Integrating Java with a Matlab environment at Studsvik Scandpower

Karl Magnusson
Studsvik Scandpower
Mälardalens Högskola

June 2011

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank the following people at Studsvik Scandpower:

Thomas Smed

Kjell Adielsson

Steve Sutton

For their continuous help and support during the work on this thesis

And from Mälardalens Högskola i would like to thank

Afshin Ameri E.

For his patience and support when reviewing this thesis.

Abstract

Matlab is a programming environment that provides a large and accessible toolbox when programming applications with high level of mathematical content. However it is a programming environment that targets a specific type of application and currently does not provide wide usability when implementing graphics with emphasis on user interfaces and interactive design. Java is a well established development environment suitable for any type of graphical programming. Together Matlab and Java make it possible to create applications with advanced mathematical computation as well as a well designed graphical interface. This paper discusses ways to integrate Java with Matlab and how to adapt an existing Matlab program in order to make it accessible when designing a Java based graphical interface.

Chapter 1

Introduction

This paper discusses the task of integrating Java with a Matlab program at Studsvik Scandpower. The program in question presents a graphical view of calculated data on the core of nuclear power-plants. The purpose of the paper is to investigate possible ways of achieving integration between the Matlab program and Java, in order to create a Java graphical user interface, and present a solution with a proof of concept program.

1.1 Background

Studsvik Scandpower is an engineering company with focus on nuclear engineering. They have numerous softwares to calculate and graphically represent data from nuclear power-plants. One of their most essential softwares is a program that gathers information from the core of a nuclear power-plant and provides a graphical representation of the core with calculated data. This program is written in Matlab and the company feels that the graphics available when programming in Matlab is limited, therefore they would like to investigate the possibilities of integrating Java with the program. It is important to have a distinct separation of the program with the data management written in Matlab and the graphics written in Java. When managing the program the engineers writing their code in Matlab should not also have to adapt the Java code.

1.2 Previous work

In the paper "How to access MatLab from Java" Andreas Klime [1] presents how to start a Matlab session using the Java Runtime Class. Communication is achieved using the standard input/output stream in Matlab. The paper discusses the advantages and disadvantages with such an approach. Advantages are that it is platform-independent and will be compilable with virtually any Matlab version. It does not demand extensive installation procedures or system setups. Disadvantages are that the transfer of data is stream based thus demanding parsing of the Matlab output stream. It also describes an alternative method using the Java Native Interface (JNI) to create a wrapper for Matlab's C engine. The process has advantages since the Matlab C engine provides functions for sending and retrieving arrays from Matlab. However the method is not platform-independent and implementation itself is inconvenient.

In the paper "Easy Java Simulations: an Open-Source Tool to Develop Interactive Virtual Laboratories Using MATLAB/Simulink" [2] the authors describes how to use Easy Java Simulation[3] with Matlab to develop interactive systems using java to construct the Graphical interface and Matlab to make as the computational machine. The tool is a free software written in Java that is meant to help non Java programmers to create interactive simulations in Java. It has an interface which enables for convenient communication between the Java application and Matlab functions. It also provides a solution for importing MatLab graphics such as Figures to the Java application.

Matlab control[4] is a Java API that provides tools for controlling and interacting with both local and remote MATLAB sessions. To control Matlab from Java they use a proxy class within the Java code. This proxy class has a number of useful functions for interaction between the Matlab session and the Java class such as receiving and setting Matlab variables and writing a Matlab command and receiving the output stream. JMatLink [5] is a Java application similar to Matlab control. It uses an engine class to communicate with Matlab through the MatLab command prompt.

MatLab builder Ja[6] is a MathWorks(creators of MatLab) product. It enables for creation of Java classes from a MatLab program. The builder encrypts MATLAB functions and generates a Java wrapper around them so that they behave like a Java class. It provides an API for conversion between MatLab and Java datatypes. MatLab figure zooming, rotating, and panning is made available through a Web Figures interface. In the MatLab builder JA's user guide[7] there are discussions on how to write MatLab code suitable for deployment to Java and they also discuss where the responsibilities of implementation lies when writing a Java and Matlab integrated system.

1.2.1 Summary of previous work

The task of integrating Matlab and Java has been addressed by several papers and softwares. There are some open source programs that provide a Java to Matlab (JMI) interface, JMatlink and Matlab mentioned in previous work are two of the most used JMI software's. These JMI interfaces uses the same basic concept of starting a Matlab session from Java and communicating with Matlab through the standard outpout/input stream in Matlab. The problem with such approaches is the parsing of data between the Matlab and Java program. Mathworks, the creator of Matlab, provides somewhat different approaches to achieve the intergeneration. Matlab has built-in support for calling Java classes and the extension tool Matlab builder JA translates Matlab code to Java, thus removing the need of having to start a Matlab session from Java as used by most JMI.

Chapter 2

Technical Introduction

This chapter describes possible ways of integrating Matlab and Java. It describes the program at Studsvik Scandpower on which the integration is to be achieved. Using the program as point of reference it discusses what method to be used when implementing the prototype.

2.1 Ways of integrating Java and Matlab

As discussed in the previous work section, there are many ways of integrating Java and Matlab. The access of Java within Matlab would seem an efficient way since it does not demand any additional software to implement. An issue with such an approach is that when implementing a Graphical user interface the user input, mouse clicks and keyboard presses etc, are typically handled by the Java program by the use of event listeners attached to the Java window. Matlab supports conversion of data types between Matlab and Java. This can be used to implement a parser functionality on the Matlab session that receives the user input from Java to Matlab and then interoperate the input and make changes to the Java graphical user interface. This specific approach of integrating Java and Matlab can be viewed as a Client Server architecture (see figure 2.1) where Matlab acts as the server on which the data is stored and computed on and Java as the client from which the user can receive output and send input using the GUI.

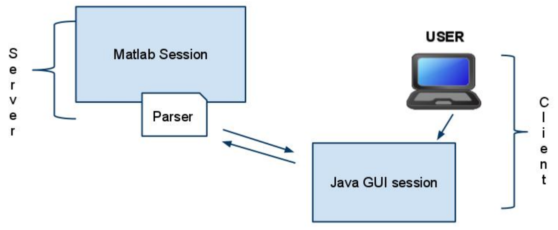


Figure 2.1: Client Server

The use of a JMI to integrate Java and Matlab would have a similar structure as if Matlab would call a Java class. It also needs two sessions, one Java and one Matlab. However in the case of a JMI the Java session controls the Matlab session and not vice versa (see figure 2.2). If data are to be passed between the two sessions, a parser that translates data types between Java and Matlab must be implemented. How and where the data of the application is stored and received must also be considered.

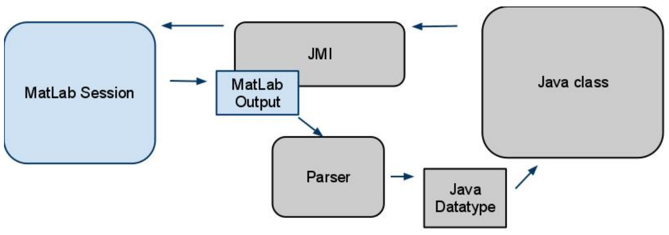


Figure 2.2: JMI - Java to Matlab interface

Instead of using the standard output input stream, a database can be used. From the database both Java and Matlab can read and write. This approach has the same structure as the JMI, with two running sessions, however it removes the need of Java having to start and control the Matlab session. The data of the application would have a more significant roll in the application design, since the database and the structure of the data is directly used in the communication between the Matlab and Java session. When the common data source is changed the two running sessions must separately interpret the change and decide how to act upon it. Matlab has libraries supporting the use of mysql[8] databases that can be used for this purpose.

Matlab builder JA provides a way for a Java application to use functions written in Matlab as if they where regular Java classes. To achieve this, the builder encrypts Matlab code and generates a Java wrapper class around them. In order to compile and run Matlab code the Java application uses a Matlab runtime compiler[9]. To manage the conversion of data types between the wrapped Matlab functions and Java, the builder uses a Java parser class named MWArray. This approach makes it possible for a standalone Java application to use Matlab defined functions, as pictured in figure 2.3

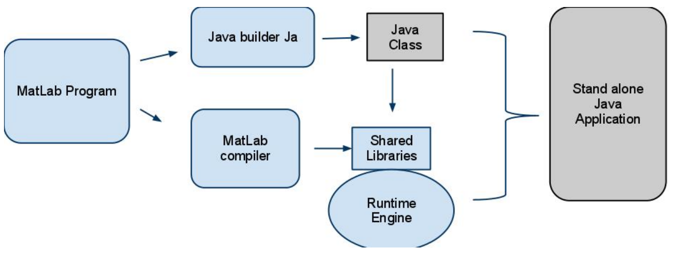
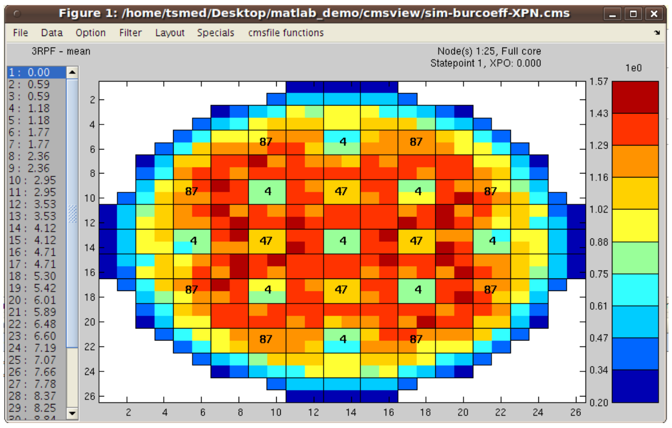


Figure 2.3: Matlab builder JA

2.2 Description of Matlab program to integrate with Java

The program to be integrated with a Java graphical user interface is named Cmsplot. Cmsplot is Matlab program developed at Studsvik Scandpower with the purpose of displaying calculated data on a nuclear core. The program displays a 2-dimensional map of a nuclear core with fuel bundles, detectors and controlrods (see figure 2.4). The fuel bundles have different colors to clarify variations on the calculated data. The nuclear core is divided into about 25 axial levels, where each level has its own calculated data. The user can choose between one of the 25 axial levels of the core and an average of them for displaying. Data that is to be displayed on the map of the core are called distribution. The distributions are divided in to a number of state-points. The user can select from a number of different distributions and which state-point in the current distribution that is to be displayed. The data of the geometry of the core and the distributions are read from file.



```
1 :  0.00
2 :  0.59
3 :  0.59
4 :  1.18
5 :  1.18
6 :  1.77
7 :  1.77
8 :  2.36
9 :  2.36
10 :  2.95
11 :  2.95
12 :  3.53
13 :  3.53
14 :  4.12
15 :  4.12
16 :  4.71
17 :  4.71
18 :  5.30
19 :  5.42
20 :  6.01
21 :  5.89
22 :  6.48
23 :  6.60
24 :  7.19
25 :  7.07
26 :  7.66
27 :  7.78
28 :  8.37
29 :  8.25
```

Figure 2.4: Cmsplot

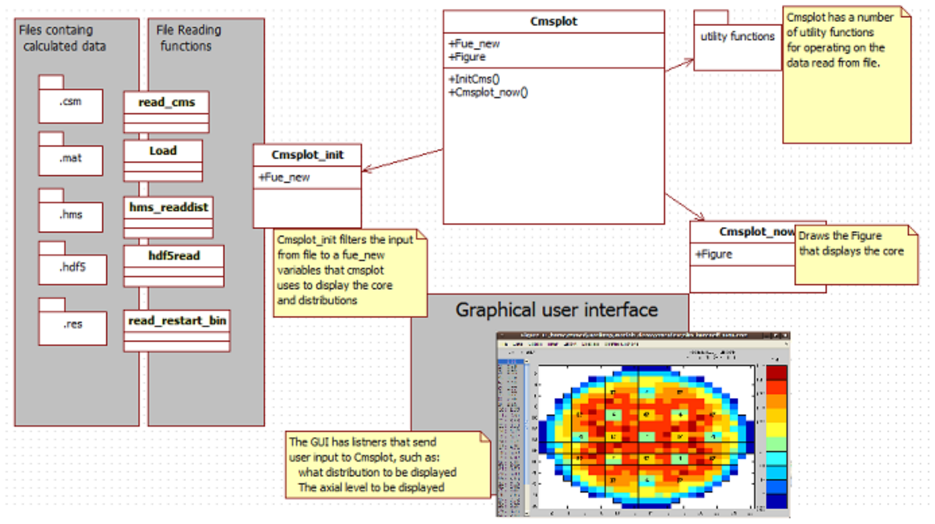

Files containg calculated data
.csm
.mat
.hms
.hdf5
.res
File Reading functions
read_cms
Load
hms_readdist
hdf5read
read_restart_bin
Cmsplot_init
+Fue_new
Cmsplot_init filters the input from file to a fue_new variables that cmsplot uses to display the core and distributions
Cmsplot
+Fue_new
+Figure
+InitCms()
+Cmsplot_now()
utility functions
Cmsplot has a number of utility functions for operating on the data read from file.
Cmsplot_now
+Figure
Draws the Figure that displays the core
Graphical user interface
The GUI has listners that send user input to Cmsplot, such as:
what distribution to be displayed
The axial level to be displayed

Figure 2.5: Cmsplot program UML

Figure 2.5 is a model of the Cmsplot program. The file types containing the distributions and coregeometry data are read from file through the initializing function Cmsplot_init. There are a number of different file types each with its own file reading function. The main function of the program is Cmsplot, it contains the Fue_new variable which is where the geometry data and distribution data are contained after it is read from file through Cmsplot_init. It also contains a figure variable that is used to draw graphics. Cmsplot now is a function that draws the coremap with the current distribution. Cmsplot uses a number of utility functions for manipulating and calculating on the data displayed in the coremap window. A specific fuelbundle distribution can be plotted in a coordinate system and an average of the axial levels on each bundle can be calculated and drawn with a color scheme. The distribution data is stored in a three dimensional matrix with the following axes:

distribution[statepoint][bundle][axial level]

Where each index in the matrix contains a double precision float of 64 bits. The distributions contain an average of 30 statepoints. The number of bundles are an average of 500 and the number of axial levels up to 25. Cmsplot has a number of functions for manipulating the data. For example creating a coremap of the distribution at a given statepoint.

2.3 Discussion of choice of method for implementing proof of concept program

When choosing what method to use for integrating a Java graphical user interface with the cmsplot program a number of issues had to be taken into consideration. The work methodology that Studsvik Scandpower uses in software projects is perhaps the most important issue to consider. An ideal method would be for the company's engineers to work only in Matlab, defining functions and algorithms and also a prototype GUI using Matlab graphics. The purpose being both to use Matlab suitability for writing high level mathematical content as well as allowing the engineers to use a program environment that they are experienced in. The program defined in Matlab would then be used by Java GUI oriented programmers. They would reuse the algorithms written in Matlab and using the prototype GUI as reference, create a java GUI. This would then be compiled in to a stand alone application suitable for distribution. Having this in mind we will have to look at the Cmsplot program and determine how an integration of this program would be achieved using the methods described in section (2.1). The main issues that a program design must address are:

- Where and how should the Geometry and Distribution data be stored
- What level of communication between Matlab and Java is necessary
- How is the difference between data types in Matlab and Java solved.
- How would this be compiled into a standalone application suitable for distribution.

If Java is called from within Matlab the data of the distributions and geometry could be stored within the Matlab session. The necessary data to draw a coremap with distribution could be translated to a Java GUI session using a parser functionality. The Java GUI session would provide interaction with the user. The user input from the GUI would also have to be sent back to the Matlab session that would interoperate the user input. This could be compiled into a standalone program, using Matlab Compiler [10] if the necessary Java files, compiler and libraries etc is bundled[11] together with the Matlab program. To use two sessions in a program solution often demands a more extensive fail control. The parser functionality is an important and complicated functionality to implement and must be adapted whenever a change to the program data is made.

The use of a JMI has to deal with the problem on what session to store the data of the distribution and geometry. Matlab functions from csmplot would be responsible for reading data from file and make computation on the data. The java session only needs to have information on the distribution at the current statepoint. It would no be sustainable for the Java session to receive and parse a whole distribution at time, too much data are contained within the distribution for this to be a viable approach. When the user makes an input using the GUI, the Java session should interoperate what data it needs to update and call the Matlab session, the returned data would then need to be parsed to a Java datatype and finally the Java GUI can update with the new information. To create a standalone program suitable for distribution the Matlab program must be compiled into a standalone program using Matlab Compiler. The Java program can be compiled into a executable jar file. This method would demand extensive work on the functionality that parses the Matlab datatypes to Java. A structured way of returning data from Matlab is necessary to decrease the amount of datatypes that the parser needs to be able to handle.

The use of a common database source to achieve integration between Matlab and Java would be very similar to the JMI approach. The common database source would have a structure containing the geometry with one distribution at a given statepoint. It would also need data describing the user input, in order for the Matlab session to know what changes should be made on the database. This would demand parser functionality on both sides, in order for the two sessions to use the data contained in the database.

Matlab builder JA has the great advantage that it does not need two sessions. The Matlab code is translated into Java and a parser functionality is available through MWArray. Necessary Matlab functions for reading and manipulating data could be translated using the builder. The program can be compiled to executable jar file. Together with a Matlab runtime Compiler [10], needed by the program to use Matlab functions at runtime, a standalone program suitable for distribution can be constructed. This would be in line with the company's preferred working methodology. Cmsplot can be used, without extensive rewriting, by a Java programmer focusing on the GUI.

Conclusion

The discussed methods have to be compared with consideration of how they address the compatibility issues mentioned earlier in this section as well as how suitable the methods are with consideration of the company's preferred working methodology. Matlab builder JA is the most suitable and time efficient. Since Matlab builder JA has an implemented parser functionality, it will save time from extensive work on implementation. It is also the most efficient method for creating a standalone program suitable for distribution. This is mainly because the method does not need two sessions in the program solution. The fact that there is only a Java session also removes the issue of deciding on what session data should be stored. The method is in line with the company's preferred work methodology. A program can be written in Matlab with a prototype GUI, necessary functions from the Matlab program can be translated using the builder and used in a Java program. With the above arguments, the method for implementing the proof of concept program described in the following chapter is Matlab builder JA.

Chapter 3

Method

This section describes the implementation of the proof of concept program, named CoreJavaMatlab, that uses functions from Cmsplot for reading and computing on data and a Java graphics library s3rview, based on Java Jframe[11], for drawing graphics. The integration of Matlab and Java is achieved through the use of Matlab builder JA.

3.1 Requirements analysis

CoreJavaMatlab is a proof of concept program and therefore needs to perform only the fundamental parts of the cmsplot program.

- A: Read data from at least one of the file types used by cmsplot
- B: Display the geometry of the core with controlrods and bundles
- C: Display distributions read from file with a color scheme on the bundles
- D: Display one axial level a time or an average of the axial levels.
- E: Read a distribution selected by the user and display it.
- F: Create an install-file suitable for distribution

A: Read data from at least one of the file types used by cmsplot

As pictured in figure 2.5 cmsplot reads data from several different filetypes. Each filetype has its own file reading function. The proof of concept program only needs to be able to read data from one of the filetypes used by cmsplot. This would suffice in accomplishing task of using Matlab defined functions from cmsplot for reading data.

B: Display the geometry of the core with controlrods and bundles

The program must be able to display a core with the correct geometry and with controlrods position. The geometry of the core and the positions of the controlrods are contained on file which is read by matlab functions from cmsplot (see requirement A). This is fundamental for the program to correctly display the calculated values which is the purpose of cmsplot and CoreJavaMatlab.

C: Display distributions read from file with a color scheme on the bundles

The distributions contains a number of statepoints, each statepoint contains values on the bundles of the coremap. These values must be attached to the correct bundle. The bundles must have a color scheme to clarify the difference of the bundle values.

D: Display one axial level a time or an average of the axial level on each bundle

The bundles of the core are divided into several axial levels, often 25. From the GUI the user must be able to select what axial level to display at any given statepoint or an average of the axial levels on each bundle with value and color scheme.

E: Read a distribution selected by the user and display it

The files from which the program reads data, contain a various number of distribution. The user must be able to select what distribution to display in the GUI.

F: Create a standalone runnable application with install file

The program should be packaged into a standalone program with an install file, runnable on Windows and Linux.

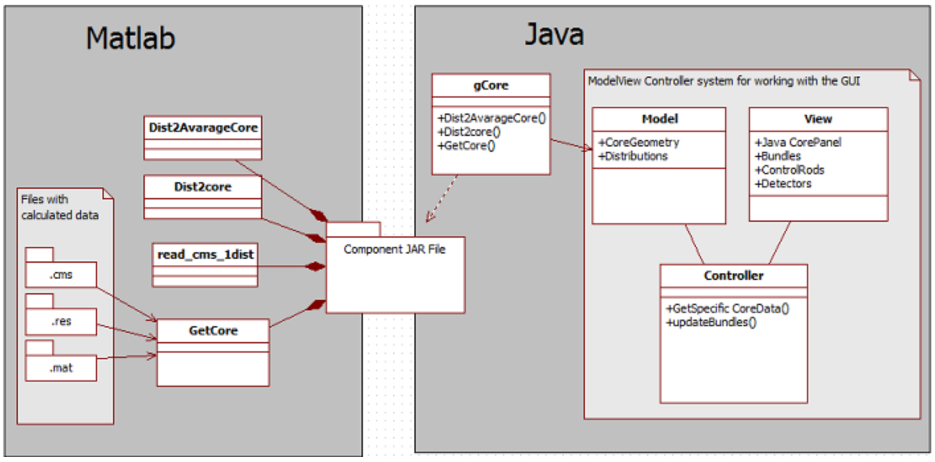


Figure 3.1: UML model of CoreJavaMatlab

3.2 Program design

The design of CoreJavaMatlab(CJM) is developed with the idea of using functions from Cmsplot for reading and computing on data, and Java to display graphics.
Figure 3.1 is a UML model displaying the design of CoreJavaMatlab. It displays the Matlab defined functions that the program uses as well as the Java classes. The Matlab functions are deployed into a component jar file, through Matlab builder JA. The Matlab functions displayed are created for CoreJavaMatlab and encapsulate functions from Cmsplot. The implementation section of the thesis contains a detailed description on the functions and classes described in the UML model together with description of how Cmsplot is incorporated in the program design of CoreJavaMatlab.

The design of CJM is based on a Model View Controller program architecture. It is a software architectural pattern often used in software engineering, especially popular when programming graphical user interfaces. The pattern separates an application in three isolated parts where each part can be updated and maintained separately.
Model contains the data describing the core as well as the distributions.
View displays the data from the model in a graphical user interface. It uses the Java Jframe [12] class ,to create a window, and a number of classes, such as the bundle and corepanel class, to display components of the core in the window. View has listeners attached to the window that detects user interaction, mousklick, keypress etc, and notifies the controller when an interaction is made.
Controller The controller contains the logic of the application. It collects information from the model about the geometry of the core and instructs the View how to display the core. Depending on the input retrieved from the user interaction with View the Controller updates the information on the bundles and the color of the bundles.

Figure 4.1 is a use-case diagram that displays the work flow of the program when a user selects a file containing core geometry and distributions that the program then displays in the graphical user interface.
The user selects a file in the graphical user interface, controlled by the view class. The input is sent to the controller class. The controller interprets the input and decides to read data from file. Controller then tells the model class to update it's data. Model uses the component jar file encapsulating Matlab functions to read data from file and update data it currently holds. When model has the correct data according to the user input it sends the current data, to be drawn in the user interface, to View.

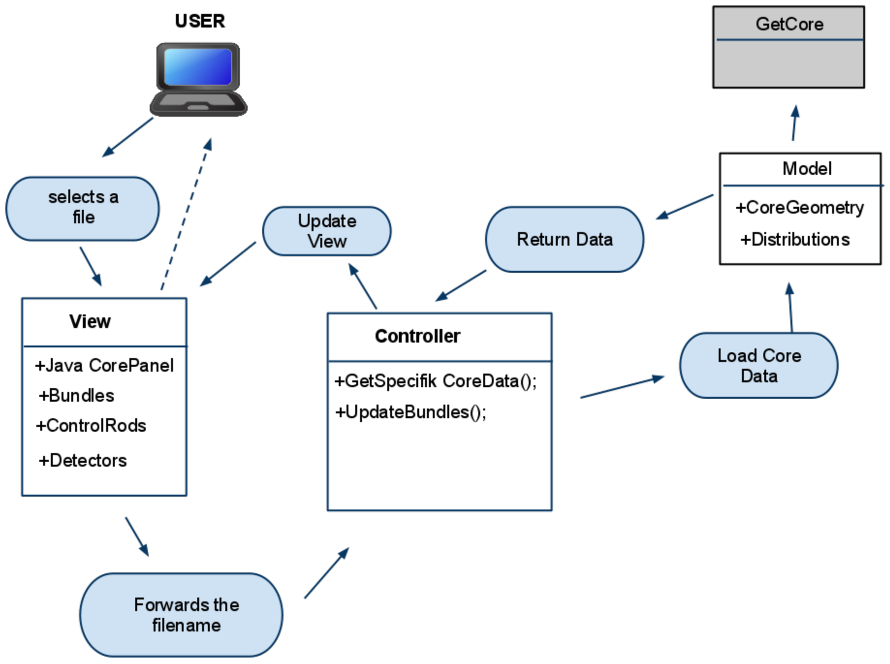


Figure 3.2: Use case diagram

3.3 Implementation

3.3.1 Matlab functions

As displayed in 3.1 CJM contains a number of Matlab functions. The functions use parts of code from cmsplot functions or entire cmsplot functions. The functions are deployed to a component JAR file, through Matlab builder JA. The builder also generates a Java class that provides an interface to the deployed functions. The class generated by the builder has been named gCore. The following sections describes the Matlab functions encapsulated by gCore.

getCore

The getCore function takes a filename as input and outputs a data structure containing the geometry of the core nessecary for displaying a correct core as well as a list of distributions associated with that filename. To achieve this it uses file reading functions from cmsplot. In cmsplot the data structure fue_new contains the corresponding information however the structure differs depending on what filetype it reads the data from. This is not suitable to directly deploy to Java. The structure that getCore outputs:

```
 CoreGeomtry=
   mminj: - Core contour
      ij: - Core coordinates
   crmninj: - Control rod contour
conrodcoor: - Control rod coordinate
  iafull: - Span of core
      kmax: - Distribution (dimension: kmax by kan)
       kan: - Distribution (dimension: kmax by kan)
      knum: - Vector of channel numbers for full symmetry.
         Matrix for symmetric case, then each row
          contains symmetric channel numbers
       sym: - Sym ['FULL', halfcore: 'N', 'S', 'E', 'W'
        or quarter-core 'NE','NW','SE',SW']
     if2x2: - If if2x2=2, then assembly is divided
        in 2-by-2 nods
     ihave: - Symmetry value, full core = 1,
        1/2 Core = 2, etc
      irmx: - Number of controlrods
   off_set: - length(mminj/2-irmx)
    konrod: - Control rod positions
    detloc: - detector location
  distlist: - List of distribution names
```

To read from file getCore calls upon an initiation function specific for each filetype. InitRes for the .RES file type and InitCms for the .CMS file type. These initiation functions are a separation of the function Cmsplot_init from Cmsplot which can take a number of different file types and then call upon the file reading function associated with that file type. InitCms and InitRes calls the same file reading functions as Cmsplot_init, read_restart_bin and read_cms. These functions return a data structure with information on the core geometry and distribution and various of other of other data into a temporary variable. getCore uses the data in the temporary variable to collect the information that the data structure (see above) must contain.
Example source-code from getCore:

```
    case '.cms'
        CmsData = InitCms(filename);

        % - Dimension
        core.mminj=CmsData.mminj;
        core.iafull=CmsData.iafull;
        core.ihave=CmsData.ihave;
        core.kmax=CmsData.kmax;
        core.knum=CmsData.knum;
        core.kan=CmsData.kan;
        core.sym=CmsData.sym;

dlist=CmsData.cmsinfo.DistNames(:,2);
        dlist(cellfun(@isempty,dlist))=[];
        core.distlist=dlist;
```

read_cms_1dist

The function read_cms_1dist takes a distribution name, file type and a state point as input and returns the distribution data and information of the dimension of the distribution. The function reads the distribution data through the function read_cms_dist from Cmsplot.
Source code:

```
function [data,dmin,dmax]=read_cms_1dist(cmsinfo,dist,N)
temp=read_cms_dist(cmsinfo,dist);
data=temp{N};
dmin=min(data(:));
dmax=max(data(:));
```

dist2core

The function dist2core takes a distribution with a state point and geometry information as input and outputs a coremap with distribution data on each bundle in the coremap. The function uses vec2core, a function from Cmsplot that takes a three dimensional vector and the contour of the core as input and outputs a coremap. The coremap is usable when drawing the core correctly with the positions of the the bundles and the distribution data associated with each bundle.
Source code:

```
function out =dist2core(power,index,kan,mminj)
        a=power(index,1:kan);
        out=vec2cor(a,mminj);
```

dist2AvarageCore

The function dist2AvarageCore outputs a coremap with an average of the distribution data of the axial level on each bundle.
Source code:

```
function out = dist2AvarageCore(power,kmax,kan,mminj)

    dump = 0;
    for s=1:1:kmax
        a =power(s,1:kan);
        dump = dump + a;
    end
    output=0;
    for t=1:1:kan
            output(t)=dump(t)/kmax;
    end
    out = vec2cor(output,mminj);
```

getDistLength

The getDistLength function returns the number of state points of a distribution. Source code

```
function distln = getDistLength(cmsinfo,dist)
temp=read_cms_dist(cmsinfo,dist);
distln=length(temp);
```

3.3.2 Java classes

This section describes the Java classes of CJM. As explained in the design section the software architecture pattern used is Model View Controller. The interaction between them is best explained through the use case diagram in figure 4.1.

Model

Model contains the data of the application that View and Controller acts upon. Since the data is read and computed on through Matlab functions, model contains an instance of the class gCore, generated by Matlab builder JA, that provides an interface with the deployed Matlab functions.
source code:

```
 public MatlabJavaConvertioner() {
        try {
                // Instansera core objectet
                core = new gCore();
}
                catch (MWException e) {
                    text = "could not initiate code";
                }
    }
```

The data of the current distribution to be displayed is contained in a member variable currentData and the geometry is contained in an MWStructArray instance named sCore.

```
  MWStructArray sCore;
double[][] currentData;
```

Model has a number of member functions that use the deployed functions to read data on geometry and distributions from file:

```
 public boolean LoadCoreData(String filename) throws MWException{
         try {
             result = core.getCore(1, filename);
             sCore = (MWStructArray)result[0];
             currentData=(double[][])this.getSpecificCoredata("power").toArray();
             return true;
            }
         catch (MWException e){
             return false;
         }
    }
```

To read a distribution from file:

```
 public MWArray getDistribution(int axis) throws MWException{

        //axis,sCore.getField("kan",1),sCore.getField("mminj",1)
        MWArray A = new MWNumericArray(axis);
        MWArray d = this.getSpecificCoredata("kan");
        MWArray e = this.getSpecificCoredata("mminj");
        Object[] res =  core.dist2core(1,currentData,A,d,e);
        MWArray ret= (MWArray)res[0];
       return  ret;
    }
```

To get specific core data from Matlab, model uses a class that searches data with a specific field name.

```
v public MWClassID getSpecificCoreDataType(String fieldname){
     return sCore.getField(fieldname,1).classID();
    }
```

When the GUI is to display an average of the axial levels, Model uses the getAvarageDistribution class. The class calls the deployed Matlab function dist2AvarageCore.

```
   public MWArray getAvarageDistribution(String distribution) throws MWException{

        MWArray d = this.getSpecificCoredata("kan");
        MWArray e = this.getSpecificCoredata("mminj");
        MWArray f = this.getSpecificCoredata("kmax");
        Object[] res =  core.dist2AvarageCore(1, new MWNumericArray(currentData, MWClassID
        MWArray ret= (MWArray)res[0];
       return  ret;
    }
```

Model also contains a number of other member functions that use the Matlab defined functions in gCore and and also a number of functions that return information on currentData and sCore:

1. getCoreVariables(): Returns a string with the names of variables in sCore
2. getSpecificCoreDataType(String fieldname): Returns the data type of a field in sCore

3. CoreVariables(): Returns sCore
4. setCurrentData(double[][]curr): Sets the currentData variable
5. getCurrendData(): Returns the currentData variable

Controller

Controller is the part responsible for the logic of the application. It decides what data model shall contain and how view shall draw the GUI. When an interaction is made, view decides how to act upon it. The most significant functions that control performs is to instruct view what bundles to be drawn and instruct model what distribution and core geometry it shall contain. To draw the bundles on the correct coordinates:

```
public void updateBundles(CorePanel corepRef) throws MWException{
      // Get IJ coordinates
          Object b = MJ.getSpecificCoredata("ij").toArray();
          double[][] ij = (double[][])b;
       //Get distribution values
          Object bund = MJ.getDistribution(axis).toArray();
          double[][] powr=(double[][])bund;

      // Draw bundles on coordinates
       for(int a=0; a<ij.length; a++){
          double[] position= ij[a];
          int i = (int)position[0];
          int j = (int)position[1];
```

The above function also decides what color the bundle shall have and attaches the value of the distribution to the bundle.

```
          double powervalue=powr[i-1][j-1];

          // Set bundle color through powervalue
          Color bundleColor;


             if (powervalue<(scalemax-scalemin)/6)
                          bundleColor = Color.blue;
          else if(powervalue<(scalemax-scalemin)/5)
                          bundleColor = Color.MAGENTA;
          else if(powervalue<(scalemax-scalemin)/4)
                          bundleColor = Color.GREEN;
          else if(powervalue<(scalemax-scalemin)/3)
                          bundleColor = Color.YELLOW;
          else if(powervalue<(scalemax-scalemin)/2)
                          bundleColor = Color.orange;
          else
                          bundleColor = Color.red;
```

The bundles are then added to a hash table with the object that view will draw in the GUI.

```
corepRef.addObjectToHash(bundle);
```

In order for the GUI to present a list of the distributions associated with the loaded file, controller reads the distribution list using the getSpecificCoreData function from model.

```
  public void setDistList() {
        Object temp = MJ.getSpecificCoredata("distlist");
        MWCellArray dlist = (MWCellArray)temp;

        int[] dim = dlist.getDimensions();
        Object[] b = dlist.toArray();

        str = new String[dim[0]];
        for(int loop=0; loop<dim[0]; loop++){
        Object[] s = (Object[]) b[loop];
        Object s2 =s[0];
        char[][] s3=(char[][])s2;
        String strtemp ="";
        for(int loop2=0; loop2<s3[0].length; loop2++){
           strtemp=strtemp+s3[0][loop2];
        }

            str[loop]=strtemp;
        }
    }
```

To instruct model what data it shall load from file, controler has a LoadFile function. In this function it tells the model instance, initiated as MJ, to use its Matlab defined functions.

```
 public void LoadFile(String filename) throws MWException{
         MJ.LoadCoreData(filename);
          Object a = MJ.getSpecificCoredata("iafull").toArray();
          double[][] s = (double[][])a;
          size = (int)s[0][0];
          Object b = MJ.getSpecificCoredata("off_set").toArray();
          double[][] off = (double[][])b;
          offset= (int)off[0][0];
          int dump =0;
          Object c = MJ.getSpecificCoredata("power").toArray();
          power = (double[][])c;
    }
```

Controller also contains the functionality for positioning the controlrods and the detectors, this is achieved using the class for retrieving specific core data, in model. When the coordinates of the controlrods and detectors are retrieved, controller adds them to the object hashtable.

```
 public void getControlRodBundles(CorePanel corepRef){

        // Get ControlRod coordinates
         Object b = MJ.getSpecificCoredata("conrodcoor").toArray();
         double[][] conrod = (double[][])b;
         String iSite="1";
         String jSite="1";

        // Get ControlRod position
         Object obj =  MJ.getSpecificCoredata("konrod").toArray();
         double[][] konrod = (double[][])obj;

         //Draw ControlRods on coordinates
         for(int a=0; a<conrod.length; a++){
            double[] position= conrod[a];
             int i = (int)position[0];
             int j = (int)position[1];

           controlrod = new ControlRod(new Point(i, j), new SiteCoordinate(iSite, jSite),

           //Set collor on controlrods grey,black
           if(konrod[0][a]<100){

               controlrod.setColor(Color.BLACK);
             }
           else{
               controlrod.setColor(Color.GRAY);
           }
           corepRef.addObjectToHash(controlrod);
         }
    }
```

View

For the implementation of CJM, View uses a number of classes from a program called S3Rview, at Studsvik Scandpower. These classes provide a basic toolbox for displaying a nuclear core together with the Java JFrame toolbox. The classes from S3Rview:

1. class Bundle: a class for drawing a bundle, with image file, bundle values.
2. class ControlRod: a class for drawing controlrods, with image file, coremap position and alignment.
3. class CorePanel: a class for drawing the core with bundles, controlrods and detectors.

The View class is an implementation of a standard Java JFrame which is a set of classes used to create graphical user interfaces. The CorePanel is an extension of a JPanel which is the foundation of the GUI. The JPanel provides a window on which menus, buttons, controlrods and corebundles etc, can be added. To add the bundle and controlrods on the JPanel, view calls upon control.

```
jCorePanel = new CorePanel(control.getPlantName(),
  new Dimension(control.getSize(),control.getSize()),
  control.GetCoreType());
           control.getCoreBundels(jCorePanel);
           control.getControlRodBundles(jCorePanel);
```

For CJM a number of menus and buttons are implemented to provide the following interactions possibilities.

1. A menu with the distributions associated with the current displayed core.
2. Buttons to change which axial level of the bundles currently displayed.
3. A Button to display the average of all the axial levels
4. A list of the state point of the current distribution

The button and menus have listeners attached to them. When an interaction is made, view calls on control with the information. The following code is an example on how view calls control when an interaction is made.

```
private void jButtonViewClicked(java.awt.event.MouseEvent evt) throws MWException
   {

        if (evt.getSource().equals(axisPlus))
        {
            control.plusAxial();
              control.updateBundles(jCorePanel);
                 this.axis.setText("Axis: "+control.getAxix());
              jCorePanel.repaint();
        }
```

3.3.3 Creating a stand alone runnable application with install file

To run CJM a computer needs a Java runtime environment and Matlab compiler runtime. Most computers have a Java Runtime Environment. The Java runtime environment is backwards compatible, which means that as long as you have a version of Java on the computer that is the same as the version used when building the end program or a newer one, the computer can run the Java program. Matlab compiler Runtime has some issues with compatibility between version. Therefore, to ensure that the computer can compile the Matlab code, it is best to have the same version of Matlab compiler Runtime as the computer on which the program has been built. The issues with compatibility of both Java and Matlab can be solved by bundling the Java and Matlab compiler Runtime with finished program. This is achieved by sending the correct version of Java and Matlab compiler Runtime together with the end program. The end program can be started using the paths of the bundled files. This will affect the start up time of the program since libraries connected with the bundled programs must be loaded before it can run. Instead of using the bundle approach, installation scripts can be written that searches the computer for versions of

Java and Matlab, ensures that the end program can run using this versions and suggest installations if needed. For CJM the correct versions are bundled together with the end program. The batch file that initiates the program first updates the temporary classpath of the computer and then starts the program. An issue with Matlab Compiler Runtime occurs when using this approach. The path to Matlab Compiler Runtime can not be dynamic. Therefore the batch file must first search for the folder of the Matlab Compiler Runtime Library and then update the library.

```
Path=%Path%;%~dp0MATLAB Compiler Runtime\v714\runtime\win64
.\java\jre6\bin\java -jar ".\coreGJavaMatlab\dist\coreGJavaMatlab.jar"
```

Through this method the current installation file of CJM can run on Windows computers without either Java or Matlab Compiler Runtime installed.

Chapter 4

Results

The purpose of this work was to investigate possible ways of achieving integration between Matlab and Java and to present a solution in the form of a proof of concept program. The result can be separated into two parts:

1. What information the investigation of ways of achieving the integration has yielded.

2. The proof of concept programs capability, in relation to the requirements analyses and the efficiency of the implementation.

The subject of integrating Matlab and Java has been investigated in this report. A number of methods are presented together with a discussion of what method to use for implementing the proof of concept program. To compare these methods there must be a detailed description of a scenario in which the integration of Matlab and Java is necessary as well as implementation of all these methods. However that comparison would be subject to the stated scenario, what method that works best for a specific scenario can vary depending on the scenario. Therefore it is not possible to directly state which method is preferable without first stating the scenario. What can be done is to weigh the strengths and the weaknesses against each other without referring to the scenario in which they will be used. The table below is a summary of the strengths and weaknesses of the discussed methods in this paper. Note that there may be other ways of achieving the integration, there is no general way of summarizing all the methods.

Table 4.1: Strengths and weaknesses of integration methods

	Integration methods			
	Matlab builder JA	Java to Matlab interface (JMI)	Calling Java from Matlab	Common database source
Implementation	Contains pre build functionality for parsing data types between Matlab and Java	Needs to implement functionality to parse data between Matlab and Java	Contains pre build functionality for parsing datatypes between Matlab and Java	Needs to implement functionality to parse data between Matlab and Java
	Program runs only on Java Session	Needs two session to run, a Java session and a Matlab session	Needs two session to run, a Java session and a Matlab session.	Needs two session to run, a Java session and a Matlab session.
		Error handling must be implemented on both sessions	Error handling must be implemented on both sessions	Error handling must be implemented on both sessions
	Needs to have Matlab builder JA installed	Needs a JMI library to run	Needs no additional software to run	Needs a database source
Distribution to end user	Can be compiled into a stand alone executable Java program	Both sessions most be compiled into standalone executables	Both sessions most be compiled into standalone executables	Both sessions most be compiled into standalone executables
Economy	Not free ware	free ware	free ware	free ware

As discussed in the section 2.2, the environment in which the implementation of the scenario is to be achieved is also important factor to be considered when deciding the method to use. The capacity of the people responsible for the implementation and the working methodology used when implementing must be considered.

The current form of CJM (see figure 4.1) can perform the following actions (as stated by the requirements analysis).

1. Read data from files of type res and cms:
   At the current form CJM can read files of types csm and res. The function getCore can be extended in order to be able to read the other file types.
2. Display the geometry of the core with controlrods and bundles
3. Display distributions read from file with a color scheme on the bundles
4. Display one axial level a time or an average of the axial levels.
5. Read a distribution selected by the user and display it.
6. The program can be installed using an install file and it does not demand an installation of other components, thus making it stand alone.

The priority of the implementation has been to create the integration between Matlab and Java necessary for the program to use functions from Cmsplot together with a Java GUI. The GUI is not designed with usability in mind thus the current form of the GUI is unimpressive. The Matlab code written for CJM often contains unnecessary left over code from Cmsplot. This is a consequence when directly using code written for Cmsplot and not adapting them for the purpose of being used in CJM. The current installation file is usable on any windows computer however the start up time for the program is perhaps longer then convenient.

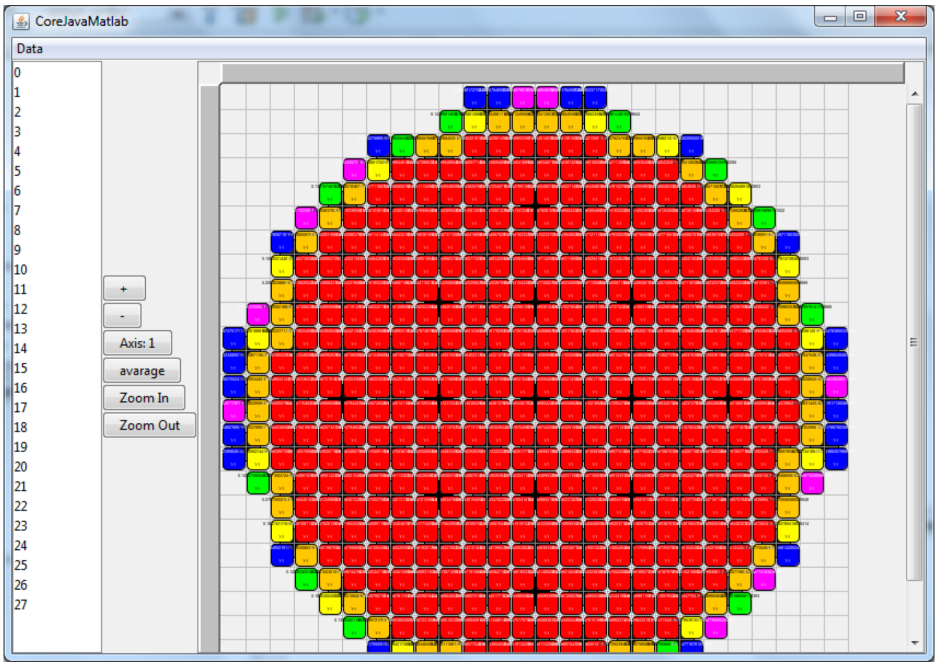


Figure 4.1: CoreJavaMatlab

Figure 4.1 displays the current form of CoreJavaMatlab.

The result of this thesis can be summarized as follow: The investigation in ways of achieving interaction has yielded a description of a number of different methods of achieving the integration. These methods has been compared however not implemented, thus the comparison lacks the information that the implementation would have yielded.
The design and implementation of the proof of concept program successfully achieved the necessary integration between Matlab and Java. The implementation was done without proper planning and testing thus it contains left over code and uses more memory than necessary. The GUI is primitive but functional. It was implemented without taking the end user into consideration.

Chapter 5

Conclusions

In this paper the subject of integrating Matlab and Java has been discussed and at some length tested. The first part of the paper presents a strictly theoretical view on the subject and the implementation part of the paper describes the work of implementing one specific method.

The intergration methods described in this paper are:

1. Java to Matlab interface (JMI): The method starts a Matlab session from Java and communicates through the standard output input stream. It has the advantage of being free-ware and is defined as a Java library, thus not demanding any additional software to implement. The method must deal with the problem of parsing data types between Matlab and Java. The two sessions must both be compiled into standalone executables in order to create a program suitable for distribution to end user.

2. Calling Java from Matlab: Matlab has built in functionality for calling Java classes. To create a Java GUI Matlab could start a Java session that displays data and receives user input. The structure would be that of a client server architecture. There are functions in Matlab that can be used to parse data between Matlab and Java. Since there are two sessions both must be compiled into standalone executables in order to create an end program suitable for distribution.

3. Common database source: The method uses a database source shared between a Matlab and a Java session. This method is similar to a JMI. The two sessions must both be able to read and write from the Database. Matlab would be responsible for manipulating the data and Java for displaying it. In order for the Matlab session to know when to make calculations there must be some type of flag in the database. Since there would be two session in the program solution both must be standalone executables in order for the program to run standalone.

4. Matlab builder JA: Matlab builder JA is a Matlab extension program that creates a Java wrapper around Matlab functions. The function can then be used like any other Java class through the use of a Matlab runtime

compiler, that compiles the Matlab functions. The add on program comes with a class that handles the parsing of data between Matlab and Java. Since there is only a Java session the end program can be made standalone by compiling the Java code into a standalone executable jar.

The integration methods presented are not the only possible ways of achieving the integration however they cover the most common methods to which there exist software tools for implementing. The part of using a common database source is not mentioned in any of the sources used for this paper however to the author it seems logical to give the data of the application a more significant part of the system design since the parsing of data is a reoccurring subject. The main obstacles of achieving the integration have been addressed for each of the methods. As mentioned in the result chapter, the shape and form of the program to be written as well as the people responsible for creating the program are also important factors.
The part of evaluating the methods lacks testing of each method and must rely upon various sources that only present one method without any comparison with other methods. The problem with testing each of the methods is that they would have to be tested on the stated scenario from Studsvik Scandpower and there was not enough time for such an attempt. Thus it is by reasoning and not testing that the conclusions of the theoretical investigation has been achieved.

When the implementation of the proof of concept program was about to begin there where a number of obstacles which proved to be time consuming. To get all software, needed for the implementation, installed correctly and working together was a tedious endeavour. Matlab has problems with the compatibility between versions and this proved to be problematic when trying to run the program on computers that had different versions of Matlab installed. It would have been much easier to use bundled versions of Java and Matlab, as used in the install file, together with the program. This is also convenient on a project with multiple programmers. If some sort of share folder is used for the project, the folder can have bundled versions of MCR and Java and started using a simple batch script. Then it is possible for Java programmers to work on the Java files of the project and make test runs without having Matlab or Matlab builder JA installed. To use Matlab builder JA to implement the integration was a valid decision. Any of the other methods would have demanded a lot more time on the implementation. The incorporation of Matlab functions into Java is achieved quickly using this method and in any of the other mentioned methods the Matlab code would most likely require extensive rewriting.

The decision to use a model view controller design(see figure 2.5) for the proof of concept program proved to be a good choice. It has the advantage of separating the program into isolated parts, which is especially suitable for a project such as this, where functions from different program languages is being used. This makes updates and change to the code more accessible and it is easier to divide the implantation to multiple programmers.

The following table is an example of how the responsibility of the Matlab and Java programmer can be divided.

Table 5.1: Responsibilities of Matlab and Java programmer

Matlab	Understand what Mathematical calculations the program needs
	Design algorithms that make the calculations
	Write Matlab functions that implement the algorithms
	Ensure that functions return data of a suitable type
	Deploy the functions using Matlab Builder JA
Java	Incorporate the deployed functions into Java
	Write Java code that accesses the deployed Matlab functions
	Build a standalone executable application

The task of integrating Matlab and Java can be addressed in several different ways. In order to make a good decision on how the integration is to be achieved, a number of methods must be considered and compared. When comparing the methods factors such as the programmers capabilities, the purpose of the program and the software available, must all be taken into consideration.

Bibliography

[1] Klimke, Andreas (2003). How to access Matlab from Java. Universität Stuttgart

[2] J. Saanchez, F. EsquembE, C. MARTI Â N, S. Dormido, S. Dordmido-Canto, R. D. Canto, R. Pastor,A.Urquiaa (2005)
"Easy Java Simulations: an Open-Source Tool to Develop Interactive Virtual Laboratories Using MATLAB/Simulink"

[3] Francisco Esquembre. Easy Java Simulations (2008)
http://www.um.es/fem/EjsWiki/

[4] Matlab Control (2010)
http://code.google.com/p/matlabcontrol/

[5] Stefan Müller, JMatLink (2005)
http://jmatlink.sourceforge.net/

[6] MathWorks, Matlab Builder JA (2010)
http://www.mathworks.com/help/toolbox/javabuilder/

[7] MathWorks, Matlab Builder JA user's guide (2010)

[8] Calling Java from Matlab
http://radio.feld.cvut.cz/matlab/techdoc/matlab_external/ch_java.html

[9] Luigi Rosa, Matlab to MySQL Interface (2007)

[10] MathWork, Matlab Compiler Runtime (2010)
http://www.mathworks.com/products/compiler/

[11] Java JFrame
URL: http://download.oracle.com/javase/1.4.2/docs/api/javax/swing/JFrame.html
All above links visited at 9/8 2011