.java:358)
        at com.google.common.base.Joiner.appendTo(Joiner.java:128)
        at com.google.common.base.Joiner.appendTo(Joiner.java:186)
        at com.google.common.base.Joiner.join(Joiner.java:243)
        at com.google.common.base.Joiner.join(Joiner.java:232)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.join(IterableExtensions.java:450)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinitionSignature(FrancaGeneratorExtensions.java:572)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinitionWithin(FrancaGeneratorExtensions.java:515)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinition(FrancaGeneratorExtensions.java:493)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxyHeader(FInterfaceDBusProxyGenerator.java:240)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxy(FInterfaceDBusProxyGenerator.java:54)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:246)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:1)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.forEach(IterableExtensions.java:399)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator.doGenerateDBusComponents(FrancaDBusGenerator.java:308)
        at org.genivi.commonapi.dbus.Generator.generate(Generator.java:21)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:299)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
java.lang.NullPointerException
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getRelativeNameReference(FrancaGeneratorExtensions.java:265)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getNameReference(FrancaGeneratorExtensions.java:1484)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getTypeName(FrancaGeneratorExtensions.java:1410)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions$5.apply(FrancaGeneratorExtensions.java:564)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions$5.apply(FrancaGeneratorExtensions.java:1)
        at org.eclipse.xtext.xbase.lib.internal.FunctionDelegate.apply(FunctionDelegate.java:41)
        at com.google.common.collect.Lists$TransformingRandomAccessList.get(Lists.java:491)
        at java.util.AbstractList$Itr.next(AbstractList.java:358)
        at com.google.common.base.Joiner.appendTo(Joiner.java:128)
        at com.google.common.base.Joiner.appendTo(Joiner.java:186)
        at com.google.common.base.Joiner.join(Joiner.java:243)
        at com.google.common.base.Joiner.join(Joiner.java:232)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.join(IterableExtensions.java:450)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinitionSignature(FrancaGeneratorExtensions.java:572)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinitionWithin(FrancaGeneratorExtensions.java:515)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinition(FrancaGeneratorExtensions.java:493)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxyHeader(FInterfaceDBusProxyGenerator.java:240)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxy(FInterfaceDBusProxyGenerator.java:54)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:246)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:1)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.forEach(IterableExtensions.java:399)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator.doGenerateDBusComponents(FrancaDBusGenerator.java:308)
        at org.genivi.commonapi.dbus.Generator.generate(Generator.java:21)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:299)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
```

Figure 3.2: Error feedback from the command line CommonAPI tool, corresponding to the same syntax error as in Figure 3.1. Note that neither the nature of the error (missing bracket) nor its location in the code is given in the verbose error message, thus making it difficult to find the error in the source code.

Table 3.1: List of common Franca IDL syntax and runtime errors

1.	Missing or additional curly bracket.	Like many other programming languages, Franca IDL uses curly brackets to define the scope of language constructs. Missing a closing bracket is a common programming error.
2.	Import of non-existing file.	You can import e.g. a type collection from an external Franca IDL file, but there might be errors during the process, such as when no file with the specified name is found.
3.	Franca file structured wrong.	Franca IDL file are required to have a certain structure. An example of this is that type definitions are required to be placed at the very end of a Franca IDL interface file, after any methods or attributes.
4.	Misspelled language construct.	This happens, for example, when the programmer writes "nethod" instead of the correct syntax "method".
5.	Usage of non-existing data type.	Using a non-existent type could both be caused by the programmer misspelling a built-in data type, or the programmer forgetting to include or define a custom data type.
6.	Extension loop for e.g. enumerations.	An extension loop happens when e.g. an enumeration definition extends another enumeration, which in turn extends the original one.
7.	More than one "out" section in method	A method definition in Franca is required to have at most one "out" section describing the return parameters of the method.
8.	No version defined.	All Franca IDL interfaces are required to include their major and minor version in the IDL file.

While very simple, this metric makes it straightforward to compare the error quality feedback of two different Franca IDL code generators in an objective way. In addition to this, cases where a code generator fails to find an error should be noted. This can be a serious problem, especially if faulty files are generated without giving any error message.

Chapter 4

Development of FrancaCCG

As part of this thesis work, a new set of code generators for Franca IDL was developed, with the aim that the identified problems of the old tools are not present in the new ones. In this chapter, the steps and decisions taken during the development of these new code generators are discussed.

The desired target implementation language of the code from the developed code generator was identified in the pre-study of this thesis as the C language, due to it being almost universally used, and easier to connect to components written in other language than the C++ language used in the CommonAPI code generator. However, to increase the general usefulness of the tools developed, the decision was taken to include D-Bus XML Introspection as an interim language. This would give two major advantages compared to developing only one code generator from Franca IDL directly to C. It would allow greater freedom in the use of the tools, since one could, for example, use another D-Bus XML Introspection code generator to output code in another implementation language, if desired. It would also allow the re-use of already existing code to parse D-Bus XML Introspection and generate code from it.

Thus, two independent code generators were to be developed; one that reads Franca IDL files and outputs D-Bus XML Introspection, and one that reads D-Bus XML Introspection and outputs C stub and proxy code. These were to be fully compatible with each other, i.e. the output of the first code generator must be valid as input to the second one. FrancaCCG (short for Franca IDL C Code Generator) was chosen as the name of this software suite.

4.1 Case study Franca IDL files

Franca IDL is a complex language with many features. Thus, it was not deemed practical to develop a set of new code generators capable of parsing and generating code for the entire Franca IDL in the limited time frame of this thesis work. Rather, a subset of Franca IDL was considered. This subset was defined based on a case study interface, used in a previous GENIVI project (see Figure 4.1). When the code generators could successfully process this case study, they were deemed feature complete, even though they did not support all features and constructs of Franca IDL. This scope was chosen so that the prototype code generators could be evaluated in regard to an actual industry project, while still being simple enough to be developed in the limited time frame of the thesis work.

The following features and constructs of Franca IDL were identified in the case study interface files. These were to be supported by the developed Franca IDL to D-Bus XML Introspection code generator. Features in italics were not present in the case study interface files, but were deemed critical enough to still warrant support in the code generators.

MediaTypes.fidl:

```
package org.genivi.mediamanager
typeCollection MediaTypes {
   enumeration BackendError extends MediaManagerError {
       BACKEND_UNREACHABLE }
   enumeration MediaManagerError { NO_ERROR }
}
```

MediaIndexer.fidl:

```
package org.genivi.mediamanager
import org.genivi.mediamanager.MediaTypes.* from "MediaTypes.fidl"
interface Indexer {
   version {
       major 1
       minor 0
   }
   <** @description: Example comment**>
   attribute IndexerStatus indexerStatus readonly noSubscriptions
   method getDatabasePath {
       out {
           String output
           IndexerError e
       }
   }
   method stopIndexing {
       out { IndexerError e }
   }

   method startIndexing {
       out { IndexerError e }
   }
   enumeration IndexerStatus {
       RUNNING
       STOPPED
       IDLE
   }
   typedef IndexerError is BackendError
}
```

Figure 4.1: Franca IDL case study files, used to limit the scope of the features in the code generators.

- Type collections
- Package names
- .fidl file import
- Structured Comments
  - @description
- *Regular comments*
- *Primitive types*
- Interfaces
  - Interface version (major/minor)
  - Methods
    - *Without in- or out-parameter(s)*
    - *With only in- parameter(s)*
    - With only out- parameter(s)
    - *With both in- and out-parameter(s)*
- Enumerations
  - Extending enumerations
- Attributes
  - Flags on attributes
    - readOnly
    - noSubscription
- Type definitions

4.2 Development of first code generator

The first code generator was to be based on the BNFC software suite [25], which takes a grammar file for a language as input, producing a parser, lexer, and skeleton code for a compiler for the language. The decision to use this software suite was made due to it being relatively easy to use and due to previous experience with the software. A Backus-Naur form (BNF) grammar for the subset of Franca IDL was constructed and can be found in Appendix B. Based on this grammar, skeleton code for a code generator was generated by BNFC. This code used the visitor pattern [28] to traverse a Franca IDL source file, and by modifying this skeleton code a code generator producing annotated D-Bus XML Introspection files was developed. BNFC supports several target languages of the generated skeleton code. C++ was chosen as a target language, due to it being the most time-efficient choice. Note that this language choice only affects the source code of the code generator, and not the code generated by it.

4.2.1 Annotations added to D-Bus XML Introspection

Most of the language features of the subset of Franca IDL in the case study files were supported by the standard D-Bus XML Introspection language. However, D-Bus lacks support for enumerations, and support for these was added to the language as annotations. The decision to add these features as annotations, and not as new XML tags, was made to preserve compatibility with other code generators unaware of these features. This essentially created a new dialect of D-Bus XML Introspection; there are other dialects available that are used in other projects, such as `eggdbus` [43].

An annotation in D-Bus XML Introspection is defined as a tuple consisting of a name string and a value string, in the form `<annotation name="" value=""/>`. As the name string, `com.pelagicore.FrancaCCodeGen.Enum.X.Y` was decided, with `X` being the name of the enumeration, and `Y` being the name of the enumeration member. The value was defined as the integer value corresponding to that particular enumeration member. See Figure 4.2 for an example of such an annotation.

```
<method name="getDatabasePath">
   <arg direction="out" name="output" type="s">
   </arg>
   <arg direction="out" name="e" type="u">
      <annotation name="com.pelagicore.FrancaCCodeGen.Enum.IndexerError.BACKEND_UNREACHABLE" value="0"/>
      <annotation name="com.pelagicore.FrancaCCodeGen.Enum.IndexerError.NO_ERROR" value="1"/>
   </arg>
</method>
```

Figure 4.2: Example of a method specified in D-Bus XML, where one of the arguments is an enumeration. Note that the argument is annotated with all allowed enumeration values.

Enumerations are commonly defined in implementation languages as non-negative integers. To prevent any ambiguity, the annotations were designed so that the integer value of each enumeration member is explicitly given. In Franca IDL, enumeration members can optionally be given an explicit integer value. In the Franca IDL to D-Bus XML Introspection code generator developed, enumeration members are given integer values starting at 0 and incrementing, skipping values that are either already given or explicitly defined in the Franca IDL file. During code generation, if an attribute or method argument is defined as an enumeration, that argument or attribute tag in the resulting D-Bus XML Introspection file is annotated with all possible enumeration values. While this approach creates some degrees of repetition in the D-Bus XML file if the same enumeration is used in multiple locations, it makes it very clear which enumeration members and values are supported each time an enumeration is used.

4.2.2 Output of first code generator

The FIDLtoXML code generator described above outputs one file when run. This file is an `*.xml` file containing the interface of the Franca IDL file used as input, converted to the D-Bus XML Introspection format. See Appendix D for an example of such a pair of a Franca IDL file and XML file generated from it.

4.3 Development of second code generator

4.3.1 Software base of the code generator

The second code generator developed as part of this thesis takes D-Bus XML Introspection files as input, outputting server stub and proxy code in the C language. Since D-Bus is a widely used IPC, a survey of existing code generators for D-Bus XML Introspection was conducted. The code generator was required to support the new annotations defined above, making it impossible to use an existing code generator without implementing support for these new annotations in it. Based on the survey, it was decided not to extend any existing code generator with new functionality, due to no suitable such code generators being identified in the survey. However, a code generator named `gdbus-codegen-glibmm`, developed in Python by Pelagicore employees and based on the `gdbus-codegen` code generator [34], was decided to form a code base for the second code generator. This code generator outputs C++ code using the `glibmm` library.

A major advantage of this choice to base the code generator on an existing code base was that an XML parser, as well as many helper functions, was given in the code base and could easily be reused in this project. In essence, only the actual code generated by the code generator had to be changed. Since the software was developed at Pelagicore, the readily available expertise there during this thesis work was another major advantage.

4.3.2 D-Bus implementation choice

While there exists an official D-Bus reference implementation and library, there are many alternatives available [20]. Some of these alternatives are bindings to the official library, `libdbus`. Others are independent implementations, sharing only the D-Bus protocol and technical specifications. When developing software using D-Bus, one must thus choose a suitable binding or reimplementation and this choice depends primarily on the programming language that will be used. For example, to implement D-Bus in a Haskell application, the `dbus-core` library can be used.

For this thesis work, a D-Bus implementation for the C language had to be chosen. Several candidates were identified.

- `libdbus`, the official D-Bus library. However, it is recommended not to use it in applications due to it being very low-level and cumbersome to use [19], and rather choose a binding for it instead [18].
- `GDBus`, which is an independent implementation of the D-Bus protocol developed by the GNOME foundation as part of the `GLib` library. It is highly documented and provides both high-level and low-level APIs [36].
- `DBus-GLib`, which is an older, deprecated D-Bus binding, also part of the `GLib` library [33].
- `libsystemd-bus`, the userspace version of the `kdbus` project, which aims to integrate D-Bus into the Linux kernel. It is an independent implementation of the D-Bus protocol, currently only available as part of `systemd` [31].

Chosen library

For this thesis work, the `GDBus` library was chosen as the D-Bus library to use for the C code of the second code generator. This library was chosen due to it having an active community of developers working on it, as well as being documented in a thorough way, making implementation easier.

4.3.3 Enumerations in the second code generator

No support for enumerations exists in the standard D-Bus XML Introspection format. Rather, an enumeration is normally defined as a regular integer by tools such as the Franca IDL to D-Bus XML Introspection code generator found in the standard Franca IDL installation. Using such tools, if an attribute or an method parameter in the Franca IDL file is defined as a enumeration, any integer will be allowed in the resulting code, not only the integers corresponding to the enumeration members of the enumeration type. This can lead to crashes or undefined behavior when using the generated code in an implementation.

By supporting enumerations in FrancaCCG and by giving the enumeration members explicit values, the programmer implementing the client or server of the interface can clearly see which integer values that are supported, and their corresponding enumeration name. The C code generated by the second code generator of FrancaCCG does not allow any other integer to be sent than the integers corresponding to the enumeration type in question.

4.3.4 Output of second code generator

The second code generator, XMLtoC, will generate seven files containing C code corresponding to the interface of the D-Bus XML Introspection file when run. These include a common header file, as well as three files corresponding to header, source and example implementation of the server stub used by the server, and three files corresponding to header, source and example implementation of the server proxy used by the client. These files, and their use, are specified in Table 4.1.

Table 4.1: Output C files from FrancaCCG

File name (prefixed by name of interface)	Description
_common.h	Common header file used by both proxy and stub, containing enumeration type definitions.
_stub.h	Header file used by the stub.
_stub.c	Stub code, used by server implementations of the interface.
_stubImplementation.c	Sample server implementation using the stub code generated. Can be used as skeleton code to build a functional server.
_proxy.h	Header file used by the proxy.
_proxy.c	Proxy code, used by client implementations of the interface.
_proxyImplementation.c	Sample client implementation using the proxy code generated. Can be used as skeleton code to build a functional client.

4.4 Compatibility with the CommonAPI tool

Since the D-Bus IPC is used both by the CommonAPI code generator and by FrancaCCG, compatibility of the resulting code of the code generators essentially means conforming to the same D-Bus interface as the code generated by CommonAPI. To check that the code is indeed compatible, the same Franca IDL `*.fidl` file can be used as an input to both code generators, producing two sets of stub and proxy server code. These two sets of code are written in different implementation languages, but are conforming to the same Franca IDL interface. By connecting the server stub generated by CommonAPI to the server proxy generated by FrancaCCG, compatibility can be easily checked, since the D-Bus daemon will not allow this connection unless they conform to the same D-Bus interface.

4.5 Resulting software suite

The two code generators described above together form the FrancaCCG software suite.

Included in the software suite is a shell script which can be used with a Franca IDL `*.fidl` file and optionally an output folder path as parameters. This shell script will then run `FIDLtoXML` on the Franca IDL interface file, converting it to a D-Bus XML Introspection file, annotated with new functionality not in the standard D-Bus XML Introspection. This `*.xml` file, and optionally the specified output folder path, will when be passed on to the `XMLtoC` code generator. This code generator parses the XML file and generates seven C files as output. Please see Appendix D for a example of such generated code, corresponding to the case study Franca IDL interface in Figure 4.1. Finally, the shell script will report the success of the code generation to the user by means of a message to the `stdout` stream.

In the case of an error during this process, code generation will be aborted, and the user will be alerted to the error by a message to the `stderr` error stream. The error feedback of FrancaCCG will be further discussed in the Results section.

Included in the FrancaCCG suite are also two shell scripts that are used to compile the code generators, as well as to clean up the folders after com-

```
./
├── cleanup.sh
├── compile.sh
├── FIDLtoXML
│   ├── CustomType.cpp
│   ├── CustomType.h
│   ├── CustomTypesParser.cpp
│   ├── CustomTypesParser.h
│   ├── franca.cf
│   ├── GenerateXML.cpp
│   ├── Makefile
│   ├── XMLGenerator.cpp
│   └── XMLGenerator.h
├── README.txt
├── run.sh
└── XMLtoC
    ├── codegen_main.py
    ├── codegen.py
    ├── config.py
    ├── dbustypes.py
    ├── __init__.py
    ├── parser.py
    └── utils.py
```

Figure 4.3: Files and folders making up the FrancaCCG code generator suite.

pilation. These scrips are named `compile.sh` and `cleanup.sh`, respectively. For a complete list of the files and folders of the FrancaCCG software suite, see Figure 4.3.

4.6 Code structure of FIDLtoXML

`CustomType.{cpp|h}` contain a class definition for a custom Franca type, which can currently be either an enumeration or type definition. It stores information about the custom type, such as enumeration members and their values, and also stores the D-Bus type signature corresponding to the custom Franca IDL type.

`GenerateXML.cpp` contains the main function of FIDLtoXML. This function will start by trying to read a `*.fidl` file specified as input. If successful, the file will be parsed by a parser for Franca IDL automatically generated by the BNFC suite, generating a parse tree. This tree is then used as input to the `findCustomTypes` function in `CustomTypesParser.{cpp|h}`. This function will traverse the parse tree, using the Visitor pattern, and create a list of all custom Franca IDL types defined in the file, as well as those defined in any imported files. The list of custom Franca IDL types is then parsed and processed, returning a list of CustomType objects, containing the D-Bus signature of each custom type. If a custom type cannot be successfully parsed, an error message is given and code generation is aborted. This can happen, for example, if an enumeration is extending another enumeration which cannot be found in the file.

If the automatically generated parser encounters any parse errors, for example due to syntax errors in the *.fidl file, it will give an error message listing the line number of the error using the standard cout function. To create more useful build feedback from this information, the output of the parser is redirected from stdout to a string, and the line number is subsequently saved. Afterwards, the output is redirected back to stdout, and the line of code corresponding to the given line number is found in the *.fidl file. This line of code is then printed to the stderr stream. This approach, shown in Figure 4.4, allows the programmer to easily see the line of code corresponding to the syntax error, without making any changes to the error feedback code of the automatically generated parser, which will be re-generated each time the code generator is built.

After producing the list of custom types, the parse tree is given as input to the generate function of XMLGenerator.{cpp|h}. This file is similar in structure to CustomTypesParser.{cpp|h}, using the same Visitor pattern to traverse the parse tree. In this file, the actual code generation from each Franca IDL contruct to its corresponding D-Bus XML Introspection data is done, using the list of custom Franca IDL types produced earlier as a dictionary when the D-Bus XML signature for a specific custom Franca IDL type is required. If a custom type is not found in this list, it has not been defined and an error message detailing this is given, and code generation is aborted.

The actual code generation to D-Bus XML is straightforward, due to the two IDL's sharing many features, and their general structure being the same. An example of this can be seen in Figure 4.5, which details the visitor functions used to generate code when processing a method definition in Franca IDL.

If the code generation is successful, the generated code will be saved to file, and the code generator will exit.

```
// Temporarily redirect cout during parsing, so that we can save
     the line number of any parse error.
std::streambuf* oldCoutStreamBuf = std::cout.rdbuf();
std::ostringstream strCout;
std::cout.rdbuf(strCout.rdbuf());

// Parse the fidl file
Program *parse_tree = pProgram(input);

// Restore old cout and save the contents of temp cout.
std::cout.rdbuf(oldCoutStreamBuf);
std::string parserOutput = strCout.str();

if (parse_tree) {

    // Generate XML file
    GenerateDBusXML *g2 = new GenerateDBusXML();
    g2->createCustomTypesList(parse_tree, pathToImportFile);
    output << g2->generate(parse_tree, pathToImportFile);
    output.close();
    std::cout << "FIDLtoXML successfully finished generating D-
        Bus XML Introspection for Franca IDL file " << argv[1] <<
         std::endl;

} else {

    // If there are parse errors, find the line number of the
        parse error and print that line
    size_t indexOfLineNbr = parserOutput.rfind("line ");
    if (indexOfLineNbr != -1) {
        // An error message containing a line number was found.
            Save the line number.
        std::string lineNbrStr = parserOutput.substr(
            indexOfLineNbr + 5, parserOutput.length());
        int lineNbr;
        std::istringstream (lineNbrStr) >> lineNbr;
```

Figure 4.4: Part of GenerateXML.cpp, showing code for redirecting `cout` to save output from automatically generated parser, and subsequently calling the `createCustomTypesList` and `generate` functions to generate XML code, if the parser did not produce any errors.

```
void GenerateDBusXML::visitDInMethod(DInMethod* p) {
    render("<method name=\"");
    visitIdent(p->id_);
    render("\">");
    increaseIndent();
    newIndLine();

    if (p->listinvari_) {
        p->listinvari_->accept(this);
    }

    removeLine();
    decreaseIndent();
    newIndLine();
    render("</method>");
    newIndLine();
}

void GenerateDBusXML::visitDInVar(DInVar* p) {
    render("<arg direction=\"in\" name=\"");
    visitIdent(p->id_);
    render("\" type=\"");
    p->type_->accept(this);
    render("\">");

    increaseIndent();
    newIndLine();
    renderEnumMembersIfNeeded();
    removeLine();
    decreaseIndent();
    newIndLine();

    render("</arg>");

    newIndLine();
}
```

Figure 4.5: Part of XMLGenerator.cpp, showing the visitor function for a Franca IDL method with in-parameters, as well as the visitor function for the actual in-parameters. Helper functions are used to keep indendation correct in the resulting D-Bus XML Introspection.

4.7 Code structure of XMLtoC

The XMLtoC code generator is based on the `gdbus-codegen-glibmm` code generator developed by Pelagicore, which in turn is based on the `gdbus-godegen` code generator available as part of the `glib` package. It is a Python application consisting of seven files.

`config.py, __init__.py, parser.py, utils.py` contain functions for parsing the XML file, as well as helper and initialization functions. Only minor changes to the `gdbus-codegen-glibmm` version of these files has been made.

`dbustypes.py` contain Python classes for the different constructs of the D-Bus XML Introspection format, such as methods, arguments, or interfaces. The tree-like structure of a D-Bus XML file is preserved due to each interface keeping a list of all methods and signals in the interface. An example of this can be seen in Figure 4.6. In the same way, methods keep a list of all arguments of the method in question.

```
class Interface:
    def __init__(self, name):
        self.name = name
        self.methods = []
        self.signals = []
        self.properties = []
        self.annotations = []
```

Figure 4.6: Code showing the `__init__` method of the interface class, initializing the lists of different constructs.

The main reason to save the constructs of the D-Bus XML Introspection file in this way is to provide an easy way to save implementation information about the construct, such as the C type corresponding to a particular argument in a method. It also allows easy iteration over, for example, all methods in an interface. This is very useful during code generation, when for example some code needs to be generated for each method in an interface.

Each class contain a post-process function which is called on all instances of the class before code generation. The post-process methods parse information about the D-Bus types into data useful during code generation. An example of this is the signature of methods. During code generation, the arguments of a method needs to be described in different ways, such as a string of comma-separated out-parameters or a concatenated string of the D-Bus signature of each in-argument. See Figure 4.7 for an example of the post-processing code for methods. By creating such strings in the post-process method, the actual code generation code is made cleaner and easier to follow.

```
sigListIn = []
sigListOut = []
argList = []
outArgList = []
inArgList = []
inArgSig = ""
outArgSig = ""
implSig = []
pointerSig = []
proxyHeaderInArgs = [""]
proxyHeaderOutArgs = [""]
proxyImplResults = [""]

for a in self.in_args:
    sigListIn.append(a.signature)
    argList.append(a.nameWithIndex)
    inArgSig = inArgSig + a.signature
    implSig.append(a.ctype_in + " " + a.nameWithIndex)
    pointerSig.append(a.ctype_in)
    proxyHeaderInArgs.append(a.ctype_in + " arg_" + a.name)
    inArgList.append("arg_" + a.name)
for a in self.out_args:
    sigListOut.append(a.signature)
    argList.append("&" + a.nameWithIndex)
    outArgList.append(a.nameWithIndex)
    outArgSig = outArgSig + a.signature
    implSig.append(a.ctype_out + " *" + a.nameWithIndex)
    pointerSig.append(a.ctype_out + "*")
    proxyHeaderOutArgs.append(a.ctype_out + " *out_" + a.nameWithIndex)
    proxyImplResults.append("&" + a.nameWithIndex + "_result")

sigStr = "__" + "_".join(sigListIn) + "__" + "_".join(sigListOut)
self.proxy_results_addresses = ", ".join(proxyImplResults)
self.proxy_header_inarg_string = ", ".join(proxyHeaderInArgs)
self.proxy_header_outarg_string = ", ".join(proxyHeaderOutArgs)
self.pointer_signature = ", ".join(pointerSig)
self.implementation_signature = ", ".join(implSig)
self.argument_string = ", ".join(argList)
self.camel_name_with_dbus_signature = self.name + sigStr
self.capital_name_with_dbus_signature = self.name.upper() + sigStr.upper()
```

Figure 4.7: Code snipped showing part of the post-processing code for methods. Here, different string representations of the method is created, and subsequently used during code generation.

Since the output language is changed to C, `dbustypes.py` is heavily rewritten compared to the `gdbus-codegen-glibmm` version, which is used to output C++ code and thus contains C++ data about the D-Bus types instead.

`codegen.py` contains the actual code generation methods. This file is almost completely rewritten and only shares its general structure and some helper functions with the `gdbus-codegen-glibmm` version. The file contains a code generation method for each of the seven output files, and a general `generate` method which will call each of the seven file-specific methods in turn.

The code generation methods all share the same general structure. Each output file is generated from top to bottom, using the interface instances generated by the parser earlier as input. By iterating over the constructs of the interfaces, all methods, arguments, properties, and signals can be accessed. By using the `format(**locals())` pattern to enable the post-processed data to be easily used. See Figure 4.8 for an example showing part of the proxy code generation method.

```
def generate_proxy_c(self):
    headerFileName = self.proxy_h.name.rsplit("/", 1)[1]
    self.emit_c_p (dedent('''\
    #include "%s"
    #include <stdio.h>
    #include <stdlib.h>
    ''' % headerFileName))

    for i in self.ifaces:
        for m in i.methods:
            self.emit_c_p(dedent('''
            void {i.camel_name}_{m.camel_name_with_dbus_signature}(GDBusProxy *proxy{m.proxy_header_inarg_string},
                   const GAsyncReadyCallback callback) {{

                g_dbus_proxy_call(
                    proxy,
                    "{m.name}",
                    {m.new_in_arguments_gvariant},
                    G_DBUS_CALL_FLAGS_NONE,
                    -1,
                    NULL,
                    callback,
                    NULL);
            }}

            void {i.camel_name}_{m.camel_name_with_dbus_signature}_finish (GDBusProxy *proxy{m.
                   proxy_header_outarg_string}, GAsyncResult *result, gboolean *success) {{
                // Get result from method call from d-bus proxy
                GVariant *wrapped;
                GError *error = NULL;
                wrapped = g_dbus_proxy_call_finish(proxy, result, &error);

                if (error != NULL) {{
                    printf("WARNING: Method call to {m.camel_name_with_dbus_signature} did not succeed.\\nGError content
                          : %s\\n", error->message);
                    *success = FALSE;
                }} else {{

                    // Put results from method call in parameter
            ''').format(**locals()))
            for a in m.out_args:
                indexOfArg = str(m.out_args.index(a))
                self.emit_c_p('''      GVariant *{a.nameWithIndex}_variant;'''.format(**locals()))
                self.emit_c_p('''      {a.nameWithIndex}_variant = g_variant_get_child_value(wrapped, {indexOfArg});'''.
                      format(**locals()))
                self.emit_c_p('''      *out_{a.nameWithIndex} = {a.g_variant_getter}({a.nameWithIndex}_variant{a.
                      g_variant_getter_extra_arguments});'''.format(**locals()))
                self.emit_c_p(''''''.format(**locals()))
            self.emit_c_p('''      *success = TRUE;'''.format(**locals()))
            self.emit_c_p('''  }''')
            self.emit_c_p('''}''')
```

Figure 4.8: Part of the code generation method for the server proxy file. Note the usage of post-processed data in the strings emitted.

`codegen_main.py` contains the main function of XMLtoC. Here, the specified XML files are first parsed, creating a list of interfaces, using the interface class from `dbustypes.py`. Afterwards, the interfaces are post-processed, which also post-processes the methods, arguments, and signals. Afterwards, the seven output C files are opened, and the `generate` function of `codegen.py` is called, generating the C code into the opened files. If successful, the files are closed, and the application exits. Code generation using the FrancaCCG suite is then completed.

4.8 Testing of generated code

The design of the code generated by XMLtoC was an important part of the development process of FrancaCCG. The generated code should be functional and free from any bugs or errors, and the server proxy code generated should be fully compatible with the server stub code generated from the same interface file. During the development of XMLtoC, the correctness of the generated code was tested by implementing support for D-Bus constructs and features in an iterative way, creating test cases to make sure each new feature worked well before adding support for another feature.

The XML file corresponding to the Franca IDL case study interface was generated by the FIDLtoXML code generator, and the D-Bus features present in the file were identified and roughly sorted by order of implementation complexity. The list of identified D-Bus features in the case study file is presented in Figure 4.9. However, care was taken to expand some features that were deemed too simple in the case study file. For example, no method in the case study file had any in-arguments, and support for such methods was deemed important even though they were absent in the case study file. Thus, the feature was added to the list.

- Empty D-Bus interfaces
- Primitive types
- Methods without arguments
- Methods with in-arguments
- Methods with out-arguments
- Properties
- Property access modifiers
- Annotation: NoSubscriptions on properties
- Annotation: Enumerations

Figure 4.9: D-Bus features to be supported by the XMLtoC code generator.

Code generation support for a new construct or feature in this list was then added to the XMLtoC code generator, and test D-Bus XML Introspection files containing these constructs were manually created. Afterwards, the XMLtoC code generator was run using the test files as input, and simple debug code was added to the generated implementation files to provide visual feedback showing if the new feature was working as intended or not. After compiling and running the generated server and client implementations, the visual feedback was studied. If the new feature was working as intended, another feature was implemented using the same procedure, until all features identified in the case study XML file had been added.

While this made sure all features in the case study interface were supported, it is possible that edge cases were not covered well enough, since test files were manually created for each new feature. Other test solutions were considered, such as automated generation of test cases by applications such as QuickCheck [11]. However, while such an approach would have been useful, the limited manual testing was deemed enough to cover the scope of this thesis, in which support for the case study interface was most important.

Chapter 5

Results

In this chapter, the results of the evaluation of the CommonAPI code generator and the code generator suite developed as part of this thesis work, FrancaCCG, is presented. The code generators are evaluated against the two identified problems described in the *Problems with current tools* chapter. At the end of this chapter, an assessment of FrancaCCG made by a Pelagicore employee is presented.

The results in this chapter were produced on a Linux system running the Debian version 8.1 64-bit operating system.

5.1 P1: Software dependencies

5.1.1 Runtime dependencies of code generated by CommonAPI

The `ldd` command was executed on a simple binary implementation of the files generated by the CommonAPI command line tool. The results can be seen in Figure 5.1. For each entry in the list shown, the command `du <library name>` was used to find out the size of each library. The resulting list of libraries with their size in kilobytes can be seen in Table 5.1. Some libraries were found to be common to both code generators; these are marked in italics. The total sum in kilobytes for the runtime dependencies of this implementation was found to be 11116 kB, with the two CommonAPI libraries contributing to 6736 of these kB.

```
linux-vdso.so.1 (0x00007ffc36367000)
libCommonAPI.so.3 => /usr/local/lib/libCommonAPI.so.3 (0x00007f7dccfb6000)
libCommonAPI-DBus.so.3 => /usr/local/lib/libCommonAPI-DBus.so.3 (0x00007f7dcc9e2000)
libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007f7dcc6d7000)
libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f7dcc4c1000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f7dcc118000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f7dcbf14000)
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f7dcbc13000)
libdbus-1.so.3 => /lib/x86_64-linux-gnu/libdbus-1.so.3 (0x00007f7dcb9ca000)
/lib64/ld-linux-x86-64.so.2 (0x00007f7dcd20f000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f7dcb7ad000)
```

Figure 5.1: Example output of `ldd` on a generated binary from CommonAPI.

Table 5.1: Size of runtime dependencies of CommonAPI generated files

Library name	Size in kB
libCommonAPI.so.3	536
libCommonAPI-DBus.so.3	6200
libstdc++.so.6	988
libgcc_s.so.1	88
libm.so.6	1028
libdbus-1.so.3	292
libc.so.6	*1692*
libdl.so.2	*16*
ld-linux-x86-64.so.2	*140*
libpthread.so.0	*136*
TOTAL	**11116**

5.1.2 Runtime dependencies of code generated by FrancaCCG

As for the CommonAPI tool, the `ldd` command was executed on a simple implementation generated by FrancaCCG for the same case study interface. The results can be seen in Figure 5.2. For each entry in the list shown, the command `du <library name>` was used to find out the size of each library. The resulting list of libraries with their size in kilobytes can be seen in Table 5.2. Libraries common to both CommonAPI and FrancaCCG generated code are marked in italics. The total sum in kilobytes for the runtime dependencies of this implementation was found to be 5720 kB.

```
linux-vdso.so.1 (0x00007fffa13e5000)
libgio-2.0.so.0 => /usr/lib/x86_64-linux-gnu/libgio-2.0.so.0 (0x00007f3bdc0da000)
libgobject-2.0.so.0 => /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0 (0x00007f3bdbe88000)
libglib-2.0.so.0 => /lib/x86_64-linux-gnu/libglib-2.0.so.0 (0x00007f3bdbb79000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f3bdb95c000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f3bdb5b3000)
libgmodule-2.0.so.0 => /usr/lib/x86_64-linux-gnu/libgmodule-2.0.so.0 (0x00007f3bdb3af000)
libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007f3bdb194000)
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f3bdaf6f000)
libresolv.so.2 => /lib/x86_64-linux-gnu/libresolv.so.2 (0x00007f3bdad58000)
libffi.so.6 => /usr/lib/x86_64-linux-gnu/libffi.so.6 (0x00007f3bdab50000)
libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007f3bda8e2000)
/lib64/ld-linux-x86-64.so.2 (0x00007f3bdc453000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f3bda6de000)
```

Figure 5.2: Example output of `ldd` on a generated binary from FrancaCCG.

Table 5.2: Size of runtime dependencies of FrancaCCG generated files

Library name	Size in kB
libgio-2.0.so.0	1504
libgobject-2.0.so.0	328
libglib-2.0.so.0	1084
libgmodule-2.0.so.0	16
libz.so.1	108
libselinux.so.1	140
libresolv.so.2	84
libffi.so.6	32
libpcre.so.3	440
libpthread.so.0	*136*
libc.so.6	*1692*
ld-linux-x86-64.so.2	*140*
libdl.so.2	*16*
TOTAL	**5720**

5.1.3 Runtime dependencies of the CommonAPI code generator

Due to not being available as a standalone binary, but rather as a plugin to a specific edition of the Eclipse Toolkit, the `ldd` approach can not be taken when measuring the runtime dependencies of the CommonAPI code generator. Instead, one can study the Eclipse Toolkit used. For many companies using Franca IDL, an Eclipse and Java installation is only used to run the CommonAPI code generator and no other software.

A fresh Eclipse Toolkit installation with the CommonAPI code generator was made on the target system, following the installation instructions in the CommonAPI Tutorial [2]. The size of the installation folder of Eclipse was then found to be 550328 kB.

5.1.4 Runtime dependencies of the FrancaCCG code generator

```
linux-vdso.so.1 (0x00007fff15bfb000)
libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007f3a3af3c000)
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f3a3ac3b000)
libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f3a3aa25000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f3a3a67c000)
/lib64/ld-linux-x86-64.so.2 (0x00007f3a3b247000)
```

Figure 5.3: Example output of `ldd` on the FIDLtoXML binary included in FrancaCCG.

Table 5.3: Size of runtime dependencies of the FIDLtoXML code generator part of FrancaCCG

Library name	Size in kB
libstdc++.so.6	988
libm.so.6	1028
libgcc_s.so.1	88
libc.so.6	1692
ld-linux-x86-64.so.2	140
TOTAL	**3936**

The size of the FrancaCCG suite, with source code, binaries, and helper shell scripts, was found to be 4844 kB.

FrancaCCG consists of two code generators. As seen in Table 5.3, the runtime dependencies of the FIDLtoXML part of the FrancaCCG software suite is very small, with only a few libraries needed, totaling 3936 kB. In addition, these libraries are standard libraries used by most Linux binaries written in C++.

FrancaCCG includes a second code generator, XMLtoC, written in Python. Due to Python being an interpreted language, no binary is available for this code generator. Instead, one can look at the installation size of a normal Python installation. Version 2.7 of Python for the AMD64 architecture was studied, and its installation size was found to be 8841 kB [32].

5.2 P2: Error feedback

The test suite of eight faulty Franca IDL files, described in Chapter 3, was run on both the CommonAPI command line tool and on FrancaCCG. For each test case, the resulting error feedback given was studied to determine if it was deemed good enough, meaning the nature of the error and the approximate location of the error was detailed by the error message.

5.2.1 Summary

Table 5.4: Results of running test suite

	CommonAPI	FrancaCCG
Cases with good error feedback	2	6
Cases with insufficient error feedback	5	0
Cases where error was not found, and faulty files were generated	1	0
Cases where error was not found, and correct files were generated	0	2

In Table 5.4, the results of running the test suite on both code generators are summarized. Running the test case suite with the CommonAPI tool produced two good error messages. In addition, one of the errors in the test suite, *"File structured wrong"*, was not found by the tool, silently producing faulty generated files without any error message. These files were missing large parts of the code corresponding to the correct Franca IDL file.

FrancaCCG, the code generator developed as part of this thesis work, produced six good error messages when running the test suite. In two cases, the error of the test case file was not found. However, in both of these cases, the code produced was identical to the code produced when using non-faulty input files.

Please see Appendix C for complete listing of the resulting error feedback for each test case.

5.2.2 Test case 1: Missing or additional curly bracket

```
Loading main class : org.genivi.commonapi.cmdline.StandaloneGen
Sat Aug 08 14:00:40 CEST 2015 - [main] Product-specified preferences called before plugin is started
0    ERROR StandaloneGen   - Exception occurred !
java.lang.NullPointerException
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getRelativeNameReference(FrancaGeneratorExtensions
              .java:265)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getNameReference(FrancaGeneratorExtensions.java
              :1484)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getTypeName(FrancaGeneratorExtensions.java:1410)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions$5.apply(FrancaGeneratorExtensions.java:564)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions$5.apply(FrancaGeneratorExtensions.java:1)
        at org.eclipse.xtext.xbase.lib.internal.FunctionDelegate.apply(FunctionDelegate.java:41)
        at com.google.common.collect.Lists$TransformingRandomAccessList.get(Lists.java:491)
        at java.util.AbstractList$Itr.next(AbstractList.java:358)
        at com.google.common.base.Joiner.appendTo(Joiner.java:128)
        at com.google.common.base.Joiner.appendTo(Joiner.java:186)
        at com.google.common.base.Joiner.join(Joiner.java:243)
        at com.google.common.base.Joiner.join(Joiner.java:232)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.join(IterableExtensions.java:450)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinitionSignature(
              FrancaGeneratorExtensions.java:572)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinitionWithin(FrancaGeneratorExtensions
              .java:515)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinition(FrancaGeneratorExtensions.java
              :493)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxyHeader(
              FInterfaceDBusProxyGenerator.java:240)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxy(FInterfaceDBusProxyGenerator.
              java:54)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:246)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:1)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.forEach(IterableExtensions.java:399)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator.doGenerateDBusComponents(FrancaDBusGenerator.java:308)
        at org.genivi.commonapi.dbus.Generator.generate(Generator.java:21)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:299)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
java.lang.NullPointerException
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getRelativeNameReference(FrancaGeneratorExtensions
              .java:265)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getNameReference(FrancaGeneratorExtensions.java
              :1484)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getTypeName(FrancaGeneratorExtensions.java:1410)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions$5.apply(FrancaGeneratorExtensions.java:564)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions$5.apply(FrancaGeneratorExtensions.java:1)
        at org.eclipse.xtext.xbase.lib.internal.FunctionDelegate.apply(FunctionDelegate.java:41)
        at com.google.common.collect.Lists$TransformingRandomAccessList.get(Lists.java:491)
        at java.util.AbstractList$Itr.next(AbstractList.java:358)
        at com.google.common.base.Joiner.appendTo(Joiner.java:128)
        at com.google.common.base.Joiner.appendTo(Joiner.java:186)
        at com.google.common.base.Joiner.join(Joiner.java:243)
        at com.google.common.base.Joiner.join(Joiner.java:232)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.join(IterableExtensions.java:450)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinitionSignature(
              FrancaGeneratorExtensions.java:572)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinitionWithin(FrancaGeneratorExtensions
              .java:515)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinition(FrancaGeneratorExtensions.java
              :493)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxyHeader(
              FInterfaceDBusProxyGenerator.java:240)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxy(FInterfaceDBusProxyGenerator.
              java:54)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:246)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:1)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.forEach(IterableExtensions.java:399)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator.doGenerateDBusComponents(FrancaDBusGenerator.java:308)
        at org.genivi.commonapi.dbus.Generator.generate(Generator.java:21)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:299)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
```

Figure 5.4: Error feedback generated by CommonAPI when processing test case 1.

The CommonAPI tool successfully finds the error, however the verbose error message, seen in Figure 5.4, does not contain any information about the nature of the error or its location. FrancaCCG correctly identifies the error as a syntax error and reports the approximate location of it in the code.

5.2.3 Test case 2: Import of non-existing file

Both code generators were able to correctly identify the error. While the error feedback from the CommonAPI code generator is verbose and hard to read, it states very early in the message that the error is caused by a missing file, and also states the path to the file. The error message from FrancaCCG is shorter and simply states that the file could not be imported.

5.2.4 Test case 3: Franca file structured wrong

```
Loading main class : org.genivi.commonapi.cmdline.StandaloneGen
Sat Aug 08 14:08:34 CEST 2015 - [main] Product-specified preferences called before plugin is started
0    INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerDBusProxy.h
3    INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerDBusProxy.cpp
4    INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerDBusStubAdapter.h
17   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerDBusStubAdapter.cpp
48   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/MediaTypes.h
50   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/Indexer.h
55   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/Indexer.h
56   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/Indexer.cpp
62   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerProxyBase.h
63   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerProxy.h
65   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerStub.h
67   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerStubDefault.h
67   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerStubDefault.cpp
68   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerServiceAbstract.h
68   INFO StandaloneGen    - FrancaStandaloneGen done.
```

Figure 5.5: Error feedback generated by CommonAPI when processing test case 3.

None of the code generators were able to find this error. However, while the CommonAPI command line tool reports no error (see Figure 5.5), the files generated by the CommonAPI tool were faulty. This will be further discussed in the *Discussion* chapter. The files generated by FrancaCCG were identical to the files generated from a Franca file with the same content but correct structure.

5.2.5 Test case 4: Misspelled language construct

```
error: parse error
Syntax error at line 10:
   nethod getDatabasePath {
Aborting code generation.
ERROR: FIDLtoXML code generator failed. Code generation aborted.
```

Figure 5.6: Error feedback generated by FrancaCCG when processing test case 4.

Both code generators found the error. The error message from the CommonAPI tool does not state the nature or location of the error in the code, while this was clearly stated in the error feedback from FrancaCCG, shown in Figure 5.6.

5.2.6 Test case 5: Usage of non-existing data type

Both code generators found the error. The error message from the CommonAPI tool does not state the nature or location of the error, while this was clearly stated in the error feedback from FrancaCCG.

5.2.7 Test case 6: Extension loop

```
ERROR: Custom Franca types cannot be resolved:
NAME: BackendError
TYPE: ENUMERATION
D-BUS SIGNATURE:
EXTENDS: MediaManagerError
ENUM MEMBER: BACKEND_UNREACHABLE =

NAME: MediaManagerError
TYPE: ENUMERATION
D-BUS SIGNATURE:
EXTENDS: BackendError
ENUM MEMBER: NO_ERROR =

NAME: IndexerError
TYPE: TYPEDEF
D-BUS SIGNATURE:
VALUE: BackendError

Aborting code generation.
ERROR: FIDLtoXML code generator failed. Code generation aborted.
```

Figure 5.7: Error feedback generated by FrancaCCG when processing test case 6.

This error was found by both code generators. They both correctly describe the nature of the error, and its approximate location in the code. The error feedback from FrancaCCG can be seen in Figure 5.7

5.2.8 Test case 7: More than one ”out” section

Both code generators found this error. The error is not identifiable by reading the feedback from the CommonAPI tool, while it is clearly stated in the feedback from FrancaCCG.

5.2.9 Test case 8: No version defined

This error is found by the CommonAPI tool, producing an error message in which the error is not correctly stated. FrancaCCG does not find this error, and outputs code equivalent to the code outputted from correct Franca files, with version defined in them. This is further discussed in the Discussion chapter.

5.3 Compatibility with CommonAPI generated code

The compatibility of the code produced by the two code generators was tested by running both the FrancaCCG code generator suite and the CommonAPI code generator, with the case study Franca IDL files described in Figure 4.1 as input. This produced two sets of server proxies and server stubs. A simple server implementation using the server stub generated by CommonAPI was then written.

By compiling and executing this server implementation, a D-Bus service is published on the system, and can subsequently be studied by using the `d-feet` tool. A client application trying to connect to this service is required by the D-Bus daemon to conform to the same D-Bus interface as the server application. Simple debug code to provide visual feedback on successful connection and successful method calls was added to the client implementation generated by FrancaCCG. This client implementation was then compiled and run, and was found to successfully connect to the D-Bus service published by the server implementation. Likewise, method calls initiated by the client were successfully received and handled by the server, and the corresponding method call responses by the server were successfully received by the client.

The code generated by FrancaCCG and CommonAPI for the case study Franca IDL interface is thus compatible on the D-Bus level. The compatibility of other Franca IDL interfaces was not tested, due to the limited scope of this thesis.

5.4 Assessment of FrancaCCG by Pelagicore

At the end of this thesis work, the FrancaCCG code generator suite was demonstrated at Pelagicore. Below, an assessment of FrancaCCG is presented, made by a Pelagicore employee very proficient in the use of Franca IDL and the CommonAPI code generator.

"The set-up process of the code generator is much simpler than the CommonAPI code generator which is based on Eclipse and Xtend/Xtext. One of the reasons why FrancaCCG is easier to set up is because the dependency chain is simpler. Where FrancaCCG depends on tools commonly installable

via Linux package managers, such as bnfc, flex and bison, the Common-API code generator depends on Eclipse plugins which often have version mismatches and can be difficult to install.

The most common use case for using Franca code generators is to automatically generate code within build systems (primarily Yocto). These build systems typically run on build servers, and always build without manual intervention. A major issue with the current Franca code generator for CommonAPI has been that there is no good command-line code generator, capable of building outside Eclipse. Pelagicore has created crude wrappers around the core libraries of the Eclipse code generator in order to run it outside Eclipse, but these wrappers lack user feedback, since the user feedback of the core libraries is intended to be displayed in the Eclipse UI, which has been stripped away by using the wrappers.

The user feedback is important to have in the automated build system, since the build system can be configured to alert developers of errors, based on this feedback. The developer introducing an erroneous FIDL-file can, for instance, have an email sent to him/her with the git commit introducing the error, and also the error output. Not all of these errors appear on developer machines, so this is a very important feature.

A major benefit of using FrancaCCG is thus that we can easily use it in our automated build systems due to the availability of user feedback. In terms of improvement, the feedback from FrancaCCG could be formatted in a standard form easily parsed by the automated build system (Jenkins in our case) - but we consider this a very minor change/improvement.

The code outputted by the code generator is C code, and relies only on the Glib and GIO standard libraries. By compiling the code generated from FrancaCCG into shared objects, we can easily interact with the generated code from other languages (such as Python). This has been an issue with the Franca CommonAPI code generator, mostly since it outputs C++ code, which is typically not as well supported for binding in other languages. Due to time constraints in the thesis project, the bindings for other languages have not been created, but based on previous experience and by the looks of the generated code, I would assume that this is rather trivial.

I believe that the straight-forward design of the FrancaCCG code generator lends it well for adding more features, which will be necessary in order to cover the same feature scope as the Franca CommonAPI code generator covers.

This project has been a success, since it has shown that the features of the Franca CommonAPI code generator can be implemented in a much simpler and cleaner way than they have been in previous projects."

- Jonatan Pålsson, Software Engineer at Pelagicore

Chapter 6

Discussion

In this chapter, the results presented in the previous chapter are discussed, and a comparison of the CommonAPI code generator and FrancaCCG is made for each of the two identified problems stated in Chapter 3.

6.1 Evaluation of the software dependencies of the tools

6.1.1 Runtime dependencies of the generated code

The implementations using code generated from FrancaCCG had much more lightweight runtime dependencies, with the dependencies being roughly half the size in kB compared to the runtime dependencies of an implementation using CommonAPI generated code.

There were four libraries used by both implementations. Removing these from consideration leaves 3736 kB of dependencies for FrancaCCG and 9132 kB for CommonAPI, further increasing the gap between the code generators.

Worth noting is that a large part of the total size of the dependencies of the CommonAPI generated code is due to the two CommonAPI libraries, with a total size of 6736 kB. Thus, on a target system already running other CommonAPI components, the additional runtime dependencies of the CommonAPI generated code are much smaller compared to a system not already running CommonAPI.

For FrancaCCG generated code, the largest runtime libraries not shared with the CommonAPI generated code are `libglib`, `libgio` and `libgobject`. These are all part of the `glib` package, a collection of low-level system libraries which is included in most Linux distributions. Thus, on most target systems of FrancaCCG, these runtime dependencies are already installed and used, and can be disregarded from the measurement of runtime dependencies.

Thus, it is my opinion that FrancaCCG solves the problem P1 in regard to the runtime dependencies of the generated code, especially in cases where the CommonAPI libraries are not already present on the target system.

6.1.2 Runtime dependencies of the code generators

The runtime dependencies of the actual code generators was harder to measure than the runtime dependencies of the generated code, due to the code generators not being available as binary files. Instead the installation size of the tools used to run them was measured.

The CommonAPI code generator was found to have an installation size as large as 550328 kB. This extremely large installation size is due to the Eclipse Toolkit being used and required by the code generator.

FrancaCCG was found to have a runtime dependency of 17621 kB. This includes the source code and binaries of both code generators, as well as libraries needed to run the binary of the first code generator, FIDLtoXML, and the Python library needed to run the second code generator, XMLtoC. However, this measurement does not take into account potential dependencies of the Python library that are not already installed. This set of libraries highly depend on the target system, since they are system libraries widely used, and it is not feasible to assess which are likely to be already installed and which are not. Though it is not feasible to measure the exact size of the set of additional libraries needed by the Python library, it is highly unlikely that this set is larger than the installation size of the CommonAPI code generator.

Thus, I believe that on most target systems the size of the runtime dependencies of the FrancaCCG code generator suite are much smaller than the size of the runtime dependencies of the CommonAPI code generator.

6.2 Evaluation of the error feedback of the tools

FrancaCCG, the code generator developed as part of this thesis work, gave error feedback of much better quality when processing files with common Franca IDL errors, compared to the CommonAPI command line tool. Out of the eight test cases, FrancaCCG gave error feedback good enough to easily identify the error in six cases, compared to CommonAPI which gave good error feedback in two cases only.

There were two test cases in which FrancaCCG did not find the error in the files. While this is still a shortcoming of the code generator, it is not critical, since these errors correspond to the limits of the currently available code generators, and to a lesser extent the design choices of Franca IDL. There were no differences in the code generated by FrancaCCG for these error cases and the code generated from the original, error-free case. Still, it would have been advantageous if the code generator gave a warning message when parsing these two error cases, since they will most likely not work on other code generators for Franca IDL. This functionality can easily be added to FrancaCCG in the future, without having to make drastic changes to the source code of the code generators.

While the eight test cases do not fully cover all error cases possible when defining interfaces in Franca IDL, they were designed in collaboration with a Franca IDL professional to be general enough to cover most such possible error cases. In addition, basing the first code generator on the BNFC suite gives good coverage of syntax errors of many kinds. This is due to the preciseness needed when designing the grammar file used as input for the BNFC suite; the generated parser and lexer will not accept Franca files with language constructs not found in the grammar file (see Appendix B). This is demonstrated in error cases such as *"missing curly bracket"* from the test suite. Here, the parser will not explicitly check if there are any missing brackets. Instead, it will check so that the file conforms to the Franca IDL grammar, which includes the number and possible locations of curly brackets, and many more possible error cases not in the test suite. Thus, it is my belief that most of the possible error cases are indeed found by FrancaCCG. This

BNF grammar file for the subset of Franca IDL could be published separately and used in other projects, for example as basis for other code generators using the BNFC suite.

The CommonAPI command line tool was not able to handle the test case containing a Franca file with the wrong internal structure. In this file, a type definition was placed early on in the file, instead of in the end as the Franca IDL specification requires. The CommonAPI tool did not report any error in this test case, and its output build feedback was indistinguishable from the output given when processing error-free files. However, the resulting code generated contained large differences from code generated from a correct file. All mentions of the type definition had been removed, greatly changing the interface of the file. This is a serious problem, since it is very hard for a programmer using the tool to notice such an error when no error feedback at all is given.

Thus, due to the greatly increased quality of the error feedback and to the lack of faulty generated files, the problem P2 is in my opinion successfully solved by FrancaCCG.

6.3 Evaluation of the compatibility with CommonAPI generated code

The server stubs and proxies generated by FrancaCCG were found to be compatible to the stubs and proxies generated by the CommonAPI code generator. This can be very useful. One could, for example, use FrancaCCG to generate a new client implementation of a Franca IDL interface which already has a working server implementation written using CommonAPI generated code, without making any changes to the server implementation.

6.4 Future work

At the present time, FrancaCCG does not support the entire Franca IDL language. Rather, only the Franca IDL features used in the case study files are supported. The current version of FrancaCCG is thus a prototype, which needs to be further developed to be of true practical use. However, adding support for all advanced features of the Franca IDL language, such as contracts, is not needed. These features are not currently supported by the existing CommonAPI code generator. At a minimum though, features such as signals and container data types should be added to FrancaCCG, if more advanced Franca IDL files are to be supported. Adding these features is however very possible to do, either by myself, Pelagicore employees, or other developers.

FrancaCCG solves the two identified problems of the old code generators in a good way, in my opinion. Therefore, it is my recommendation to keep developing FrancaCCG, by adding more Franca IDL features to the code generators. Even if it does not fully replace the command line CommonAPI code generator in all use cases, it can be a valuable tool for Franca IDL development, not only by Pelagicore but also by other companies using Franca IDL in their workflow.

Both the FrancaCCG code generator suite and the code generated by it would benefit from more thorough testing and debugging. Since the input of the code generators can vary in a multitude of ways, a system with automatically generated input would be very useful, since it is not feasible to cover all test cases by manual design. Thus it would be advantageous to make use of an automatic test system such as QuickCheck, which has been proven successful in testing other automotive industry software [4].

An potential future course of action is to integrate FrancaCCG into the Eclipse Toolkit. This allows developers currently using the Eclipse Toolkit to develop Franca IDL interfaces to easily try FrancaCCG out, while still retaining advantages of the Eclipse Toolkit such as syntax highlighting. This can be an efficient work flow for some use cases, such as when generating a server stub using CommonAPI and a server proxy using FrancaCCG for the same Franca IDL interface. FrancaCCG can be integrated to the Eclipse Toolkit either as an external tool [16], or as a plug-in using the Plug-in Development Environment [17].

Chapter 7

Conclusion

In this chapter, the conclusions learned during this thesis work is presented.

7.1 Resulting code generators

During this thesis work a software suite, FrancaCCG, consisting of prototypes of new code generators for Franca IDL, was developed. FrancaCCG was then evaluated against two known problems of the old code generators.

The first problem stated that the present code generators available had very large runtime dependencies, both in regard to the code generator itself and to the code generated by it. FrancaCCG was found to have a installation size in the order of one tenth of the installation size of the old CommonAPI code generator. Also, it was found to have a runtime dependency size of the generated code roughly half of the size of the dependencies corresponding to CommonAPI generated code. Thus, FrancaCCG was deemed successful in regard of solving the problem of the size of the runtime dependencies.

The second problem described the difficulty of using the old tools in automated build systems, mainly due to the lack of good error feedback. A test suite of eight faulty Franca IDL interface files was given as input to both code generators. The command line version of the CommonAPI code generator produced feedback deemed good enough to find the error in two cases, while FrancaCCG produced such error feedback in six cases. In addition, CommonAPI produced faulty output files but reported no error in one case, making the error very difficult to find. Based on this, FrancaCCG was deemed to give build feedback of better quality than the old code generators.

7.2 Personal experience

During this thesis work, a variety of different programming and scripting languages were used, including C, C++, Python, Franca IDL, D-Bus XML Introspection, Makefile, and BNF. Getting practical experience working with all these languages in a software project was very rewarding on a personal level. Working in collaboration with industry professionals at a small software development company such as Pelagicore was also a very positive and rewarding experience.

Were I to re-do this thesis work, I would have put a larger focus on finding working examples right away when learning how to implement new libraries, such as the `GDBus` library used in the resulting code of the second code generator developed. Much time was spent during this thesis work studying the documentation of such libraries, and I believe it would have been more efficient to rather find a working example of code right away, and subsequently study the documentation.

7.3 Availability and future of FrancaCCG

The code generators of FrancaCCG presently only support a subset of all features of Franca IDL. While FrancaCCG is useful even though not all features of Franca IDL are supported, some features such as support for container types would greatly increase its usefulness. A suitable goal for further development would be to add support in the code generators for all features in the standard D-Bus XML Introspection format. This would enable the second code generator, XMLtoC, to be used in other D-Bus projects not using Franca IDL. The main features of D-Bus XML Introspection currently missing for this to work are support for container types and variants, which can be added to the software suite.

After this thesis work, I plan to continue development of FrancaCCG, with the goal of adding support for all features in the D-Bus XML Introspection in both code generators. This would increase the size of the subset of Franca IDL supported, reaching a level of support close to the current CommonAPI code generator.

FrancaCCG is freely available on Pelagicore's public GitHub page[1], as open source software using the GPL2 licence [15]. Interested readers are encouraged to download FrancaCCG, try it out, and contribute to the project. Included in the git repository of FrancaCCG is a number of example `*.fidl` files, including the test suite of eight faulty Franca IDL files used in this thesis.

[1]https://github.com/Pelagicore/FrancaCCG

References

[1] GENIVI Alliance. *About the Alliance*. URL: http://www.genivi.org/ (Visited in June 2015).

[2] GENIVI Alliance. *CommonAPI C++ Tutorial*. URL: http://docs.projects.genivi.org/ipc.common-api-tools/2.1.6/pdf/Tutorial.pdf (Visited in July 2015).

[3] GENIVI Alliance. *IPC CommonAPI C++*. URL: http://projects.genivi.org/commonapi/home (Visited in June 2015).

[4] T. Arts et al. “Testing AUTOSAR software with QuickCheck”. In: *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*. 2015, pp. 1–4. DOI: 10.1109/ICSTW.2015.7107466.

[5] Klaus Birken. *Franca 0.10.0 available*. URL: http://lists.genivi.org/pipermail/genivi-projects/2015-July/000594.html (Visited in July 2015).

[6] Klaus Birken. *Franca Introduction 1.0*. URL: https://code.google.com/a/eclipselabs.org/p/franca/downloads/detail?name=Franca_Introduction_v1_0_121002.pdf (Visited in Aug. 2015).

[7] Klaus Birken. *Franca Quick Install Guide*. URL: https://code.google.com/a/eclipselabs.org/p/franca/wiki/FrancaQuickInstallGuide (Visited in June 2015).

[8] Klaus Birken, Tamas Szabo, and Steffen Weik. *Franca*. URL: http://eclipse.org/proposals/modeling.franca/ (Visited in June 2015).

[9] Andrew D. Birrell and Bruce Jay Nelson. “Implementing Remote Procedure Calls”. In: *SIGOPS Oper. Syst. Rev.* 17.5 (Oct. 1983), pp. 3–. ISSN: 0163-5980. DOI: 10.1145/773379.806609. URL: http://doi.acm.org.proxy.lib.chalmers.se/10.1145/773379.806609.

[10] Erik Botö and Jonatan Pålsson. Private interview conducted at Pelagicore. 2015.

[11] Koen Claessen and John Hughes. “QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs”. In: *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming.* ICFP ’00. New York, NY, USA: ACM, 2000, pp. 268–279. ISBN: 1-58113-202-6. DOI: `10.1145/351240.351266`. URL: `http://doi.acm.org/10.1145/351240.351266`.

[12] Eric Eide et al. “Flick: A Flexible, Optimizing IDL Compiler”. In: *SIGPLAN Not.* 32.5 (May 1997), pp. 44–56. ISSN: 0362-1340. DOI: `10.1145/258916.258921`. URL: `http://doi.acm.org.proxy.lib.chalmers.se/10.1145/258916.258921`.

[13] Norman Feske. “A Case Study on the Cost and Benefit of Dynamic RPC Marshalling for Low-level System Components”. In: *SIGOPS Oper. Syst. Rev.* 41.4 (July 2007), pp. 40–48. ISSN: 0163-5980. DOI: `10.1145/1278901.1278908`. URL: `http://doi.acm.org.proxy.lib.chalmers.se/10.1145/1278901.1278908`.

[14] Apache Software Foundation. *Apache Thrift - Inderface Description Language (IDL).* URL: `https://thrift.apache.org/docs/idl` (Visited in Aug. 2015).

[15] Free Software Foundation. *GNU General Public Licence v2.0.* URL: `http://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html` (Visited in Aug. 2015).

[16] The Eclipse Foundation. *Help - Eclipse Platform.* URL: `http://help.eclipse.org/mars/index.jsp?topic=%2Forg.eclipse.platform.doc.user%2FgettingStarted%2Fqs-97_standalone_ets.htm` (Visited in Aug. 2015).

[17] The Eclipse Foundation. *PDE.* URL: `http://www.eclipse.org/pde/` (Visited in Aug. 2015).

[18] freedesktop.org. *D-Bus: Main Page.* URL: `http://dbus.freedesktop.org/doc/api/html/index.html` (Visited in June 2015).

[19] freedesktop.org. *dbus.* URL: `http://www.freedesktop.org/wiki/Software/dbus/` (Visited in June 2015).

[20] freedesktop.org. *DBusBindings.* URL: `http://www.freedesktop.org/wiki/Software/DBusBindings/` (Visited in June 2015).

[21] David A. Holland. *stdin(3) - Linux manual page.* URL: `http://man7.org/linux/man-pages/man3/stdout.3.html` (Visited in Aug. 2015).

[22] Object Management Group Inc. *OMG IDL.* URL: `http://www.omg.org/gettingstarted/omg_idl.htm` (Visited in Aug. 2015).

[23] H.-A. Jacobsen and B.J. Kramer. “Modeling interface definition language extensions”. In: *Technology of Object-Oriented Languages and Systems, 2000. TOOLS-Pacific 2000. Proceedings. 37th International Conference on.* 2000, pp. 242–252. DOI: 10.1109/TOOLS.2000.891373.

[24] A. Kaplan, J. Ridgway, and J. C. Wileden. “Why IDLs Are Not Ideal”. In: *Proceedings of the 9th International Workshop on Software Specification and Design.* IWSSD ’98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 2–. ISBN: 0-8186-8439-9. URL: http://dl.acm.org.proxy.lib.chalmers.se/citation.cfm?id=857205.858288.

[25] Centre for Language Technology. *The BNF Converter.* URL: http://bnfc.digitalgrammars.com/ (Visited in Aug. 2015).

[26] Robert Love. “Get on the D-BUS”. In: *Linux J.* 2005.130 (Feb. 2005), pp. 3–. ISSN: 1075-3583. URL: http://dl.acm.org.proxy.lib.chalmers.se/citation.cfm?id=1048011.1048014.

[27] John R Nestor, William A Wulf, and David A Lamb. *IDL-Interface description language: Formal description.* Carnegie-Mellon University. Department of Computer Science, 1981.

[28] J. Palsberg and C.B. Jay. “The essence of the Visitor pattern”. In: *Computer Software and Applications Conference, 1998. COMPSAC ’98. Proceedings. The Twenty-Second Annual International.* 1998, pp. 9–15. DOI: 10.1109/CMPSAC.1998.716629.

[29] *Pelagicore/common-api-cmdline - Github.* URL: https://github.com/Pelagicore/common-api-cmdline (Visited in July 2015).

[30] Havoc Pennington et al. *D-Bus specification.* URL: http://dbus.freedesktop.org/doc/dbus-specification.html#introspection-format (Visited in July 2015).

[31] Lennart Poettering. *[systemd-devel] [HEADSUP] libsystemd-bus + kdbus plans.* URL: http://lists.freedesktop.org/archives/systemd-devel/2013-March/009797.html (Visited in July 2015).

[32] The Debian Project. *Debian – Details of package python2.7 in wheezy.* URL: https://packages.debian.org/wheezy/python2.7 (Visited in Aug. 2015).

[33] The GNOME Project. *D-Bus GLib bindings - Reference Manual.* URL: https://developer.gnome.org/dbus-glib/unstable/ (Visited in July 2015).

[34] The GNOME Project. *gdbus-codegen: GIO Reference Manual.* URL: https://developer.gnome.org/gio/stable/gdbus-codegen.html (Visited in July 2015).

[35] The GNOME Project. *GNOME.* URL: https://www.gnome.org/ (Visited in July 2015).

[36] The GNOME Project. *Migrating to GDBus: GIO Reference Manual.* URL: https://developer.gnome.org/gio/unstable//ch35.html (Visited in July 2015).

[37] The Jenkins Project. *Welcome to Jenkins CI!* URL: https://jenkins-ci.org/ (Visited in Aug. 2015).

[38] Yocto Project. *Yocto Project — Open Source embedded Linux build system, package metadata and SDK generator.* URL: https://www.yoctoproject.org/ (Visited in Aug. 2015).

[39] Jonatan Pålsson. Private interview conducted at Pelagicore. 2015.

[40] Scott James Remnant. *Ubuntu Manpage: nih-dbus-tool - D-Bus binding code generator.* URL: http://manpages.ubuntu.com/manpages/karmic/man1/nih-dbus-tool.1.html (Visited in July 2015).

[41] David A Rusling. *Interprocess Communication Mechanisms.* URL: http://www.tldp.org/LDP/tlk/tlk.html (Visited in Aug. 2015).

[42] Andreas Warnke. *ipc-quartztime: Main Page.* URL: http://andreaswarnke.de/ipc-quartztime/html/index.html (Visited in July 2015).

[43] David Zeuthen. *FreshPorts – devel/eggdbus.* URL: http://www.freshports.org/devel/eggdbus/ (Visited in July 2015).

Appendix A

Source code for the test suite

In this appendix, the source code for the eight faulty Franca IDL files used in the evaluation of P2 is presented. The source code for the original, error-free Franca interface which each error case is based on is given. For each of the eight test cases, the lines of code differing from the original file is shown, and the location of the error is marked in red in the code.

A.1 Original, error-free files

MediaTypes.fidl:

```
package org.genivi.mediamanager
typeCollection MediaTypes {
  enumeration BackendError extends MediaManagerError {
      BACKEND_UNREACHABLE }
  enumeration MediaManagerError { NO_ERROR }
}
```

MediaIndexer.fidl:

```
package org.genivi.mediamanager
import org.genivi.mediamanager.MediaTypes.* from "MediaTypes.fidl"
interface Indexer {
  version {
      major 1
      minor 0
  }
  <** @description: Example comment**>
  attribute IndexerStatus indexerStatus readonly noSubscriptions
  method getDatabasePath {
      out {
          String output
          IndexerError e
      }
  }
  method stopIndexing {
      out { IndexerError e }
  }

  method startIndexing {
      out { IndexerError e }
  }
  enumeration IndexerStatus {
      RUNNING
      STOPPED
      IDLE
  }
  typedef IndexerError is BackendError
}
```

A.2 Case 1: Missing curly bracket

MediaIndexer.fidl:

```
  }
  method stopIndexing
     out { IndexerError e }
```

A.3 Case 2: Import of non-existing file

MediaIndexer.fidl:

```
package org.genivi.mediamanager
import org.genivi.mediamanager.MediaTypes.* from "MediaType.fidl"
interface Indexer {
```

A.4 Case 3: File structured wrong

MediaIndexer.fidl:

```
interface Indexer {
  version {
     major 1
     minor 0
  }
  typedef IndexerError is BackendError
  <** @description: Example comment**>
  attribute IndexerStatus indexerStatus readonly noSubscriptions
```

A.5 Case 4: Misspelled language construct

MediaIndexer.fidl:

```
  attribute IndexerStatus indexerStatus readonly noSubscriptions
  nethod getDatabasePath {
     out {
```

A.6 Case 5: Usage of non-existing data type

MediaIndexer.fidl:

```
    method stopIndexing {
        out { IndexError e }
    }
```

A.7 Case 6: Circular dependency in extensions

MediaTypes.fidl:

```
typeCollection MediaTypes {
    enumeration BackendError extends MediaManagerError{
        BACKEND_UNREACHABLE }
    enumeration MediaManagerError extends BackendError{ NO_ERROR }
}
```

A.8 Case 7: Several "out" sections in method definition

MediaIndexer.fidl:

```
    method getDatabasePath {
        out {
            String output
        }
        out {
            IndexerError e
        }
    }
```

A.9 Case 8: No version defined

MediaIndexer.fidl:

```
import org.genivi.mediamanager.MediaTypes.* from "MediaTypes.fidl"
interface Indexer {

  <** @description: Example comment**>
```

Appendix B

BNF grammar for a subset of Franca IDL

```
-- Franca IDL grammar file
-- BNF grammar for subset of Franca IDL
-- Author     Jesper Lundqvist
-- Version    0.3

-- Usage: Use with BNFC software suite.


-- definitions
Prog.                Program ::= [Def] ;

-- A Franca IDL file consist of a series of definitions.
DPackage.            Def ::= "package" PackageName ;
DPackageName.        PackageName ::= [NamespaceID] ;
DInterface.          Def ::= "interface" Id "{" IBody "}" ;
terminator Def "" ;


-- I define a type collection in the same way as an interface.
-- This is to avoid having separate definitions for the same
   structure,
-- e.g. Enumerations as IBodyItem and TypeCollectionItem.
DTypeCollection.     Def ::= "typeCollection" Id "{" IBody "}" ;
```

```
-- Import from another Franca file.
DImport.            Def ::= "import" Namespace "from" "\""
   FileName "\"" ;
-- DImport.         Def ::= "import" Namespace ".*" "from" "\""
   FileName "\"" ;
DFileName.          FileName ::= Id "." FileEnding ;
DFileNameNoEnd.     FileName ::= Id ;
DFileEnding.        FileEnding ::= Id ;
DNamespace.         Namespace ::= [NamespaceID] ;
DNamespaceID.       NamespaceID ::= Id ;
DNamespaceIDAll.       NamespaceID ::= "*" ;
separator NamespaceID "." ;


-- Interfaces consist of a number of interface body items.
DIBody.             IBody ::= [IBodyItem] ;
terminator IBodyItem "" ;
--separator IBodyItem "" ;


-- Methods are one kind of interface body item. They can have
   either
-- in- or out-parameters, both, or none.
DMethod.            IBodyItem ::= "method" Id "{" "}" ;
DInMethod.          IBodyItem ::= "method" Id "{" "in" "{" [InVari
   ] "}" "}" ;
DOutMethod.         IBodyItem ::= "method" Id "{" "out" "{" [
   OutVari] "}" "}" ;
DIOMethod.          IBodyItem ::= "method" Id "{" "in" "{" [InVari
   ] "}" "out" "{" [OutVari] "}" "}" ;


-- Version is another interface body item.
DVersion.           IBodyItem ::= "version" "{" "major" Integer "
   minor" Integer "}" ;


-- Attributes are also an interface body item.
-- Due to only two attribute flags being possible, the combinations
    are
-- listed here separately to make code generation easier.
```

```
DAttrib.              IBodyItem ::= "attribute" Type Id ;
DAttribReadOnly.      IBodyItem ::= "attribute" Type Id "readonly" ;
DAttribNoSub.         IBodyItem ::= "attribute" Type Id "
   noSubscriptions" ;
DAttribRONS.  IBodyItem ::= "attribute" Type Id "readonly" "
   noSubscriptions" ;


-- Parameters consist of a type and an id.
DInVar.               InVari ::= Type Id ;
DOutVar.              OutVari ::= Type Id ;
--DInVarCustomType.   InVari ::= CustomType Id ;


DVar.                 Vari ::= Type Id ;
--DVarArr.            Vari ::= Type "[]" Id ;
terminator Vari "" ;
terminator InVari "" ;
terminator OutVari "" ;


-- Enumerations (interface body item)
DEnumDef.             IBodyItem ::= "enumeration" Id "{" EnumList
   "}" ;
DExEnumDef.   IBodyItem ::= "enumeration" EnumId "extends" Id "{"
   EnumList "}" ;

DEnumIdent.           EnumId ::= Id ;
DEnumList.            EnumList ::= [Enum] ;
terminator Enum "" ;
DEnum.                Enum ::= Id ;
DEnumValue.           Enum ::= Id "=" Integer ;


-- Typedefs
DTypeDef.             IBodyItem ::= "typedef" TypeDefId "is" Type ;
DTypeDefCustom.       IBodyItem ::= "typedef" TypeDefId "is" Id ;
DTypeDefIdent.        TypeDefId ::= Id ;


-- Franca IDL types

DUIntEight.           Type ::= "UInt8" ;
```

```
DIntEight.          Type ::= "Int8" ;
DUIntSixteen.       Type ::= "UInt16" ;
DIntSixteen.        Type ::= "Int16" ;
DUIntThirtyTwo.     Type ::= "UInt32" ;
DIntThirtyTwo.      Type ::= "Int32" ;
DUIntSixtyFour.     Type ::= "UInt64" ;
DIntSixtyFour.      Type ::= "Int64" ;
DBoolean.           Type ::= "Boolean" ;
DFloat.             Type ::= "Float" ;
DDouble.            Type ::= "Double" ;
DString.            Type ::= "String" ;
DByteBuffer.        Type ::= "ByteBuffer" ;
DCustomType.        Type ::= Id ;


-- Identifiers
token Id (letter (letter | digit | '_')*) ;


-- Regular comments
comment "//" ;
comment "/*" "*/" ;

-- Structured comments. These can be further implemented;
-- for now they are defined as regular comments.
comment "<**" "**>" ;
```

Appendix C

Error feedback from test suite

In this appendix, the resulting error feedback from running each test case Franca IDL file with both the CommonAPI command line tool, and the FrancaCCG tool, is presented.

C.1 Test case 1: Missing curly bracket

3.1.1 CommonAPI error feedback

```
Loading main class : org.genivi.commonapi.cmdline.StandaloneGen
Sat Aug 08 14:00:40 CEST 2015 - [main] Product-specified preferences called before plugin is started
0    ERROR StandaloneGen    - Exception occurred !
java.lang.NullPointerException
	at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getRelativeNameReference(FrancaGeneratorExtensions
	     .java:265)
	at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getNameReference(FrancaGeneratorExtensions.java
	     :1484)
	at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getTypeName(FrancaGeneratorExtensions.java:1410)
	at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions$5.apply(FrancaGeneratorExtensions.java:564)
	at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions$5.apply(FrancaGeneratorExtensions.java:1)
	at org.eclipse.xtext.xbase.lib.internal.FunctionDelegate.apply(FunctionDelegate.java:41)
	at com.google.common.collect.Lists$TransformingRandomAccessList.get(Lists.java:491)
	at java.util.AbstractList$Itr.next(AbstractList.java:358)
	at com.google.common.base.Joiner.appendTo(Joiner.java:128)
	at com.google.common.base.Joiner.appendTo(Joiner.java:186)
	at com.google.common.base.Joiner.join(Joiner.java:243)
	at com.google.common.base.Joiner.join(Joiner.java:232)
	at org.eclipse.xtext.xbase.lib.IterableExtensions.join(IterableExtensions.java:450)
	at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinitionSignature(
	     FrancaGeneratorExtensions.java:572)
	at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinitionWithin(FrancaGeneratorExtensions
	     .java:515)
	at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinition(FrancaGeneratorExtensions.java
	     :493)
	at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxyHeader(
	     FInterfaceDBusProxyGenerator.java:240)
	at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxy(FInterfaceDBusProxyGenerator.
	     java:54)
	at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:246)
	at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:1)
	at org.eclipse.xtext.xbase.lib.IterableExtensions.forEach(IterableExtensions.java:399)
```

```
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator.doGenerateDBusComponents(FrancaDBusGenerator.java:308)
        at org.genivi.commonapi.dbus.Generator.generate(Generator.java:21)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:299)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
java.lang.NullPointerException
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getRelativeNameReference(FrancaGeneratorExtensions
            .java:265)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getNameReference(FrancaGeneratorExtensions.java
            :1484)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getTypeName(FrancaGeneratorExtensions.java:1410)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions$5.apply(FrancaGeneratorExtensions.java:564)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions$5.apply(FrancaGeneratorExtensions.java:1)
        at org.eclipse.xtext.xbase.lib.internal.FunctionDelegate.apply(FunctionDelegate.java:41)
        at com.google.common.collect.Lists$TransformingRandomAccessList.get(Lists.java:491)
        at java.util.AbstractList$Itr.next(AbstractList.java:358)
        at com.google.common.base.Joiner.appendTo(Joiner.java:128)
        at com.google.common.base.Joiner.appendTo(Joiner.java:186)
        at com.google.common.base.Joiner.join(Joiner.java:243)
        at com.google.common.base.Joiner.join(Joiner.java:232)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.join(IterableExtensions.java:450)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinitionSignature(
            FrancaGeneratorExtensions.java:572)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinitionWithin(FrancaGeneratorExtensions
            .java:515)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinition(FrancaGeneratorExtensions.java
            :493)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxyHeader(
            FInterfaceDBusProxyGenerator.java:240)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxy(FInterfaceDBusProxyGenerator.
            java:54)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:246)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:1)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.forEach(IterableExtensions.java:399)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator.doGenerateDBusComponents(FrancaDBusGenerator.java:308)
        at org.genivi.commonapi.dbus.Generator.generate(Generator.java:21)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:299)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
```

3.1.2 FrancaCCG error feedback

```
error: parse error
Syntax error at line 17:
       out { IndexerError e }
Aborting code generation.
ERROR: FIDLtoXML code generator failed. Code generation aborted.
```

C.2 Test case 2: Import of non-existing file

3.2.1 CommonAPI error feedback

```
Loading main class : org.genivi.commonapi.cmdline.StandaloneGen
1    ERROR StandaloneGen    - Exception occurred !
org.eclipse.emf.ecore.resource.impl.ResourceSetImpl$1DiagnosticWrappedException: java.io.FileNotFoundException: /home/
       jesper/Documents/francaccodegen/testsuite/2ImportOfNonexistingFile/MediaType.fidl (No such file or directory)
        at org.eclipse.emf.ecore.resource.impl.ResourceSetImpl.handleDemandLoadException(ResourceSetImpl.java:319)
        at org.eclipse.emf.ecore.resource.impl.ResourceSetImpl.demandLoadHelper(ResourceSetImpl.java:278)
        at org.eclipse.xtext.resource.XtextResourceSet.getResource(XtextResourceSet.java:201)
        at org.eclipse.xtext.resource.SynchronizedXtextResourceSet.getResource(SynchronizedXtextResourceSet.java:26)
        at org.franca.core.utils.ModelPersistenceHandler.loadModel(ModelPersistenceHandler.java:77)
        at org.franca.core.utils.ModelPersistenceHandler.loadModel(ModelPersistenceHandler.java:94)
        at org.franca.core.utils.ModelPersistenceHandler.loadModel(ModelPersistenceHandler.java:94)
        at org.franca.deploymodel.dsl.FDeployPersistenceManager.loadModel(FDeployPersistenceManager.java:74)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:162)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
Caused by: java.io.FileNotFoundException: /home/jesper/Documents/francaccodegen/testsuite/2ImportOfNonexistingFile/
       MediaType.fidl (No such file or directory)
        at java.io.FileInputStream.open(Native Method)
        at java.io.FileInputStream.<init>(FileInputStream.java:146)
        at org.eclipse.emf.ecore.resource.impl.FileURIHandlerImpl.createInputStream(FileURIHandlerImpl.java:99)
        at org.eclipse.emf.ecore.resource.impl.ExtensibleURIConverterImpl.createInputStream(ExtensibleURIConverterImpl.java
               :360)
        at org.eclipse.xtext.resource.XtextResourceSet$1.createInputStream(XtextResourceSet.java:230)
        at org.eclipse.emf.ecore.resource.impl.ResourceImpl.load(ResourceImpl.java:1269)
        at org.eclipse.emf.ecore.resource.impl.ResourceSetImpl.demandLoad(ResourceSetImpl.java:259)
        at org.eclipse.emf.ecore.resource.impl.ResourceSetImpl.demandLoadHelper(ResourceSetImpl.java:274)
        ... 9 more
org.eclipse.emf.ecore.resource.impl.ResourceSetImpl$1DiagnosticWrappedException: java.io.FileNotFoundException: /home/
       jesper/Documents/francaccodegen/testsuite/2ImportOfNonexistingFile/MediaType.fidl (No such file or directory)
        at org.eclipse.emf.ecore.resource.impl.ResourceSetImpl.handleDemandLoadException(ResourceSetImpl.java:319)
        at org.eclipse.emf.ecore.resource.impl.ResourceSetImpl.demandLoadHelper(ResourceSetImpl.java:278)
        at org.eclipse.xtext.resource.XtextResourceSet.getResource(XtextResourceSet.java:201)
        at org.eclipse.xtext.resource.SynchronizedXtextResourceSet.getResource(SynchronizedXtextResourceSet.java:26)
        at org.franca.core.utils.ModelPersistenceHandler.loadModel(ModelPersistenceHandler.java:77)
        at org.franca.core.utils.ModelPersistenceHandler.loadModel(ModelPersistenceHandler.java:94)
        at org.franca.core.utils.ModelPersistenceHandler.loadModel(ModelPersistenceHandler.java:94)
        at org.franca.deploymodel.dsl.FDeployPersistenceManager.loadModel(FDeployPersistenceManager.java:74)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:162)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
Caused by: java.io.FileNotFoundException: /home/jesper/Documents/francaccodegen/testsuite/2ImportOfNonexistingFile/
       MediaType.fidl (No such file or directory)
        at java.io.FileInputStream.open(Native Method)
        at java.io.FileInputStream.<init>(FileInputStream.java:146)
        at org.eclipse.emf.ecore.resource.impl.FileURIHandlerImpl.createInputStream(FileURIHandlerImpl.java:99)
        at org.eclipse.emf.ecore.resource.impl.ExtensibleURIConverterImpl.createInputStream(ExtensibleURIConverterImpl.java
               :360)
        at org.eclipse.xtext.resource.XtextResourceSet$1.createInputStream(XtextResourceSet.java:230)
        at org.eclipse.emf.ecore.resource.impl.ResourceImpl.load(ResourceImpl.java:1269)
        at org.eclipse.emf.ecore.resource.impl.ResourceSetImpl.demandLoad(ResourceSetImpl.java:259)
        at org.eclipse.emf.ecore.resource.impl.ResourceSetImpl.demandLoadHelper(ResourceSetImpl.java:274)
        ... 9 more
```

3.2.2 FrancaCCG error feedback

```
Error opening fidl file from import statement: MediaType.fidl
ERROR: FIDLtoXML code generator failed. Code generation aborted.
```

C.3 Test case 3: File structured wrong

3.3.1 CommonAPI error feedback

```
Loading main class : org.genivi.commonapi.cmdline.StandaloneGen
Sat Aug 08 14:08:34 CEST 2015 - [main] Product-specified preferences called before plugin is started
0    INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerDBusProxy.h
3    INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerDBusProxy.cpp
4    INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerDBusStubAdapter.h
17   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerDBusStubAdapter.cpp
48   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/MediaTypes.h
50   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/Indexer.h
55   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/Indexer.h
56   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/Indexer.cpp
62   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerProxyBase.h
63   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerProxy.h
65   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerStub.h
67   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerStubDefault.h
67   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerStubDefault.cpp
68   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/3FileStructuredWrong/
       output_commonapi/org/genivi/mediamanager/IndexerServiceAbstract.h
68   INFO StandaloneGen    - FrancaStandaloneGen done.
```

3.3.2 FrancaCCG error feedback

```
FIDLtoXML successfully finished generating D-Bus XML Introspection for Franca IDL file ../testsuite/3FileStructuredWrong/
       MediaIndexer.fidl
XMLtoC successfully finished generating C server proxy and stub for D-Bus XML file testsuite/3FileStructuredWrong/
       MediaIndexer.xml
Code generation for Franca IDL file testsuite/3FileStructuredWrong/MediaIndexer.fidl was successful.
```

C.4 Test case 4: Misspelled language construct

3.4.1 CommonAPI error feedback

```
Loading main class : org.genivi.commonapi.cmdline.StandaloneGen
Sat Aug 08 14:12:44 CEST 2015 - [main] Product-specified preferences called before plugin is started
1    INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/4MisspelledLanguageConstruct/
       output_commonapi/org/genivi/mediamanager/IndexerDBusProxy.h
18  ERROR StandaloneGen   - Exception occurred !
java.lang.NullPointerException
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.dbusSignature(FrancaDBusGeneratorExtensions.
              java:150)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions._dbusFTypeSignature(
              FrancaDBusGeneratorExtensions.java:167)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.dbusFTypeSignature(
              FrancaDBusGeneratorExtensions.java:360)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.dbusSignature(FrancaDBusGeneratorExtensions.
              java:154)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.getTypeDbusSignature(
              FrancaDBusGeneratorExtensions.java:145)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.dbusSignature(FrancaDBusGeneratorExtensions.
              java:119)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusVariableInit(
              FInterfaceDBusProxyGenerator.java:972)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxySource(
              FInterfaceDBusProxyGenerator.java:481)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxy(FInterfaceDBusProxyGenerator.
              java:57)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:246)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:1)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.forEach(IterableExtensions.java:399)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator.doGenerateDBusComponents(FrancaDBusGenerator.java:308)
        at org.genivi.commonapi.dbus.Generator.generate(Generator.java:21)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:299)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
java.lang.NullPointerException
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.dbusSignature(FrancaDBusGeneratorExtensions.
              java:150)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions._dbusFTypeSignature(
              FrancaDBusGeneratorExtensions.java:167)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.dbusFTypeSignature(
              FrancaDBusGeneratorExtensions.java:360)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.dbusSignature(FrancaDBusGeneratorExtensions.
              java:154)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.getTypeDbusSignature(
              FrancaDBusGeneratorExtensions.java:145)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.dbusSignature(FrancaDBusGeneratorExtensions.
              java:119)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusVariableInit(
              FInterfaceDBusProxyGenerator.java:972)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxySource(
              FInterfaceDBusProxyGenerator.java:481)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxy(FInterfaceDBusProxyGenerator.
              java:57)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:246)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:1)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.forEach(IterableExtensions.java:399)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator.doGenerateDBusComponents(FrancaDBusGenerator.java:308)
        at org.genivi.commonapi.dbus.Generator.generate(Generator.java:21)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:299)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
```

3.4.2 FrancaCCG error feedback

```
error: parse error
Syntax error at line 10:
   nethod getDatabasePath {
Aborting code generation.
ERROR: FIDLtoXML code generator failed. Code generation aborted.
```

C.5 Test case 5: Usage of non-existing data type

3.5.1 CommonAPI error feedback

```
Loading main class : org.genivi.commonapi.cmdline.StandaloneGen
Sat Aug 08 14:14:41 CEST 2015 - [main] Product-specified preferences called before plugin is started
0    ERROR StandaloneGen    - Exception occurred !
java.lang.NullPointerException
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getRelativeNameReference(FrancaGeneratorExtensions
             .java:265)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getNameReference(FrancaGeneratorExtensions.java
             :1484)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getTypeName(FrancaGeneratorExtensions.java:1410)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions$5.apply(FrancaGeneratorExtensions.java:564)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions$5.apply(FrancaGeneratorExtensions.java:1)
        at org.eclipse.xtext.xbase.lib.internal.FunctionDelegate.apply(FunctionDelegate.java:41)
        at com.google.common.collect.Lists$TransformingRandomAccessList.get(Lists.java:491)
        at java.util.AbstractList$Itr.next(AbstractList.java:358)
        at com.google.common.base.Joiner.appendTo(Joiner.java:125)
        at com.google.common.base.Joiner.appendTo(Joiner.java:186)
        at com.google.common.base.Joiner.join(Joiner.java:243)
        at com.google.common.base.Joiner.join(Joiner.java:232)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.join(IterableExtensions.java:450)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinitionSignature(
             FrancaGeneratorExtensions.java:572)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinitionWithin(FrancaGeneratorExtensions
             .java:515)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinition(FrancaGeneratorExtensions.java
             :493)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxyHeader(
             FInterfaceDBusProxyGenerator.java:240)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxy(FInterfaceDBusProxyGenerator.
             java:54)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:246)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:1)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.forEach(IterableExtensions.java:399)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator.doGenerateDBusComponents(FrancaDBusGenerator.java:308)
        at org.genivi.commonapi.dbus.Generator.generate(Generator.java:21)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:299)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
java.lang.NullPointerException
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getRelativeNameReference(FrancaGeneratorExtensions
             .java:265)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getNameReference(FrancaGeneratorExtensions.java
             :1484)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.getTypeName(FrancaGeneratorExtensions.java:1410)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions$5.apply(FrancaGeneratorExtensions.java:564)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions$5.apply(FrancaGeneratorExtensions.java:1)
        at org.eclipse.xtext.xbase.lib.internal.FunctionDelegate.apply(FunctionDelegate.java:41)
        at com.google.common.collect.Lists$TransformingRandomAccessList.get(Lists.java:491)
        at java.util.AbstractList$Itr.next(AbstractList.java:358)
        at com.google.common.base.Joiner.appendTo(Joiner.java:125)
        at com.google.common.base.Joiner.appendTo(Joiner.java:186)
        at com.google.common.base.Joiner.join(Joiner.java:243)
        at com.google.common.base.Joiner.join(Joiner.java:232)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.join(IterableExtensions.java:450)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinitionSignature(
             FrancaGeneratorExtensions.java:572)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinitionWithin(FrancaGeneratorExtensions
             .java:515)
        at org.genivi.commonapi.core.generator.FrancaGeneratorExtensions.generateDefinition(FrancaGeneratorExtensions.java
             :493)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxyHeader(
             FInterfaceDBusProxyGenerator.java:240)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxy(FInterfaceDBusProxyGenerator.
             java:54)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:246)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:1)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.forEach(IterableExtensions.java:399)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator.doGenerateDBusComponents(FrancaDBusGenerator.java:308)
        at org.genivi.commonapi.dbus.Generator.generate(Generator.java:21)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:299)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
```

3.5.2 FrancaCCG error feedback

```
ERROR: Custom Franca type 'IndexError' has not been defined.
Aborting code generation.
ERROR: FIDLtoXML code generator failed. Code generation aborted.
```

C.6 Test case 6: Circular dependency in extensions

3.6.1 CommonAPI error feedback

```
Loading main class : org.genivi.commonapi.cmdline.StandaloneGen
Sat Aug 08 14:17:04 CEST 2015 - [main] Product-specified preferences called before plugin is started
1    INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/6ExtensionLoop/output_commonapi/
       org/genivi/mediamanager/IndexerDBusProxy.h
12   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/6ExtensionLoop/output_commonapi/
       org/genivi/mediamanager/IndexerDBusProxy.cpp
15   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/6ExtensionLoop/output_commonapi/
       org/genivi/mediamanager/IndexerDBusStubAdapter.h
21   INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/6ExtensionLoop/output_commonapi/
       org/genivi/mediamanager/IndexerDBusStubAdapter.cpp
36   ERROR StandaloneGen   - Exception occurred !
java.lang.IllegalArgumentException: FTypeCollection or FInterface has circular dependencies: org.franca.core.franca.impl.
       FTypeCollectionImpl@1dfd9611 (name: MediaTypes)
        at com.google.common.base.Preconditions.checkArgument(Preconditions.java:92)
        at org.genivi.commonapi.core.generator.FTypeGenerator.sortTypes(FTypeGenerator.java:453)
        at org.genivi.commonapi.core.generator.FTypeGenerator.generateFTypeDeclarations(FTypeGenerator.java:411)
        at org.genivi.commonapi.core.generator.FTypeCollectionGenerator.generateHeader(FTypeCollectionGenerator.java:132)
        at org.genivi.commonapi.core.generator.FTypeCollectionGenerator.generate(FTypeCollectionGenerator.java:53)
        at org.genivi.commonapi.core.generator.FrancaGenerator$5.apply(FrancaGenerator.java:261)
        at org.genivi.commonapi.core.generator.FrancaGenerator$5.apply(FrancaGenerator.java:1)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.forEach(IterableExtensions.java:399)
        at org.genivi.commonapi.core.generator.FrancaGenerator.doGenerateComponents(FrancaGenerator.java:264)
        at org.genivi.commonapi.core.Generator.generate(Generator.java:21)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:299)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
java.lang.IllegalArgumentException: FTypeCollection or FInterface has circular dependencies: org.franca.core.franca.impl.
       FTypeCollectionImpl@1dfd9611 (name: MediaTypes)
        at com.google.common.base.Preconditions.checkArgument(Preconditions.java:92)
        at org.genivi.commonapi.core.generator.FTypeGenerator.sortTypes(FTypeGenerator.java:453)
        at org.genivi.commonapi.core.generator.FTypeGenerator.generateFTypeDeclarations(FTypeGenerator.java:411)
        at org.genivi.commonapi.core.generator.FTypeCollectionGenerator.generateHeader(FTypeCollectionGenerator.java:132)
        at org.genivi.commonapi.core.generator.FTypeCollectionGenerator.generate(FTypeCollectionGenerator.java:53)
        at org.genivi.commonapi.core.generator.FrancaGenerator$5.apply(FrancaGenerator.java:261)
        at org.genivi.commonapi.core.generator.FrancaGenerator$5.apply(FrancaGenerator.java:1)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.forEach(IterableExtensions.java:399)
        at org.genivi.commonapi.core.generator.FrancaGenerator.doGenerateComponents(FrancaGenerator.java:264)
        at org.genivi.commonapi.core.Generator.generate(Generator.java:21)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:299)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
```

3.6.2 FrancaCCG error feedback

```
ERROR: Custom Franca types cannot be resolved:
NAME: BackendError
TYPE: ENUMERATION
D-BUS SIGNATURE:
EXTENDS: MediaManagerError
ENUM MEMBER: BACKEND_UNREACHABLE =

NAME: MediaManagerError
TYPE: ENUMERATION
D-BUS SIGNATURE:
EXTENDS: BackendError
ENUM MEMBER: NO_ERROR =

NAME: IndexerError
TYPE: TYPEDEF
D-BUS SIGNATURE:
VALUE: BackendError

Aborting code generation.
ERROR: FIDLtoXML code generator failed. Code generation aborted.
```

C.7 Test case 7: Several “out” sections in method definition

3.7.1 CommonAPI error feedback

```
Loading main class : org.genivi.commonapi.cmdline.StandaloneGen
Sat Aug 08 14:20:07 CEST 2015 - [main] Product-specified preferences called before plugin is started
0    INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/7SeveralOutSections/
       output_commonapi/org/genivi/mediamanager/IndexerDBusProxy.h
17   ERROR StandaloneGen    - Exception occurred !
java.lang.NullPointerException
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.dbusSignature(FrancaDBusGeneratorExtensions.
               java:150)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions._dbusFTypeSignature(
               FrancaDBusGeneratorExtensions.java:167)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.dbusFTypeSignature(
               FrancaDBusGeneratorExtensions.java:360)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.dbusSignature(FrancaDBusGeneratorExtensions.
               java:154)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.getTypeDbusSignature(
               FrancaDBusGeneratorExtensions.java:145)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.dbusSignature(FrancaDBusGeneratorExtensions.
               java:119)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusVariableInit(
               FInterfaceDBusProxyGenerator.java:972)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxySource(
               FInterfaceDBusProxyGenerator.java:481)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxy(FInterfaceDBusProxyGenerator.
               java:57)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:246)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:1)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.forEach(IterableExtensions.java:399)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator.doGenerateDBusComponents(FrancaDBusGenerator.java:308)
        at org.genivi.commonapi.dbus.Generator.generate(Generator.java:21)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:299)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
java.lang.NullPointerException
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.dbusSignature(FrancaDBusGeneratorExtensions.
               java:150)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions._dbusFTypeSignature(
               FrancaDBusGeneratorExtensions.java:167)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.dbusFTypeSignature(
               FrancaDBusGeneratorExtensions.java:360)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.dbusSignature(FrancaDBusGeneratorExtensions.
               java:154)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.getTypeDbusSignature(
               FrancaDBusGeneratorExtensions.java:145)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGeneratorExtensions.dbusSignature(FrancaDBusGeneratorExtensions.
               java:119)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusVariableInit(
               FInterfaceDBusProxyGenerator.java:972)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxySource(
               FInterfaceDBusProxyGenerator.java:481)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxy(FInterfaceDBusProxyGenerator.
               java:57)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:246)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:1)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.forEach(IterableExtensions.java:399)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator.doGenerateDBusComponents(FrancaDBusGenerator.java:308)
        at org.genivi.commonapi.dbus.Generator.generate(Generator.java:21)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:299)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
```

3.7.2 FrancaCCG error feedback

```
error: parse error
Syntax error at line 14:
       out {
Aborting code generation.
ERROR: FIDLtoXML code generator failed. Code generation aborted.
```

C.8 Test case 8: No version defined

3.8.1 CommonAPI error feedback

```
Loading main class : org.genivi.commonapi.cmdline.StandaloneGen
Sat Aug 08 14:21:31 CEST 2015 - [main] Product-specified preferences called before plugin is started
0    INFO StandaloneGen$1  - Writing file /home/jesper/Documents/francaccodegen/testsuite/8NoVersionDefined/
       output_commonapi/org/genivi/mediamanager/IndexerDBusProxy.h
10   ERROR StandaloneGen    - Exception occurred !
java.lang.NullPointerException
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxySource(
               FInterfaceDBusProxyGenerator.java:745)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxy(FInterfaceDBusProxyGenerator.
               java:57)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:246)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:1)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.forEach(IterableExtensions.java:399)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator.doGenerateDBusComponents(FrancaDBusGenerator.java:308)
        at org.genivi.commonapi.dbus.Generator.generate(Generator.java:21)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:299)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
java.lang.NullPointerException
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxySource(
               FInterfaceDBusProxyGenerator.java:745)
        at org.genivi.commonapi.dbus.generator.FInterfaceDBusProxyGenerator.generateDBusProxy(FInterfaceDBusProxyGenerator.
               java:57)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:246)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator$1.apply(FrancaDBusGenerator.java:1)
        at org.eclipse.xtext.xbase.lib.IterableExtensions.forEach(IterableExtensions.java:399)
        at org.genivi.commonapi.dbus.generator.FrancaDBusGenerator.doGenerateDBusComponents(FrancaDBusGenerator.java:308)
        at org.genivi.commonapi.dbus.Generator.generate(Generator.java:21)
        at org.genivi.commonapi.cmdline.StandaloneGen.run(StandaloneGen.java:299)
        at org.genivi.commonapi.cmdline.StandaloneGen.go(StandaloneGen.java:82)
        at org.genivi.commonapi.cmdline.main.Main.main(Main.java:80)
```

3.8.2 FrancaCCG error feedback

```
FIDLtoXML successfully finished generating D-Bus XML Introspection for Franca IDL file ../testsuite/8NoVersionDefined/
       MediaIndexer.fidl
XMLtoC successfully finished generating C server proxy and stub for D-Bus XML file testsuite/8NoVersionDefined/
       MediaIndexer.xml
Code generation for Franca IDL file testsuite/8NoVersionDefined/MediaIndexer.fidl was successful.
```

Appendix D

Code generated by FrancaCCG from case study Franca IDL interface

In this appendix, an example of the code generated by FrancaCCG is presented. First, the two Franca IDL `*.fidl` files used as input to FrancaCCG is presented. Subsequently, the D-Bus XML Introspection file generated by FrancaCCG is shown. Finally, the seven files containing C code generated by FrancaCCG corresponding to the interface is presented. These correspond to a common header file, as well as three files corresponding to header, source and example implementation of the server stub used by the server, and three files corresponding to header, source and example implementation of the server proxy used by the client.

D.1 MediaTypes.fidl

```
package org.genivi.mediamanager
typeCollection MediaTypes {
   enumeration BackendError extends MediaManagerError { BACKEND_UNREACHABLE }
   enumeration MediaManagerError { NO_ERROR }
}
```

D.2 MediaIndexer.fidl

```
package org.genivi.mediamanager
import org.genivi.mediamanager.MediaTypes.* from "MediaTypes.fidl"
interface Indexer {
   version {
       major 1
       minor 0
   }
   <** @description: Example comment**>
   attribute IndexerStatus indexerStatus readonly noSubscriptions
   method getDatabasePath {
       out {
           String output
           IndexerError e
       }
   }
   method stopIndexing {
       out { IndexerError e }
   }

   method startIndexing {
       out { IndexerError e }
   }
   enumeration IndexerStatus {
       RUNNING
       STOPPED
       IDLE
   }
   typedef IndexerError is BackendError
}
```

D.3 MediaIndexer.xml

```
<!DOCTYPE node PUBLIC "-//freedesktop//DTD D-BUS Object Introspection 1.0//EN"
 "http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">
<node name="org.genivi.mediamanager">
   <interface name="org.genivi.mediamanager.Indexer">
      <property access="read" name="indexerStatus" type="u">
         <annotation name="com.pelagicore.FrancaCCodeGen.Enum.IndexerStatus.RUNNING" value="0"/>
         <annotation name="com.pelagicore.FrancaCCodeGen.Enum.IndexerStatus.STOPPED" value="1"/>
         <annotation name="com.pelagicore.FrancaCCodeGen.Enum.IndexerStatus.IDLE" value="2"/>
         <annotation name="com.pelagicore.FrancaCCodeGen.NoSubscriptions" value="True"/>
      </property>
      <method name="getDatabasePath">
         <arg direction="out" name="output" type="s">
         </arg>
         <arg direction="out" name="e" type="u">
            <annotation name="com.pelagicore.FrancaCCodeGen.Enum.IndexerError.BACKEND_UNREACHABLE" value="0"/>
            <annotation name="com.pelagicore.FrancaCCodeGen.Enum.IndexerError.NO_ERROR" value="1"/>
         </arg>
      </method>
      <method name="stopIndexing">
         <arg direction="out" name="e" type="u">
            <annotation name="com.pelagicore.FrancaCCodeGen.Enum.IndexerError.BACKEND_UNREACHABLE" value="0"/>
            <annotation name="com.pelagicore.FrancaCCodeGen.Enum.IndexerError.NO_ERROR" value="1"/>
         </arg>
      </method>
      <method name="startIndexing">
         <arg direction="out" name="e" type="u">
            <annotation name="com.pelagicore.FrancaCCodeGen.Enum.IndexerError.BACKEND_UNREACHABLE" value="0"/>
            <annotation name="com.pelagicore.FrancaCCodeGen.Enum.IndexerError.NO_ERROR" value="1"/>
         </arg>
      </method>
   </interface>
</node>
```

D.4 MediaIndexer_common.h

```
typedef enum {
    IndexerError_BACKEND_UNREACHABLE = 0,
    IndexerError_NO_ERROR = 1,
} IndexerError_type;

typedef enum {
    IndexerStatus_RUNNING = 0,
    IndexerStatus_STOPPED = 1,
    IndexerStatus_IDLE = 2,
} IndexerStatus_type;
```

D.5 MediaIndexer_proxy.h

```
#include "MediaIndexer_common.h"
#include <glib.h>
#include <gio/gio.h>


void OrgGeniviMediamanagerIndexer_createForBus (GBusType bus_type, GDBusProxyFlags flags, const gchar *name, const gchar *
      objectPath, const GAsyncReadyCallback slot);

GDBusProxy* OrgGeniviMediamanagerIndexer_createforBusFinish(GAsyncResult* result);

void OrgGeniviMediamanagerIndexer_connectToPropertiesChanged(GDBusProxy *proxy, void(*callback)(GDBusProxy*, GVariant*,
      const gchar* const*, gpointer));


void OrgGeniviMediamanagerIndexer_getDatabasePath____s_u(GDBusProxy *proxy, const GAsyncReadyCallback callback);
void OrgGeniviMediamanagerIndexer_getDatabasePath____s_u_finish(GDBusProxy *proxy, const gchar * *out_output_out0,
      IndexerError_type *out_e_out1, GAsyncResult *result, gboolean *success);

void OrgGeniviMediamanagerIndexer_stopIndexing____u(GDBusProxy *proxy, const GAsyncReadyCallback callback);
void OrgGeniviMediamanagerIndexer_stopIndexing____u_finish(GDBusProxy *proxy, IndexerError_type *out_e_out0, GAsyncResult
      *result, gboolean *success);

void OrgGeniviMediamanagerIndexer_startIndexing____u(GDBusProxy *proxy, const GAsyncReadyCallback callback);
void OrgGeniviMediamanagerIndexer_startIndexing____u_finish(GDBusProxy *proxy, IndexerError_type *out_e_out0, GAsyncResult
       *result, gboolean *success);


void OrgGeniviMediamanagerIndexer_indexerStatus_u_set(GDBusProxy *proxy, IndexerStatus_type value, const
      GAsyncReadyCallback callback);
void OrgGeniviMediamanagerIndexer_indexerStatus_u_set_finish(GDBusProxy *proxy, GAsyncResult *result, gboolean *success);
void OrgGeniviMediamanagerIndexer_indexerStatus_u_get(GDBusProxy *proxy, const GAsyncReadyCallback callback);
void OrgGeniviMediamanagerIndexer_indexerStatus_u_get_finish(GDBusProxy *proxy, IndexerStatus_type *value, GAsyncResult *
      result, gboolean *success);
```

D.6 MediaIndexer_proxy.c

```
#include "MediaIndexer_proxy.h"
#include <stdio.h>
#include <stdlib.h>


void OrgGeniviMediamanagerIndexer_getDatabasePath____s_u(GDBusProxy *proxy, const GAsyncReadyCallback callback) {

    g_dbus_proxy_call(
        proxy,
        "getDatabasePath",
        NULL,
        G_DBUS_CALL_FLAGS_NONE,
        -1,
        NULL,
        callback,
        NULL);
}

void OrgGeniviMediamanagerIndexer_getDatabasePath____s_u_finish (GDBusProxy *proxy, const gchar * *out_output_out0,
       IndexerError_type *out_e_out1, GAsyncResult *result, gboolean *success) {
    // Get result from method call from d-bus proxy
    GVariant *wrapped;
    GError *error = NULL;
    wrapped = g_dbus_proxy_call_finish(proxy, result, &error);

    if (error != NULL) {
        printf("WARNING: Method call to getDatabasePath____s_u did not succeed.\nGError content: %s\n", error->message);
        *success = FALSE;
    } else {

        // Put results from method call in parameter

        GVariant *output_out0_variant;
        output_out0_variant = g_variant_get_child_value(wrapped, 0);
        *out_output_out0 = g_variant_get_string(output_out0_variant, NULL);

        GVariant *e_out1_variant;
        e_out1_variant = g_variant_get_child_value(wrapped, 1);
        *out_e_out1 = g_variant_get_uint32(e_out1_variant);

        *success = TRUE;
    }
}

void OrgGeniviMediamanagerIndexer_stopIndexing____u(GDBusProxy *proxy, const GAsyncReadyCallback callback) {

    g_dbus_proxy_call(
        proxy,
        "stopIndexing",
        NULL,
        G_DBUS_CALL_FLAGS_NONE,
        -1,
        NULL,
        callback,
        NULL);
}

void OrgGeniviMediamanagerIndexer_stopIndexing____u_finish (GDBusProxy *proxy, IndexerError_type *out_e_out0, GAsyncResult
        *result, gboolean *success) {
    // Get result from method call from d-bus proxy
    GVariant *wrapped;
    GError *error = NULL;
    wrapped = g_dbus_proxy_call_finish(proxy, result, &error);

    if (error != NULL) {
        printf("WARNING: Method call to stopIndexing____u did not succeed.\nGError content: %s\n", error->message);
        *success = FALSE;
    } else {

        // Put results from method call in parameter

        GVariant *e_out0_variant;
        e_out0_variant = g_variant_get_child_value(wrapped, 0);
        *out_e_out0 = g_variant_get_uint32(e_out0_variant);

        *success = TRUE;
    }
}

void OrgGeniviMediamanagerIndexer_startIndexing____u(GDBusProxy *proxy, const GAsyncReadyCallback callback) {
    g_dbus_proxy_call(
        proxy,
        "startIndexing",
        NULL,
        G_DBUS_CALL_FLAGS_NONE,
        -1,
        NULL,
        callback,
        NULL);
}

void OrgGeniviMediamanagerIndexer_startIndexing____u_finish (GDBusProxy *proxy, IndexerError_type *out_e_out0,
       GAsyncResult *result, gboolean *success) {
    // Get result from method call from d-bus proxy
    GVariant *wrapped;
    GError *error = NULL;
    wrapped = g_dbus_proxy_call_finish(proxy, result, &error);

    if (error != NULL) {
        printf("WARNING: Method call to startIndexing____u did not succeed.\nGError content: %s\n", error->message);
        *success = FALSE;
    } else {

        // Put results from method call in parameter

        GVariant *e_out0_variant;
        e_out0_variant = g_variant_get_child_value(wrapped, 0);
        *out_e_out0 = g_variant_get_uint32(e_out0_variant);

        *success = TRUE;
    }
}


// This function is called from implementation to create a proxy for the specified D-bus service
void OrgGeniviMediamanagerIndexer_createForBus (GBusType bus_type, GDBusProxyFlags flags, const gchar *name, const gchar *
       objectPath, const GAsyncReadyCallback slot) {
    g_dbus_proxy_new_for_bus (
        bus_type,
        flags,
        NULL,
        name,
        objectPath,
        "org.genivi.mediamanager.Indexer",
        NULL, //Gcancellable
        slot, // GAsyncReadyCallback callback
        NULL); // gpointer user_data
}

// This function is called from implementation to finish creating proxy
GDBusProxy* OrgGeniviMediamanagerIndexer_createforBusFinish(GAsyncResult* result) {
    GError *error = NULL;
    GDBusProxy* proxy = g_dbus_proxy_new_for_bus_finish(result, &error);
    if (error == NULL) {
        return proxy;
    } else {
        printf("ERROR: Cannot create proxy. Server stub is possibly offline.\nGError content: %s\nClosing proxy...\n",
               error->message);
        exit(1);
        return NULL;
    }
}


void OrgGeniviMediamanagerIndexer_connectToPropertiesChanged(GDBusProxy *proxy, void(*callback)(GDBusProxy*, GVariant*,
       const gchar* const*, gpointer)) {
    g_signal_connect (proxy,
                      "g-properties-changed",
                      G_CALLBACK(callback),
                      NULL);
}


void OrgGeniviMediamanagerIndexer_indexerStatus_u_set(GDBusProxy *proxy, IndexerStatus_type value, const
       GAsyncReadyCallback callback) {

    g_dbus_proxy_call(
        proxy,
        "org.freedesktop.DBus.Properties.Set",
        g_variant_new ("(ssv)", "org.genivi.mediamanager.Indexer", "indexerStatus", g_variant_new_uint32(value)),
        G_DBUS_CALL_FLAGS_NONE,
        -1,
        NULL,
        callback,
        NULL);

}

void OrgGeniviMediamanagerIndexer_indexerStatus_u_set_finish (GDBusProxy *proxy, GAsyncResult *result, gboolean *success)
       {

    // Get result from method call from d-bus proxy
    GVariant *wrapped;
    GError *error = NULL;
    wrapped = g_dbus_proxy_call_finish(proxy, result, &error);

    if (error != NULL) {
        printf("WARNING: Method call to indexerStatus_u_set did not succeed.\nGError content: %s\n", error->message);
        *success = FALSE;
    } else {
        *success = TRUE;
    }
}

void OrgGeniviMediamanagerIndexer_indexerStatus_u_get(GDBusProxy *proxy, const GAsyncReadyCallback callback) {

g_dbus_proxy_call(
    proxy,
    "org.freedesktop.DBus.Properties.Get",
    g_variant_new ("(ss)", "org.genivi.mediamanager.Indexer", "indexerStatus"),
    G_DBUS_CALL_FLAGS_NONE,
    -1,
    NULL,
    callback,
    NULL);
}

void OrgGeniviMediamanagerIndexer_indexerStatus_u_get_finish(GDBusProxy *proxy, IndexerStatus_type *value, GAsyncResult *
       result, gboolean *success) {
    // Get result from method call from d-bus proxy
    GVariant *wrapped;
    GError *error = NULL;
    wrapped = g_dbus_proxy_call_finish(proxy, result, &error);

    if (error != NULL) {
        printf("WARNING: Method call to indexerStatus_u_get did not succeed.\nGError content: %s\n", error->message);
        *success = FALSE;
    } else {
        GVariant *value_variant;
        value_variant = g_variant_get_child_value(wrapped, 0);
        *value = g_variant_get_uint32(g_variant_get_variant(value_variant));
        *success = TRUE;
    }
}
```

D.7 MediaIndexer_stub.h

```
#include "MediaIndexer_common.h"
#include <glib.h>
#include <gio/gio.h>

static const char interfaceXml0[] = { Hex code for D-Bus XML Introspection redacted
};
typedef void(*getDatabasePath____s_uSignature)(const gchar **, IndexerError_type*);
typedef void(*stopIndexing____uSignature)(IndexerError_type*);
typedef void(*startIndexing____uSignature)(IndexerError_type*);

typedef struct
{
       getDatabasePath____s_uSignature getDatabasePath____s_uHandler;
       stopIndexing____uSignature stopIndexing____uHandler;
       startIndexing____uSignature startIndexing____uHandler;
} Handlers_struct;

typedef enum HandlersEnum
{
       GETDATABASEPATH____S_U, STOPINDEXING____U, STARTINDEXING____U
} Handlers_enum;

GDBusNodeInfo *nodeinfo;
guint owner_id;
Handlers_struct methodsStruct;


IndexerStatus_type indexerStatus_u;

void OrgGeniviMediamanagerIndexer_connect(GBusType bustype);

void OrgGeniviMediamanagerIndexer_dereference();

void OrgGeniviMediamanagerIndexer_on_method_call(
       GDBusConnection *connection,
       const gchar *sender,
       const gchar *object_path,
       const gchar *interface_name,
       const gchar *method_name,
       GVariant *parameters,
       GDBusMethodInvocation *invocation,
       gpointer user_data);

void OrgGeniviMediamanagerIndexer_on_bus_acquired(GDBusConnection *connection, const gchar *name, gpointer user_data);

void OrgGeniviMediamanagerIndexer_on_name_acquired(GDBusConnection *connection, const gchar *name, gpointer user_data);

void OrgGeniviMediamanagerIndexer_on_name_lost(GDBusConnection *connection, const gchar *name, gpointer user_data);

void register_handler(Handlers_enum e, void* f);

GVariant* OrgGeniviMediamanagerIndexer_on_get_property (
       GDBusConnection *connection,
       const gchar    *sender,
       const gchar    *object_path,
       const gchar    *interface_name,
       const gchar    *property_name,
       GError        **error,
       gpointer        user_data);

gboolean OrgGeniviMediamanagerIndexer_on_set_property (
       GDBusConnection *connection,
       const gchar    *sender,
       const gchar    *object_path,
       const gchar    *interface_name,
       const gchar    *property_name,
       GVariant       *value,
       GError        **error,
       gpointer        user_data);
```

D.8 MediaIndexer_stub.c

```
#include "MediaIndexer_stub.h"
#include <stdio.h>
#include <stdlib.h>

void OrgGeniviMediamanagerIndexer_connect(GBusType bustype) {

    // Create D-Bus Introspection structure from XML
    nodeinfo = g_dbus_node_info_new_for_xml(interfaceXml0, NULL);

    if (nodeinfo == NULL) {
        printf("ERROR: Couldn't create introspection data structures from XML: \n%s\n", interfaceXml0);
        exit(1);
    }

    // Start D-Bus service
    owner_id = g_bus_own_name (bustype,
                               "org.genivi.mediamanager.Indexer",
                               G_BUS_NAME_OWNER_FLAGS_NONE,
                               OrgGeniviMediamanagerIndexer_on_bus_acquired,
                               OrgGeniviMediamanagerIndexer_on_name_acquired,
                               OrgGeniviMediamanagerIndexer_on_name_lost,
                               NULL,
                               NULL);
    }

void OrgGeniviMediamanagerIndexer_dereference() {
    g_bus_unown_name (owner_id);
    g_dbus_node_info_unref (nodeinfo);
}

// When bus acquired, register interface in the d-bus connection
void OrgGeniviMediamanagerIndexer_on_bus_acquired(GDBusConnection *connection, const gchar *name, gpointer user_data){
    guint registration_id;

    static const GDBusInterfaceVTable interface_vtable =
    {
        OrgGeniviMediamanagerIndexer_on_method_call,
        OrgGeniviMediamanagerIndexer_on_get_property,
        OrgGeniviMediamanagerIndexer_on_set_property
    };


    registration_id = g_dbus_connection_register_object (connection,
                                                         "/org/genivi/mediamanager/Indexer",
                                                         nodeinfo->interfaces[0],
                                                         &interface_vtable,
                                                         NULL, /* user_data */
                                                         NULL, /* user_data_free_func */
                                                         NULL); /* GError** */

    g_assert (registration_id > 0);

}

// The following functions needs to be here. They do nothing (except printing errors) for now.
void OrgGeniviMediamanagerIndexer_on_name_acquired(GDBusConnection *connection, const gchar *name, gpointer user_data){
}

void OrgGeniviMediamanagerIndexer_on_name_lost(GDBusConnection *connection, const gchar *name, gpointer user_data){
    printf("ERROR: Lost D-Bus name: %s\n", name);
    exit(1);
}

GVariant* OrgGeniviMediamanagerIndexer_on_get_property (GDBusConnection *connection,
                                      const gchar     *sender,
                                      const gchar     *object_path,
                                      const gchar     *interface_name,
                                      const gchar     *property_name,
                                      GError         **error,
                                      gpointer         user_data){

    // TODO error handling, should use the gerror since it is already here

    GVariant *ret;
    ret = NULL;

    if (g_strcmp0 (property_name, "indexerStatus") == 0) {
        ret = g_variant_new_uint32(indexerStatus_u);
    } else
    {
        printf("ERROR: Interface does not contain property: %s\n", property_name);
        exit(1);
    }
    return ret;
}

gboolean OrgGeniviMediamanagerIndexer_on_set_property (GDBusConnection *connection,
                                     const gchar    *sender,
                                     const gchar    *object_path,
                                     const gchar    *interface_name,
                                     const gchar    *property_name,
                                     GVariant       *value,
                                     GError        **error,
                                     gpointer        user_data){

    if (g_strcmp0 (property_name, "indexerStatus") == 0)
    {
        if (indexerStatus_u != g_variant_get_uint32(value))
        {
            indexerStatus_u = g_variant_get_uint32(value);
            // Unless com.pelagicore.FrancaCCodeGen.Enum.NoSubscriptions annotation is set to true, send PropertiesChanged
                signal
            // Annotation is set to TRUE. Do not send signal.
        }
    } else
    {
        printf("ERROR: Interface does not contain property: %s\n", property_name);
        exit(1);
    }
    return *error == NULL;
}

// Register method handler function
void register_handler(Handlers_enum e, void* f){
    switch (e)
    {
    case GETDATABASEPATH____S_U:
        methodsStruct.getDatabasePath____s_uHandler = (getDatabasePath____s_uSignature)f;
        break;
    case STOPINDEXING____U:
        methodsStruct.stopIndexing____uHandler = (stopIndexing____uSignature)f;
        break;
    case STARTINDEXING____U:
        methodsStruct.startIndexing____uHandler = (startIndexing____uSignature)f;
        break;

    default:
        printf("ERROR: No such handler enum defined: %u\n", e);
        exit(1);
    }
}

// Handle method calls
void OrgGeniviMediamanagerIndexer_on_method_call(GDBusConnection *connection,
                                const gchar *sender,
                                const gchar *object_path,
                                const gchar *interface_name,
                                const gchar *method_name,
                                GVariant *parameters,
                                GDBusMethodInvocation *invocation,
                                gpointer user_data){
    // TODO needs to compare on other stuff (ie signature) than name, can be collisions

    if (g_strcmp0 (method_name, "getDatabasePath") == 0)
    {
        const gchar * output_out0;
        IndexerError_type e_out1;
        // Call the registered method handler, if one is registered.
        if (methodsStruct.getDatabasePath____s_uHandler != NULL) {
            methodsStruct.getDatabasePath____s_uHandler(&output_out0, &e_out1);
        } else {
            printf("ERROR: No method handler function registered for method: %s\n", method_name);
            exit(1);
        }
        //Return
        g_dbus_method_invocation_return_value (invocation, g_variant_new ("(su)", output_out0, e_out1));
    } else
    if (g_strcmp0 (method_name, "stopIndexing") == 0)
    {
        IndexerError_type e_out0;
        // Call the registered method handler, if one is registered.
        if (methodsStruct.stopIndexing____uHandler != NULL) {
            methodsStruct.stopIndexing____uHandler(&e_out0);
        } else {
            printf("ERROR: No method handler function registered for method: %s\n", method_name);
            exit(1);
        }
        //Return
        g_dbus_method_invocation_return_value (invocation, g_variant_new ("(u)", e_out0));
    } else
    if (g_strcmp0 (method_name, "startIndexing") == 0)
    {
        IndexerError_type e_out0;
        // Call the registered method handler, if one is registered.
        if (methodsStruct.startIndexing____uHandler != NULL) {
            methodsStruct.startIndexing____uHandler(&e_out0);
        } else {
            printf("ERROR: No method handler function registered for method: %s\n", method_name);
            exit(1);
        }
        //Return
        g_dbus_method_invocation_return_value (invocation, g_variant_new ("(u)", e_out0));
    } else

    {
        printf("ERROR: No such method in registered D-Bus interface. Method name: %s\n", method_name);
        exit(1);
    }

}
```

D.9 MediaIndexer_proxyImplementation.c

```c
#include "MediaIndexer_proxy.h"
#include <stdio.h>

GDBusProxy* proxy;

// This function is called from simpleFranca_proxy when method call is finished.
// It must have same signature as a GAsyncReadyCallback! void function_name(GObject *source_object, GAsyncResult *res,
       gpointer user_data)
void on_getDatabasePath____s_u_finished(GObject *obj, GAsyncResult *result, gpointer userdata) {

    const gchar * output_out0_result;
    IndexerError_type e_out1_result;
    gboolean success = FALSE;
    OrgGeniviMediamanagerIndexer_getDatabasePath____s_u_finish(proxy, &output_out0_result, &e_out1_result, result, &
          success);

    // success now contains whether method call was successful or not
    // _result variables now contains the results of the method call.
    // Implementation goes here
}

// This function is called from simpleFranca_proxy when method call is finished.
// It must have same signature as a GAsyncReadyCallback! void function_name(GObject *source_object, GAsyncResult *res,
       gpointer user_data)
void on_stopIndexing____u_finished(GObject *obj, GAsyncResult *result, gpointer userdata) {

    IndexerError_type e_out0_result;
    gboolean success = FALSE;
    OrgGeniviMediamanagerIndexer_stopIndexing____u_finish(proxy, &e_out0_result, result, &success);

    // success now contains whether method call was successful or not
    // _result variables now contains the results of the method call.
    // Implementation goes here
}

// This function is called from simpleFranca_proxy when method call is finished.
// It must have same signature as a GAsyncReadyCallback! void function_name(GObject *source_object, GAsyncResult *res,
       gpointer user_data)
void on_startIndexing____u_finished(GObject *obj, GAsyncResult *result, gpointer userdata) {

    IndexerError_type e_out0_result;
    gboolean success = FALSE;
    OrgGeniviMediamanagerIndexer_startIndexing____u_finish(proxy, &e_out0_result, result, &success);

    // success now contains whether method call was successful or not
    // _result variables now contains the results of the method call.
    // Implementation goes here
}

void on_OrgGeniviMediamanagerIndexer_indexerStatus_u_set_finished(GObject *obj, GAsyncResult *result, gpointer userdata) {
    gboolean success = FALSE;

    OrgGeniviMediamanagerIndexer_indexerStatus_u_set_finish(proxy, result, &success);
    // success now contains whether method call was successful or not
}

void on_OrgGeniviMediamanagerIndexer_indexerStatus_u_get_finished(GObject *obj, GAsyncResult *result, gpointer userdata) {
    IndexerStatus_type indexerStatus_u;
    gboolean success = FALSE;

    OrgGeniviMediamanagerIndexer_indexerStatus_u_get_finish(proxy, &indexerStatus_u, result, &success);
    // success now contains whether method call was successful or not
    // indexerStatus_u now contains current value of property.
    // Implementation goes here

}



void on_properties_changed (GDBusProxy      *proxy,
                          GVariant          *changed_properties,
                          const gchar* const *invalidated_properties,
                          gpointer           user_data)
{
    // Implementation of the handling of the PropertiesChanged signal is done here.
    // Note that local cache is automatically updated and needs not to be implemented here.
}


// This function is called from OrgGeniviMediamanagerIndexer_proxy when proxy has been created.
// It must have same signature as a GAsyncReadyCallback! void function_name(GObject *source_object, GAsyncResult *res,
       gpointer user_data)
void proxy_created(GObject *obj, GAsyncResult *result, gpointer userdata) {
    proxy = OrgGeniviMediamanagerIndexer_createforBusFinish(result);
    // Proxy has been created.

    // Connect to signal handler.
    OrgGeniviMediamanagerIndexer_connectToPropertiesChanged(proxy, &on_properties_changed);


}

int main(int argc, char **argv) {

    // Create pointer to function to call when proxy has been created.
    GAsyncReadyCallback proxycreated_pointer = &proxy_created;

    // Connect to D-Bus service.
    OrgGeniviMediamanagerIndexer_createForBus(G_BUS_TYPE_SESSION, G_DBUS_PROXY_FLAGS_NONE, "org.genivi.mediamanager.
           Indexer", "/org/genivi/mediamanager/Indexer", proxycreated_pointer);

    // Run main loop.
    GMainLoop *mainloop = g_main_loop_new(NULL, FALSE);
    g_main_run(mainloop);
    return 0;
}
```

D.10 MediaIndexer_stubImplementation.c

```c
#include "MediaIndexer_stub.h"
void getDatabasePath____s_uImplementation(const gchar * *output_out0, IndexerError_type *e_out1){
    // Implementation of method getDatabasePath goes here
}
void stopIndexing____uImplementation(IndexerError_type *e_out0){
    // Implementation of method stopIndexing goes here
}
void startIndexing____uImplementation(IndexerError_type *e_out0){
    // Implementation of method startIndexing goes here
}
int main(int argc, char **argv) {

    // Register method callback functions
    getDatabasePath____s_uSignature getDatabasePath____s_u_pointer = &getDatabasePath____s_uImplementation;
    register_handler(GETDATABASEPATH____S_U, getDatabasePath____s_u_pointer);

    stopIndexing____uSignature stopIndexing____u_pointer = &stopIndexing____uImplementation;
    register_handler(STOPINDEXING____U, stopIndexing____u_pointer);

    startIndexing____uSignature startIndexing____u_pointer = &startIndexing____uImplementation;
    register_handler(STARTINDEXING____U, startIndexing____u_pointer);


    // Create D-Bus service
    OrgGeniviMediamanagerIndexer_connect(G_BUS_TYPE_SESSION);

    // Create and run main loop
    GMainLoop *mainloop = g_main_loop_new(NULL, FALSE);
    g_main_loop_run (mainloop);

    OrgGeniviMediamanagerIndexer_dereference();

    return 0;
}
```