TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

INGO FRÖHLING

DEVELOPMENT OF A FRAMEWORK FOR A JAVA-BASED SIGNAL PROCESSING E-LEARNING PLATFORM

Master's thesis

Examiners:
Prof. Irek Defee and
Prof. Markku Renfors
Examiners and topic approved by
the Faculty Council of the Faculty of
Computing and Electrical Engineering on February 5, 2014

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Degree Programme in Information Technology
FRÖHLING, INGO: Development of a Framework for a Java-Based Signal Processing E-Learning Platform
Master of Science Thesis, 67 pages, 9 Appendix pages
June 2014
Major subject: Signal processing
Examiner: Prof. Irek Defee
Keywords: e-learning platform, JavaServer Faces, software architecture, software development

The design and implementation of an interactive, but easy to extend and to maintain e-learning platform is a complex task. In order to achieve this, existing learning platforms have been reviewed concerning learning principles applied in them. Then, use cases have been defined and a prototype of a web based learning platform has been built. Out of this prototype creation process, the software architecture of the learning platform as well as a model for creating exercises have been developed.

As a result of this thesis project, a well-structured, JavaServer Faces based distributed e-learning software system has been developed. This software system integrates MATLAB® functions and provides a well-structured user interface. The user is able to configure the input signals as well as the digital signal processing algorithms freely, which gives her the possibility to study the algorithm in a way she desires. The results of the algorithms are presented using interactive charts, which can be saved to local disk for later reference. Printable question sheets are attached to the exercises, which guide the student towards to learning goals defined in advance.

By this thesis project it has been shown, how modern web technologies like JavaServer Faces, jQuery and Highcharts are used to create an e-learning platform with MATLAB® as an back end. The Model-View-Controller based software architecture of the learning platform allows to separate responsibilities and thus keeps the code understandable and clean. Due to its flexible software architecture, the learning platform can be extended by other exercises, but also by other back ends like GNU Octave.

In the future, it shall be investigated how the e-learning platform can be extended to two and three dimensional signals like images and videos. Also, an integration of simple vector- and matrix exercises is desirable.

PREFACE

This work, on which this thesis is based on, has been carried out at the Department of Electronics at Tampere University of Technology. The thesis has been individually prepared by Mr. Ingo Fröhling. Its topic *Development of a Framework for a Java-Based Signal Processing E-Learning Platform* has been suggested and the thesis has been supervised by Prof. Markku Renfors, whom I would like to warmly thank for the supervision and for arranging many practical matters especially in the beginning of the thesis project. Furthermore, I would like to warmly thank Prof. Irek Defee for his supervision, interest and valuable input in matters of learning as well as the flexible arrangements to finish the thesis in my home country.

I would also like to thank Prof. Rainer Creutzburg from Brandenburg University of Applied Sciences, who recommended me to study at Tampere University of Technology and who helped me arranging the thesis seminar presentation remotely here from Germany. For the guidance through my studies and the thesis writing process, I would like to thank the coordinators of the international study programmes in Information Technology at Tampere University of Technology, Ms. Anna-Mari Viitala and Ms. Elina Orava.

Finally, I like to warmly thank my relatives and friends, especially the members of ESN INTO and Tampereen TietoTeekkarikilta for their mental support during the thesis process. Without them, my studies would not have been half as valuable. Thus, I would like to dedicate this thesis to the international students in Tampere.

Tampere, 04.06.2014

Ingo Fröhling

ABBREVIATIONS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CSS	Cascading Style Sheets
DAO	Data Access Object
DFT	Discrete Fourier Transform
DOM	Document Object Model
DSP	Digital Signal Processing
ECTS	European Credit Transfer System
EJB	Enterprise Java Beans
EU	European Union
FFT	Fast Fourier Transform
FTP	File Transfer Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IP	Internet Protocol
Java EE	Java Enterprise Edition
JSF	JavaServer Faces
JVM	Java Virtual Machine

MCR	MATLAB® Compiler Runtime
MVC	Model-View-Controller Software Design Pattern
OSI	Open Systems Interconnection Model
PHP	Hypertext Preprocessor
PPP	Point to Point Protocol
REST	Representational State Transfer
ROI	Region of Interest
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
TUT	Tampere University of Technology
UDP	Unified Datagram Protocol
UI	User interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WWW	World Wide Web
XHTML	Extensible HyperText Markup Language
XML	Extended Markup Language

CONTENTS

1. INTRODUCTION

For some students, studying signal processing is a challenging task, because it involves understanding abstract mathematical concepts. Additionally, traditional classroom-based teaching forces students to visit those lecture and exercise sessions on a given time schedule. On the other side, students need to understand the taught knowledge from different perspectives, which includes hands-on examples and demos. This gives students 24/7 access to the learning material and allows them to plan their study week more independently. Additional material from external sources, exempli gratia, from other websites, needs to be easily accessible by the student.

For Tampere University of Technology's departments of Signal Processing and Digital Communication Systems, a prototype of a learning platform, which supplements classroom teaching is needed. It will be developed throughout this thesis work. The focus of the learning platform lies in the area of signal processing and communication theory.

Since TUT receives students from a wide variety of countries and universities, the level of subject-related and prerequisite knowledge as well as the language proficiency in English[1] varies between the students, which especially affects the introductory courses like *SGN-1107 Introductory Signal Processing* and *TLT-5206 Communication Theory*.

Challenges with teaching and their solution approaches have been addressed in many scientific articles. Based on the origin of the articles, recently industrialized and developing countries like China and India need to educate many students at low budgets.

Many approaches like [1] and [2] identify the student as the main actor of the system. These approaches are grounded on the fact, that in many universities, teaching is centered on the student.

The learning platforms, which were developed or employed in the studied articles ground on different teaching models. Problem-centered education [3] applies the method of helping the student to decompose a complex problem into smaller, solvable problems[2] by letting the students develop a solution concept. The workflow based approach [4] considers the whole process of learning, teaching and administrating the learning platform. The question and test driven learning model [5] focuses on challenging the students with questions and small exams to figure out the next useful steps to improve the knowledge of the student. Another article [6] discusses the traditional task-driven model. The conclusion, however, is, that this model does not motivate students to learn actively. Also the

[1] English is the language of instruction of the international Master's degree programmes at TUT

[2] This is referred as Divide and conquer

idea of cooperative learning [6] is addressed, which employs the positive effects of team work when solving exercise tasks.

Through the years, Moodle [7], a modular general-purpose learning platform has been developed and is still undergoing further development. It consists of useful features, which enhance the learning outcomes of the students. These are: chat module, forum, self-tests for students, different languages for the user interface, et cetera. It also allows to assign roles to the user like student, teacher, administrator.

Moodle and many other learning platforms developed by universities come with management tools. These allow teachers to create content, compile courses out of that content, create examinations, and observe the student's learning progress. The latter feature allows the teacher to detect weaknesses and react in time by exempli gratia arranging tutoring sessions to improve the student's learning outcomes further. This helps the student to pass the course using less attempts. Moodle's feedback module allows the students to comment on the course and suggest improvements for the next implementation of the course. In [6] an approach of designing an online learning platform using Moodle is discussed.

It is mentioned in [3], that in order to facilitate the student's way of learning, the material needs to be presented from different points of view. This is done using different contexts [3].

With the introduction of Web 2.0 [4] and HTML5, rich and easy-to-use user interfaces using the Browser-Server architecture [2, 5, 1] can be built. Already existing back-end technologies like Java EE, PHP, MySQL [5] and Microsoft's ASP are used to fetch, filter and prepare content for displaying it in the user interface. Integrating social features like forums and chats allows the students to exchange knowledge and help each other understanding difficult topics. Before the era of HTML5, proprietary platforms like Flash [8] or Microsoft .NET [9] were widely used for web-based learning platforms. These platforms, however, introduce additional costs in form of royalty fees for licensing.

Another form of learning platforms are remote laboratories [10], where students are able to conduct experiments remotely, id est from their home PCs or mobile phones. As mentioned earlier, especially in industrializing and developing countries, but also at universities in industrialized countries, education has to be cost-efficient. Learning platforms allow to cut costs by giving courses entirely online by reducing the amount of physical exercise and tutoring sessions. Such remote laboratories allow to cut costs by giving the students the possibility to study on themselves rather than being supervised by a teacher.

When choosing or developing a suitable platform, license fees have to be considered. Expensive, ready-made platforms work out-of-the-box and are usually maintained by external companies. Open-source platforms,on the other hand, come with virtually no costs, except for the hardware, but require maintenance. However, features are added and bugs are fixed usually by a community of professional and hobby software developers. Plat-

[3] theoretical mathematical context, practical application context, demo

[4] Web 2.0 introduced technical improvements and social networks

forms, which are entirely implemented by the teaching institution, are tailored directly to the institution's needs, meaning that unnecessary features are dropped. However, they come at the cost of entirely implementing and maintaining the system by the teaching institution. Powerful open-source software or freeware like Java EE, MySQL, PHP and JavaScript libraries [5, 7, 6] still allow for a rapid development of useful learning platform software. Additionally, since these technologies are well-known to programmers, it is easy to find skilled personnel to create a learning platform based on those open-source software packages.

Cloud storage [11] allows teaching institutions to exchange teaching material or to design common examinations. It is even possible to apply existing cloud applications like Google Docs, Google Calendar, et cetera. Since the cloud environment is a virtual environment, storage and application are scalable to fit the needs of the educational institution. However, this comes with the penalty of reliability issues, because cloud providers have the possibility to close their services or delete user accounts. Thus, the same data needs to be stored on different cloud providers. A very promising example for a cooperation between different universities of technology on US-wide and even international scale using cloud service are edX [12] and Coursera [13]. Coursera offers distance learning courses, of which students can also take examinations and receive a certificate of successful completion of the course. The edX service also provides an excellent example of reusing lecture material, which has been generated and updated by one institution, for giving courses at other universities.

The interesting problem of creating a meaningful data model for the learning platforms has been discussed in some articles. It is suggested by [14], that learning material is decomposed into atomic pieces, which can be easily reused by other lectures or courses. Such atomic pieces are images, graphics, small chunks of texts, videos, et cetera. Those are grouped into concept maps[5] out of which lectures and courses are assembled. It is natural at this point, that such an approach minimizes time needed for authoring and reduces the amount of disk space needed to store the material. Such systems for authoring and composing courses are referred as course management systems. Besides the above mentioned features, course management systems allow the management of access rights for users.

For digital signal processing students, lecture material is presented using multimedia objects[6], which are presented by the user's browser to allow platform-independency. Multimedia objects,and the non-linearity of hypertext documents[7] and demos allow to exploit the human associative way of thinking and makes accessing of external resources very easy and natural [8]. When talking of platform-independent multimedia objects, it is nat-

[5]Concept maps allow to describe how a complex problem or course can decomposed into smaller, understandable and reusable pieces

[6]The term Multimedia objects refers to animations, videos and images

[7]The non-linearity of hypertext is generated using hyperlinks to other learning units in the learning platform or even to external resources

ural to also name mobile phones as means of access [15, 14], because nowadays the vast majority of mobile phones are able to display images, hypertext, play audio and video. This allows the student to study even when traveling or on vacation, where ordinary PCs are not available or not desired due to their extra weight and packing space usage.

A challenge in the context of hypertext document using mathematical formulas [8] is the problem of typesetting them. This has been resolved with the introduction of the MathML markup language [16] and the ability to transform formulas into images. The MathJAX JavaScript library [17] automatically chooses the best suitable option for embedding formulas into a web page.

As an example for teaching signal processing, a learning platform for the Fast Fourier transform is discussed in [18]. It is a typical example of a mathematical, abstract concept, which is yet one of the main fundamental concepts in signal processing and communication. The learning platform discussed in that article visualizes the concept and enables the student to set input signals and additional parameters and to directly see how the inputs influence the output of the Fast Fourier transform. Also [19] discussed a learning platform for improving the understanding of the fundamental concepts of digital signal processing.

Some articles suggest to assess the learning progress of the students using quizzes, intermediate online exams and feedback forms [3, 6]. Impact of a recently introduced learning platform is measured by comparing collected feedback from a course implementation without using the learning platform with an implementation, that uses the learning platform. Research suggests, that learning platforms increase the motivation of the students to learn, which positively influences the grades and the percentage of students, who pass the course [2]. Regular checks of the students performance not only help to figure out the need for extra exercise or tutoring sessions, but also allow to adjust the learning goals of the course [6].

A possible choice for implementing the learning platform is MATLAB® Builder JA [20]. A possible implementation of such a learning platform for signal processing is described in [21].

Another interesting kind of learning platforms are ubiquitous learning platforms [22, 23]. Those platforms allow implementing real-time help-seeking features. This is useful when the student performs a laboratory exercise and needs help about an item. The items in the questions are equipped with QR codes, which are scanned by the student's phone. The learning platform is then able to immediately give hints to the student about the item in order to help her to successfully complete the laboratory exercise.

2. BACKGROUND

This chapter introduces all concepts necessary to understand the thesis work and its findings. First, the concepts of the WWW and the client side languages HTML, CSS and JavaScript are introduced. This is followed by a brief discussion of the Model-View-Controller software pattern. Thereafter, the MATLAB® and GNU Octave numerical computation systems are introduced. The chapter is concluded by an introduction of Java Enterprise Edition.

2.1. The World Wide Web

This section discusses several aspects of the World Wide Web (WWW) relevant to this thesis work. This includes the network topology, the HTTP protocol as well as relevant languages used to crated web pages.

2.1.1. The Networking Stack

Every unit participating in a computer network implements a network stack according to the Open Systems Interconnection Reference Model[1]. Cisco has described the OSI stack comprehensively in [24]. Such a stack yields a standardized programmers interface, which provides system-independence and abstraction.

In what follows, the OSI stack is introduced in a bottom-up manner. Used protocols and possible programmers interfaces are discussed. The OSI stack consists of seven layers. Each layer allows to abstract from the layers below them.

The lowest layer in the OSI stack is the physical layer. It defines the devices, signals and properties on the wire. This is the place, where the actual transmission of data occurs.

On the top of the physical layer, the data link layer is situated. It defines the transmission protocols, which are used on the physical medium. However, it abstracts from the physical layer in that sense, that in here properties, signals and devices form the basis of this layer, but are not subject to it. Protocols used in this layer are exempli gratia Point-to-Point protocol (PPP), Ethernet and Wireless LAN protocols. These protocols only define the communication link between two directly linked devices, but do not allow for communication and addressing over intermediate nodes. An example for this layer is the communication between an Ethernet card and an Ethernet router.

[1] This is referred to as the OSI stack

In order to build up the Internet, communication must be possible over several nodes. The network layer is responsible to carry out addressing and the communication over several nodes in the network. As such, it abstracts from the data link layer by not being involved in direct link communication. The best known protocol in the layer is the Internet Protocol (IP). Its addressing mechanism allows to communicate with other devices, which are not reachable over a direct communication link. An example for this is communication from a laptop computer in the USA with the server *www.tut.fi*. The link between those two devices is established through routers, which connect the different networks with each other. Usually, the communication goes through multiple networks of which both devices at the termini do not know. Yet, it must be taken into account, that communication over the network layer is unreliable.

To overcome the unreliability of the network layer and to allow to address services running on a device, the transport layer has been introduced. It defines protocols to handle packet losses, flow control and packet segmentation or desegmentation. Packet segmentation and desegmentation allows to adapt the size of the packets to be suitable being sent over the networks in question.

The two protocols mainly employed in this layer are Transmission Control Protocol (TCP) and Unified Datagram Protocol (UDP). TCP is the more reliable protocol as it is able to detect packet losses and has the ability to request retransmission of the lost packets. It also adds an error-checking mechanism, which is able to mark erroneous packets as lost and request retransmission in the case of error as well. TCP provides abstraction for the upper layers by accepting messages of arbitrary length, which allows applications on the higher layers to abstract entirely from the underlying network, its properties and restrictions. TCP is used for the World Wide Web as it provides reliability and thus allows for a smooth data transfer from the web browsers point of view. A more lightweight alternative to TCP is UDP. By being more lightweight, it however does not provide a reliable link between two applications anymore. It is mainly used for streaming services, where packet losses are not fatal the content transmitted.

Both, the TCP and UDP protocols, introduce the concept of ports, which allow multiple applications to act on the same machine. As an example, a computer, which serves as a web server, but also as a file server is mentioned here. Also each client uses a separate ports for communication. Exempli gratia, a web browser uses port X, whereas a Secure Shell (SSH) client uses port Y for this communication.

The session layer is responsible for establishing and maintaining a connection between devices. Example applications of this layer are remote procedure calls. This layer provides a common way of interpreting messages between the two applications involved in the communication. A popular example is encryption and compression, which are both employed in the communication between two WWW terminals. Encryption secures the connection between the browser and the bank or the online store for making transfers and purchases securely and provides authentication of both terminals if needed.

Finally, the application layer defines the high-level communication protocols between two devices or terminals. Examples for this are the Hypertext Transfer Protocol (HTTP) and the File Transfer Protocol (FTP). The application layer is the layer closest to the end-user application in the OSI stack. For the WWW, the HTTP protocol is used to transfer web pages, images, style sheets, JavaScript files, but it is generally suited to transfer any content on top of it.

2.1.2. Architecture of the World Wide Web

The WWW is based on a client-server architecture, where the browser acts as the client and the web server as the server. To be more precise, the WWW follows the principle of information request and retrieval. This means, that the client as the active part, initiates a connection to a server of its choice and requests information from it. The server then replies with that information or in case of an error with an error message.

Suppose, the browser is asked by the end-user to display the weather forecast for Tampere from the national Finnish weather service. The browser opens a TCP[2] connection to the web server of the national Finnish weather service and asks it to deliver back the weather information for Tampere. The server processes the request and replies with a document, that includes the requested weather forecast.

In order to aid the browser understanding which resource the end-user wants to access, URLs have been introduced. URLs are in the form of <protocol>://<server>:[<port>] /<path>/<to>/<resource>. The protocol parameter defines which application-level protocol is used for retrieving the resource. For the WWW, http or https[3] are used. The server parameter defines the server name or IP address, whereas the optional port parameter defines the TCP port to be used for the connection. After it, the Unified Resource Identifier (URI), which defines the resource on the server, is appended.

For the weather forecast retrieval example, the URL is *http://ilmatieteenlaitos.fi/saa /Tampere*. This means, that the resource */saa/Tampere* is retrieved using the HTTP protocol from the server at *ilmatieteenlaitos.fi*. As mentioned above, the Hypertext Transfer Protocol (HTTP) is used to transfer information through the WWW and thus forms its fundamental base. The HTTP protocol is defined in a Request for Comments (RFC) at the IETF in document [25]. The RFC defines syntax, methods and status codes used in HTTP based communication. This remainder of this subsection only briefly describes the main aspects of HTTP. It is not meant as a reference to this protocol.

The HTTP Request Since the WWW is a request-response-based system, communication is always initiated upon a request sent by the browser. The HTTP request is composed of multiple rows separated by \r\n marks. The request consists of a header,

[2]TCP is a network protocol in the transport layer of the OSI stack as described in section 2.1.1.
[3]The https prefix defines, that SSL/TLS secured HTTP is used to load the page

which defines the resource and gives the server additional metadata needed to process the request. Optionally, a request body is also present if the browser wishes to send more information to the server like a submitted form. The body separated from the header introducing another \r\n line break. The syntax of HTTP requests is:

```
<method> <URI> <protocol version>\r\n
<header1>\r\n
...
<headerN>\r\n
\r\n
[<request body>\r\n
\r\n]
```

The methods specified in HTTP 1.1 are GET, POST, PUT, DELETE, OPTIONS. The GET method is used to simply request information without expecting to change the state of the data on the server and without the need to transfer a form. The POST method, on the other side, is used if data is submitted to a server and the state of its data is allowed to be changed. The GET and POST methods are used in the communication between browser and web server, whereas the remaining methods are used in web services interfaced by SOAP or REST[4] APIs. A comprehensive book [26] about RESTful APIs is available.

In the example case of the weather forecast retrieval for Tampere, the above mentioned URL yields the following HTTP request:

```
GET /saa/Tampere HTTP/1.1\r\n
Host: ilmatieteenlaitos.fi\r\n
\r\n
```

The HTTP Response After the web server has processed the request, it sends a HTTP reply. As the request, the response also consists of header and body. The basic structure of the reply is:

```
<Protocol> <status code> <status message>\r\n
<header1>\r\n
...
<headerN>\r\n
\r\n
[<response body>\r\n
\r\n]
```

The first line indicates the status of the request and the protocol, to which the message complies to. It is followed by the headers and the message body. In the case of the weather forecast retrieval to received message is shown in figure 2.1.

From this example it can be seen, that through the header a lot of meta data has been provided besides the status code. After the header has ended, the body is transferred. The

[4]Representational state transfer

```
Status Code: 200 OK\r\n
Cache-Control: public, max-age=60\r\n
Connection: keep-alive\r\n
Content-Encoding: gzip\r\n
Content-Length: 8461\r\n
Content-Type: text/html;charset=UTF-8\r\n
Date: Sat, 09 Nov 2013 12:18:00 GMT\r\n
Expires: Sat, 09 Nov 2013 12:19:01 GMT\r\n
Server: Apache\r\n
Set-Cookie: JSESSIONID=3CE3941E0862671ED5EA0F58EFA3372E; Path=/
   COOKIE_SUPPORT=true; Expires=Sun, 09-Nov-2014 12:18:01 GMT;
   Path=/ GUEST_LANGUAGE_ID=fi_FI; Expires=Sun, 09-Nov-2014
   12:18:01 GMT; Path=/\r\n
Vary: Accept-Encoding,User-Agent\r\n
\r\n
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "
   http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">\r\n
<html dir="ltr" xmlns="http://www.w3.org/1999/xhtml" class="html
   ">\r\n
<head>\r\n
<title>Tampere - Paikallissää - Ilmatieteen laitos</title>\r\n
...
```

Figure 2.1. *A HTTP response*

body has been cut after the fourth line as it contains the complete markup of the web page, which is several kilobytes long.

HTTP status codes To indicate the status of the request, the server sends a HTTP status code alongside the response. These status codes are divided into five categories. The codes between 100-199 indicate, that the request is still ongoing and that the browser is asked to perform an action in order to have the request succeeding. Codes between 200 and 299 are used to indicate successful request processing. For example, the code 200 refers to a successfully completed request with the entire results being supplied in the body part of the response message. Status codes between 300 and 399 inform the browser about a redirection to another URL, which it has to follow in order to retrieve the response. 400-499 status codes inform the browser about a problem on the client side. The codes in this category are:

- 400: Bad Request - This status code is returned if the request of the browser is malformed, id est does not conform to the HTTP syntax.
- 401: Authentication required is issued if the browser tries to access a resource, for which credentials must be supplied. The browser shall request those from the end-user.

- 403: Forbidden - This status code is issued, if the browser is not authorized to access the resource in question even if credentials were supplied.
- 404: Not Found is issued if the requested resource does not exist on the server.

Finally, status codes between 500 and 599 indicate a problem on the server. These are exempli gratia server-side runtime errors of web applications, network problems within the server's network or a misconfigured server.

2.2. Languages for the World Wide Web

In order to facilitate displaying rich content through the web well-formatted and to add interactivity to the content, several languages have been developed. The most important languages like HTML [27], CSS [28, 29] and JavaScript [30] as well as the mechanisms on how they interact are introduced in this section.

Web browsers internally store the web pages in the DOM[5]. The DOM is a representation of the web page's markup and thus its content as a tree structure. It does not only serve as a data model used to display the content on the screen, but also offers an interface for scripting languages to add, modify or remove elements. Since the DOM also stores the attributes given in the markup[6], it is possible to programmatically change the look and feel of the elements.

2.2.1. HTML and CSS

In the majority of cases, rich content in web pages is defined as HTML or XHTML markup. Both are based on the XML language, which uses tags and attributes. The only difference between these two is, that XHTML requires the markup to be well-formed according to XML. In what follows, only HTML is considered, but the knowledge is transferable to XHTML as well. A very simple example page markup is shown in figure 2.1.

In this example, first the type of the document is indicated. This defines, whether a XHTML or HTML document has to be expected by the parser and it also defines the version of markup standard[7].

As with HTTP messages, HTML documents consist of header and body. These are indicated by the head and body tags. Within the page header, the page title and any additional JavaScript[8] and CSS style sheet files to be included are defined.

[5] Document Object Model

[6] Those attributes contain style information, additional information used by JavaScript framework like jQuery or Knockout.js

[7] Currently, HTML 4.01 and 5 and XHTML 1.0 and 1.1 are is use. However, older version of HTML markup is still present on legacy pages.

[8] Technically, it is also possible to insert the JavaScript tags anywhere in the markup. The JavaScript code is loaded and inserted at the place of the script tag.

```
<!doctype html>
<html>
<head>
    <title>Page title</title>
</head>
<body>
    <p style="font-family: Arial,Helvetica;">This is a text
       paragraph typesetted using the Arial font.</p>
</body>
</html>
```

Figure 2.1. *HTML markup of a simple web page*

In what follows, the most important HTML markup tags are introduced. The <*body*> section contains the actual page content. The <*p*> tag defines an ordinary text paragraph. The text is automatically wrapped at the end of the line. A line break is forced with the <*br*> tag. Links to other documents or resources are inserted using the <*a*> tag. It takes the URL as the *href* parameter and the optional alternative text as the *alt* parameter. Items on a web page are further groupable using a <*div*> container element. These are furthermore used to style the child elements using CSS. The <*span*> tag serves the same purpose as the <*div*> tag. The only difference is, that it does not force a newline on the beginning and end of it. Unordered lists are inserted into the document using the <*ul*> tag, whereas ordered lists are inserted using <*ol*> tags. Elements of such lists are defined using the <*li*> tag.

The HTML markup language only serves to define the elements to be shown in the web page. Their look and feel, however, is defined using CSS style sheets. CSS code is defined either inline directly using the element's style attribute or external using a file with *.css* extension. If a CSS file is used, upon page rendering, elements in the DOM tree are matched to the style definitions. If a match has occurred, the style definitions are applied.

CSS stands for Cascading Style Sheets, which means, that style definitions of a parent element in the DOM tree are applied first to itself and to all of its children. If for the child element another definition matches, its attributes are applied to the child element. This means, that the rough definition of the parent elements are overridden by their children.

Suppose, the following HTML markup fragment has been loaded into the DOM:

```
<div id="myContainer">
    <p id="myParagraph">This is some text.</p>
    <p id="anotherParagraph">This is some other text.</p>
</div>
```

and the following CSS style sheet fragment has been defined:

```
#myContainer {color: red;}
#myParagraph {color: blue;}
```

This typesets the first line in blue color, whereas the second line stays red.

In order to be able to choose, which DOM elements are affected by the style definition, selectors have been introduced to CSS. Some are briefly introduced in this paragraph. For a complete reference, to [29] and for the standard definition to [28] is referred. DOM elements are selected by their unique ID using the # symbol. Thus, the selectors shown in the example above only select the DOM elements having *myContainer* and *myParagraph* as their ID, respectively. HTML markup allows to add class attributes to DOM elements. This is useful, if a style definition shall be applied to a set of DOM elements. In the style definition, the . symbol is prepended to the classes name. It is also possible to use a tag name as a selector. This matches all tags with that name and applies the associated style definition to them. Descendants, children and siblings are selected using combinators.

2.2.2. JavaScript

JavaScript is a scripting language, whose purpose in the browser context is to create or modify page content dynamically at runtime. As mentioned in the introduction of this section, the browsers JavaScript DOM APIs have the ability to directly access, add, delete and modify elements and their attributes in the browser's DOM. This makes JavaScript together with HTML and CSS a very powerful tool to create visually and functionally appealing web pages, which are able to act like an ordinary desktop application. JavaScript code is usually stored in external *.js* files, from where it is included using:

```
<script src="http://server.ip/path/to/the/JavaScript.js">
</script>
```

It is also possible to directly insert JavaScript code into the HTML file using the script tag:

```
<script>
alert('This is a JavaScript code snippet');
</script>
```

JavaScript is an object-oriented language, whose syntax is similar to C/C++'s syntax. However, it does not come with a strong-typed system. This means, that classes cannot be defined, but objects having attributes and methods are possible to realize in JavaScript. In what follows, the principles of the inter-operation between JavaScript and the DOM is introduced. A full JavaScript and DOM reference is available at [30].

JavaScript and the DOM The DOM API of the browser introduces functions to retrieve, add, delete and modify DOM elements and their attributes. This paragraph gives a brief introduction into these functions.

A new DOM element is created using *var element = document.createElement(type)*. This just creates a new element but does not add it to the DOM. Since it is not a child of any child of the document object, it does not appear on the web page.

Assigning a DOM element as a child of another element is done using *anotherElement.appendChild(child)*. If *anotherElement* is reachable through the DOM tree from the document node[9], after the function call its just assigned child node is and thus will be displayed on the web page.

DOM elements are retrieved by invoking *document.getElementById(domId)*, *document.getElementByName(domName)* or *document.getElementByTagName(tagName)*. The first call queries the DOM for the element having the unique ID *domId*, whereas the second call returns the DOM element having *domName* as its name attribute. The latter function returns all DOM elements having the HTML tag *tagName*. It also possible to dynamically remove elements from the DOM tree. This is achieved by using *parentElement.removeChild(childElement)*.

Suppose as an example a <*div*> container element created by the following HTML markup:

```
<div id="container"></div>
```

This <*div*> element with the ID container is empty. As such it does not display anything on the web page. A JavaScript code snippet, which adds content and modifies attributes of the <*div*> element is shown in what follows:

```
var containerElement = document.getElementById('container');
var textParagraph = document.createElement('p');
var textNode = document.createTextNode('Hello World');
textParagraph.appendChild(textNode);
containerElement.appendChild(textParagraph);
containerElement.style.color = blue;
```

This JavaScript code snippet first retrieves a reference to the div element. Then, it adds a new paragraph and a new text node into the DOM. Now, the pieces are wired up by first assigning the text node as a child to the paragraph node. After that, the paragraph node itself is assigned as a child element to the <*div*> element. Finally, the <*div*> element and all of its children are styled using CSS[10] with blue as the foreground color. Expressed in HTML markup, this yields the following:

```
<div id="container" style="color: blue;">
    <p>Hello World</p>
</div>
```

This example demonstrates the power of dynamically manipulating the DOM using JavaScript. Out of this feature many JavaScript libraries and framework have been developed. Some of these are discussed in what follows.

Asynchronous JavaScript and XML - AJAX Another very powerful feature of JavaScript is AJAX. It is used to request data from a web server without the need to reload

[9] The document node is the root node of the DOM tree

[10] see section 2.2.1.

the entire web page. AJAX requests are thus used if only a DOM element is modified by the results of the request, but not the entire web page. It is obvious, that this increases interactivity and responsiveness of the web page and it reduces the load on the network. Since the request is sent asynchronously, the web page does not block or freeze during its execution. Tutorials and references of AJAX can be found at [31] and [32].

AJAX requests are managed in the browser by a *XmlHttpRequest* object. On the server side, these are treated as ordinary HTTP requests such as issued by clicking a hyperlink or submitting a form.

To issue an AJAX request, first a *XmlHttpRequest* object is created and initialized with the HTTP method to be used and the URL. Then, the request is sent to the server, which processes the request and sends a reply according to the HTTP protocol as described in section 2.1.2. After the response has been received[11], it is processed by JavaScript and finally inserted into a DOM element or the contents of a DOM element is modified.

Suppose, there is a need to compute the number of samples based on the signal's minimum and maximum values on the time axis and the sampling frequency. These three inputs are queried from the user using three input fields and a text field is reserved for holding the computed signal length in samples.

The HTML markup snippet for such a case is introduced in what follows:

```
Minimum x value: <input type="text" id="minimumX"><br>
Maximum x value: <input type="text" id="maximumX"><br>
Sampling frequency: <input type="text" id="fs"><br>
<span id="signalLength"></span> samples
```

The JavaScript code, which processes the inputs, issues the AJAX request and fills in the received value into the *signalLength* <*span*> is shown in figure A.3.

As it can be seen from that example, only the *signalLength* <*span*> is modified by the JavaScript snippet issuing the AJAX request, while the rest of the page is unchanged. The amount of data transferred is significantly smaller than even transferring the small snippet of HTML markup back to the browser.

2.2.3. JavaScript Libraries

Over the time, JavaScript libraries and frameworks have been developed. In this subsection, libraries and frameworks relevant for this thesis work are introduced.

jQuery jQuery is a JavaScript library used to access and manipulate DOM elements as well as issuing AJAX requests in a much easier fashion than using the browser's DOM API described above. This is achieved by adding an abstraction layer to the browser's DOM API. Since browsers still have slightly different DOM API functionalities, jQuery

[11] A completed AJAX request is reported by the *XmlHttpRequest* object having its *readystate* attribute set to 4.

```
var minimumX = $('#minimumX').value;
var maximumX = $('#maximumX').value;
var fs = $('#fs').value;

$.ajax({
    url: '/signallength?minimumX='+minimumX+'&maximumX='+
       maximumX+'&fs='+fs,
    method: GET,
    success: function(result) {
        $('#signallength').html(result);
    }
});
```

Figure 2.2. *jQuery based JavaScript code of an AJAX request*

aids in overcoming cross-browser compatibility challenges when it comes to manipulating the DOM tree. jQuery is extendable using plugins. The jQuery web page [33] is a rich resource on API documentation and tutorials. Another source for documentation on jQuery is [34].

Elements are queried from the DOM tree using selectors. These selectors follow the same principles as the CSS selectors described in section 2.2.1. In what follows, the DOM manipulation example from section 2.2.2. is transformed to jQuery.

```
var newParagraph=$('<p>', {text: 'Hello World'});
$('#container').append(newParagraph);
$('#container').css('color', 'blue');
```

The first line of this example creates a new paragraph node with the text 'Hello World'. It is then appended to the *container* div. Finally, the blue foreground color is applied to the div and all of its children.

As mentioned above, jQuery ships with AJAX functionality. The AJAX example from section 2.2.2. can be rewritten as shown by figure 2.2. Compared to the plain JavaScript example in figure A.3., the code is significantly cleaner and easier to understand.

Flotcharts In order to interactively visualize data in the browser, JavaScript based chart libraries have been developed. Most of them are jQuery based.

The Flotcharts library [35] is able to generate a huge number of different chart types and is further extensible using plugins. Plugins allow to extend the basic functionality of Flotcharts by introducing new chart types, add labeling and drawing options, modify grids and axes and save the generated chart as an image to the client computer's disk. An example of a chart generated by Flotcharts is shown in figure A.10. The same chart zoomed along the x axis is shown in figure A.11.[12]

[12] Own experiments have shown, that saving charts as images to disk does not work reliability in all web browsers.

Flotcharts uses references to DOM elements to determine into which container element in the DOM tree the chart has to be placed. This allows the user of the library to get started very quickly. This is demonstrated using time series depicting the average temperatures of Tampere [36] throughout the year, which is plotted into a line chart. First, the container element is defined in the web page's HTML markup:

```
<div id="chartContainer"></div>
```

Then the data series is defined and the chart object is inserted into this container using JavaScript:

```
var dataSeries = [ [
[1, -6.4], [ 2, -6.9], [ 3, -2.8], [ 4,  3.3],
[5,  9.7], [ 6, 14.1], [ 7, 16.9], [ 8, 15.0],
[9, 10.0], [10,  4.6], [11, -0.6], [12, -4.5]
] ];
var plot = $('#chartContainer').plot(dataSeries);
```

This code is sufficient to produce a basic line plot showing such a time series. This example has shown the abstraction power from the DOM and any low-level line drawing algorithmic challenges, of which jQuery and Flotcharts are capable of.

Highcharts Another JavaScript chart library is Highcharts [37]. An example of a chart generated by the Highcharts library is shown in figure A.13. As with Flotcharts, a Highcharts chart is inserted into a DOM element. Data series are defined in a one-dimensional JavaScript array through the option array. The above mentioned example transformed to Highcharts is shown in figure 2.3. The *highchartsOptions* object in figure 2.3. controls the whole chart and thus reduces the need to programmatically call functions on the *highcharts* object. The Highcharts API [38] extensively specifies the attributes of such an object. Figure 2.3. shows the previous example transformed for the Highcharts library.

2.3. The Model-View-Controller Software Pattern

Model-View-Controller (MVC) is a software pattern, which is widely used in graphical user interface (GUI) programming. Its purpose is to separate application logic from the data model and the view [39]. Such a separation allows the graphical page design to be carried out by user interface (UI) designers, whereas technical aspects and program logic are implemented by software developers.

MVC constitutes of a model, one to many views and one to many controllers associated to it. However, the model does not have any knowledge of its associated views or controllers [40]. The model is regarded as the state of the application as it holds all the required data needed to represent that state [39]. Since the model has no knowledge about the other components of the software, there is a need for a mechanism, which is capable

```
var highchartsOptions = {
    series: [
        {
            type: "line",
            name: "Temperature [C]",
            pointInterval: 1,
            pointStart: 1,
            data: [ -6.4, -6.9, -2.8, 3.3,  9.7, 14.1,
                    16.9, 15.0, 10.0, 4.6, -0.6, -4.5]
        }
    ]
};
$("#chartContainer").highcharts(highchartsOptions);
```

Figure 2.3. *JavaScript code for producing a Highcharts chart showing the mean temperatures in Tampere*

of updating the model and notifying the view of the updated model. This is achieved by the controllers.

Controllers are responsible for reacting to and retrieving user input, processing it, triggering all the required action to other components of the software system and filling the model. At the process, the controller chooses and notifies the correct view about the updated data [41].

Finally, the views are used to present the current state of the application partly or completely to the user. The data is queried by the model and simply inserted into the places, where it needs to be displayed. Views are divided into sub-views allowing for logical separation and reuse of those sub-views [41].

As mentioned earlier, there is the possibility of having more than one view and controller sharing a model. Such a situation arises from different views, that are needed for different user groups, output devices, screen resolutions et cetera.

In the remainder of this section, aspects of employing the MVC pattern into web applications are discussed. Generally, the use of MVC in web applications is possible. However, during the design phase, the application has to be partitioned properly in order to scale properly for the intended end-user devices and expected network traffic of the application.

One possibility is to have the whole logic running entirely in the web browser[13]. The view and the controller have be downloaded only on the first request as the former gets updated by the controller and the latter does not change on the client itself. On every update, however, the data model has to be fetched from the server. This choice is clearly useful if the data model is small and the operations are not computationally demanding and do not consume huge amounts of the web browser's memory. The advantages of such a partition is to have very short to almost instant response times.

[13]This is referred to as Fat Client

Another partition possibility is to deploy the whole application logic including the data model on the server and also generate the markup of the view there[14]. This avoids downloading the data model and the application logic to the web browser. It also reduces the computational power and memory needed on the web browser to run the web application. However, the entire view markup has to be transferred to the web browser even when just a very tiny piece of the view has changed.

Since the introduction of AJAX[15], web applications can be partitioned in an arbitrary manner. This allows parts of the applications, like text labels, to be updated from the server without the need to transfer the whole page. This makes the page more responsive and reduces network traffic.

2.4. Numerical Computing Systems

MATLAB® Builder JA [42, 20] is part of the MATLAB® software package, but not included in the standard distribution. It is sold with MATLAB® Compiler, on which MATLAB® Builder JA is based.

MATLAB® Builder JA takes one or more MATLAB® functions as input and generates a Java jar package out of them. During the compilation process, each MATLAB® function is embedded into a Java class, which is then packaged into a jar file for easy deployment. The parameters for the compilation are set and the compilation itself is triggered comfortably using the command *deploytool* in MATLAB® Desktop.

After the compilation succeeded, the jar file is copied to its intended place[16] and is used there as an ordinary Java library. For the jarred MATLAB® component, in order to be able to run, it needs a special runtime environment, referred to as *MATLAB® Compiler Runtime (MCR)*[17]. The MCR includes the MATLAB® interpreter, which enables the jarred MATLAB® function to execute and produce results as it would run inside the MATLAB® version. When downloading and installing the MCR, it must be ensured, that the version of the MCR matches the version of MATLAB® desktop, which was used to create the jarred component.

The WebFigure feature of MATLAB® Builder JA allows to produce figures using MATLAB® code, which are embedded into a web page. The figures are zoom-able and pan-able. The figures are embedded into a web page by using a JSP tag library or by retrieving HTML markup and embedding it into a web page. On the MATLAB® side, the figures are created as any figure except that they are hidden. When the deployed MATLAB® function returns, the WebFigures are transformed to a *WebFigure* object and returned to the calling method. Technically, a MATLAB® WebFigure runs in an ex-

[14] This is referred to as Thin Client

[15] Asynchronous JavaScript and XML, see section 2.2.2.

[16] This process is referred to as deployment.

[17] MCR can be downloaded free-of-charge from `http://www.mathworks.com/products/compiler/mcr/`

tra servlet provided by MATLAB® Builder JA. An example of a figure generated by MATLAB® WebFigures is shown in figure A.8.. A zoomed version of the same figure is shown by figure A.9..

As an alternative to MATLAB®, GNU Octave [43] is considered. It serves the same purpose and follows the basic syntactical principles as MATLAB® does. Additionally, GNU Octave is developed with good compatibility to MATLAB® in mind. However, full compatibility[18] is currently not achieved.

In contrast to MATLAB®, GNU Octave is GPL[19] licensed, which allows to freely download and install the software. As an important feature of the GPL, the source code can be examined and modified, which allows to fully prove the correctness of the calculations done by GNU Octave.

As with MATLAB®, GNU Octave is also interfaceable from the Java programming language. The interface library is JavaOctave [44]. JavaOctave works different compared to MATLAB® Builder JA. While MATLAB® Builder JA compiles all MATLAB® code into Java jar packages, JavaOctave simply fires up the Octave engine. To this Octave engine, matrices, scalars and vectors and GNU Octave code are fed in. The code is then executed and the results are fetched from the Octave engine and are converted to Java objects. The main limitation of JavaOctave is the missing support for struct vectors and matrices. Simple 1x1 structs are, however, supported. Thus, workarounds, like wrapper functions, must be employed to convert those struct matrices and vectors to 1x1 structs. Another limitation is the missing support for retrieving vectors and matrices created by vectorized code like

```
x=0:0.1:10;
```

Such vectors are placed into OctaveFake objects, which must be read as strings and converted to Java double arrays using custom Java code.

2.5. Java Enterprise Edition

Since the Java based learning platform will be used through the WWW, components of Java Enterprise Edition (Java EE) are used. In this section, several aspects of Java EE are discussed in relation to the problems of web UI development and deploying a MATLAB® function for the web.

2.5.1. Servlets and JavaServer Pages

Java EE allows web pages to be dynamically generated on the server by servlets or JSP pages. The difference between those two is that, JSP pages are rather HTML pages having

[18] Full compatibility refers to being able to execute MATLAB® scripts in GNU Octave and vice versa without the need for modification.

[19] GNU General Public License

JSP tags embedded, whereas a servlet is technically a Java class. JSP pages are parsed and translated into servlet classes and executed as if they were ordinary servlets.

Servlets are equipped with a *PrintWriter* and an *OutputStream* object, which allow to send output back to the web browser in any format desired. The *PrintWriter* object is used for text-based output, like HTML markup, whereas *OutputStream* is used for binary data like images. Since the *PrintWriter* object is only capable of producing plain string based output, care must be taken when generating HTML markup. As the *PrintWriter* object is not able to check for syntax errors, the risk of sending malformed HTML is imminent and naturally grows with the complexity of the web page being generated. Such errors are hard to detect unless a validation service like W3C Validator [45] is used, because web browsers still try to render the page in a way they consider it to be most correct[20]. Servlets are also capable of managing sessions over HTTP and they allow attaching, retrieving and removing attributes to and from sessions.

JSP pages are chosen by Java EE software developers to generate the HTML markup of the view. This reduces the risk of malformed HTML significantly. Servlets, on the other side, are used as controllers, which interact with data sources or DAOs[21] or act as an adapter between the web browser and the business logic. Dividing view and controller in this way also allows to spread duties between business logic developers and user interface developers. Such a division is made possible by the servlet's *RequestDispatcher* object, which is capable of forwarding the request and the session including it's attributes to a JSP file. In case of having only a small HTML page to generate, it is however still feasible to omit the forwarding of the request and directly write the HTML markup using the *PrintWriter* object.

2.5.2. JavaServer Faces

When creating a sophisticated web-based user interface, Servlets and JavaServer Pages[22] do not provide an easy-to-use and easy-to-understand way of structuring the application. A popular choice for such a structure is the Model-View-Controller (MVC) software design pattern[23].

While Servlets and JavaServer Pages allow to imitate the MVC pattern using the *RequestDispatcher* class, the JavaServer Faces (JSF) user interface framework [46] is developed with this pattern in mind. As well as servlets and JavaServer Pages, JSF is standardized as well. The current standard is JSF 2.2. In this subsection, several aspects of JSF are discussed.

[20] This rendering mode is referred to as Quirks mode.
[21] Data Access objects
[22] Servlets and JavaServer Pages have been introduced in section 2.5.1.
[23] See section 2.3.

```
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>
        javax.faces.webapp.FacesServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>
        Faces Servlet
    </servlet-name>
    <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

Figure 2.1. *Deployment descriptor for Java Faces Servlet application*

JavaServer Faces implementations The JSF reference implementation is Mojarra [47], which is developed by Oracle and provided with the GlassFish application server [48]. Another JSF implementation is Apache MyFaces [49], which is installable alongside all Java EE application servers. Apache MyFaces is a natural choice to extend the functionalities of Apache Tomcat[24] by adding a JSF implementation.

Deployment of JavaServer Faces applications Since, internally, a JSF implementation is a servlet, which is started by the servlet container, the servlet container needs to be informed about that servlet. This is achieved by mapping a URL pattern to the JSF servlet in the deployment descriptor. An example URL mapping for the JavaServer Faces servlet is shown in figure 2.1. In this example, the JSF servlet is mapped to all URL starting with */faces/*.

In order to receive more verbose debugging information in case an exception has been thrown during the JSF lifecycle, the JSF based web application is developed in debug mode. This is also configured through the deployment descriptor file.

As a difference to plain servlet classes, when deployed in development mode, JSF pages do not require a republish operation nor a servlet container restart. It is clear, that this speeds up the web application development. The JSF servlet takes care of rendering the view pages, binding data and method calls to the underlying Java Beans.

Format of JSF pages JSF was originally specified in its first version to be based on JSP markup. This, however, has caused lifecycle problems with execution of the JSF pages. Thus, another markup language was sought. This finally led to the specification of JSF 2.0, which uses XHTML as the markup language. JSF Pages written in XHTML markup are referred to as Facelets. Due to lifecycle problems caused by the JSP markup, it has been deprecated with the introduction of the JSF 2.0 standard [50].

[24] Java EE runtimes, application- and web servers are discussed in section 2.5.3..

Since XHTML is based on XML, JSF pages have to be well-formed. Otherwise, the request for the page in question is rejected with an internal server error. Besides ordinary XHTML markup, JSF pages consist of additional special tags collected in taglibs[25]. These allow to bind data and actions to UI elements such as forms, text blocks et cetera. Also AJAX[26] calls can be bound to UI elements.

HTML forms are used to collect user input. Since JSF allows to create UI elements and to bind data and actions to UI elements, which include forms, input fields, et cetera, sophisticated user interfaces are buildable using JSF.

Managed Beans Another powerful mechanism of the JSF framework is the possibility to bind elements of the view to attributes and methods of underlying Java Beans. View elements, which are bound to bean attributes, are bound to the data model. On the other side, attributes, which are bound to bean methods, are said to be bound to the controller[27]. Thus, this binding mechanism establishes the link between the models, the views and the controllers. It is thus an essential part of JSF framework's MVC pattern realization[28].

Binding data to UI elements Since JSF is based on the MVC pattern, JSF applications possess a data model, which is represented in the back end code by Java Beans[29]. JSF markup and expression language [53] allow to bind UI elements to attributes of those Beans. Values are bound to JSF UI elements using the value attribute. An example of such a binding is:

```
<h:inputText
    id="boxcarAmplitude"
    value="#{boxcarParameters.amplitude}"
    size="4"
    style="text-align: right;" />
```

This line of markup defines an input text field with DOM ID *boxcarAmplitude* and binds it to the bean of class *BoxcarParameter* and to its *amplitude* attribute. The attributes on the end define the layout of this field.

Composing web pages and view reuse In contrast to plain HTML or XHTML, JSF allows to build complex web pages out of smaller XHTML files. This also allows to reuse view components like form elements, pager headers and footers.

The content of an includable XHTML fragment file is embedded into *<ui:composition>* tags. A very simple example is:

[25] These taglibs are specified in the JSF specification [51]

[26] AJAX is an abbreviation for Asynchronous JavaScript and XML, but transferred data is not restricted to XML.

[27] In JSF, controllers are bound to a UI component using the action attribute.

[28] MVC refers to Model-View-Controller software pattern, which has been introduced in section 2.3.

[29] Java Beans are Java classes, which follow the Bean specification [52].

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets">
    <p>Page Header</p>
</ui:composition>
```

In this example, an includable XHTML fragment, which adds at the place of inclusion the HTML markup fragment: * Page Header *, is defined.

Including external resources External resources like CSS style sheets and JavaScript files are also managed by the JSF framework. For this to work, they are stored in the *WebContent/resources* directory. From there they are included within the *<h:head>* section.

```
<h:head>
    <h:outputStylesheet
        name="style/jsfstyles.css" />
    <h:outputScript
        name="js/libs/jquery/jquery-1.10.2.js" />
    <title>
        <ui:insert name="title">FFT JSF</ui:insert>
    </title>
</h:head>
```

This markup includes the style sheet from *WebContent/resources/style/jsfstyles.css* and the JavaScript file *WebContent/resources/js/libs/jquery/jquery-1.10.2.js* into the web page and defines FFT JSF as the title of the web page. It is clear, that this file inclusion mechanism allows to abstract from the storage place within the JSF project as the JSF servlet takes care of looking up the files and inserting the correct markup into the rendered output.

Bean Validation A data model has to be consistent at any time during runtime. This reduces the risks of running malicious code and makes sure, that input and output of the application are always meaningful. A very simple example for this is a group of date input fields. One field collects the day of the month, a second one the month and a third one the year. The model has to make sure, that it only accepts valid dates. For example, inputing as day 31, as month February and as year 2006 is nonsense and thus has to be rejected.

Bean validation is an easy solution offered by the JSF framework and is achieved using validators. The first simple check, which it is performed, is for data type integrity. If, exempli gratia, only integer values are allowed for this attribute, any value, which is not convertible to integer, will be rejected. A second and yet more powerful feature of bean validation is the ability to check, whether a given input is within a desired range. This

allows to check for dates, which are not between 1 and 31. Such consistency checks are triggered by Java Annotations, which are shipped by the JSF framework. Annotations are, however, not powerful enough to check the consistency of the bean itself. For such cases, own methods are defined and bound to the UI. In case the data model is not consistent, a *ValidationException* is thrown, to which a custom error message are added within the current JSF context. On occurrence, this *ValidationException* is thrown to the view, which then displays the error message at its place defined by the JSF markup.

Failed bean validation always cancels the execution of controller functions. This prohibits erroneous data to be inserted into databases or malicious code to be executed. Still, it is obvious, that the programmer carefully has to define the data model including the values it accepts.

Lifecycle of a JSF application After the request has been forwarded by the servlet container to the JSF servlet, a rather complex process [54] is triggered before the rendered reply is transfered back to the browser. This process is briefly described in this chapter.

Upon startup, JSF binds event handlers and validators to the FacesContext[30]. Once the FacesContext object is set up, the request parameters are bound to the UI components. Data type violations cause exceptions to be thrown in this phase.

The next step is the validation of the request parameters using the methods discussed in the previous paragraph. If the validation phase has been completed successfully, the request parameters are bound to their respective bean attributes.

After all parameters are bound to their bean attributes, the business logic of the application is invoked. In the vast majority of cases, the business logic modifies attributes of the beans. This is caused by exempli gratia database lookups, calculations et cetera.

As a last step, the response is rendered. This is achieved by interpreting the given JSF markup and inserting values of the bound bean attributes into the markup. The final result, a valid XHTML web page, is then send back to the browser, which renders the XHTML markup on the screen.

JSF page navigation handling A JSF application has the ability to dynamically define on runtime, which view page will be rendered. This is useful to forward to error pages or to display the correct view based on the users choice earlier in the application.

JSF framework's page navigation handling is based on the request dispatcher mechanism of servlets[31]. Since the *RequestDispatcher* object simply forwards to a servlet or JSP file given as a string, also within the JSF framework the XHTML file to be rendered is defined using a string. This string is defined as the return value of the controller function, which was bound to the previously displayed page. If the controller function returns a

[30]The FacesContext [55] represents the state of the JSF request and as such, it is responsible to bind all involved elements together. It is transfered through the whole request processing chain until it response is sent to the browser.

[31]Servlets are discussed in section 2.5.1..

null reference, the same JSF file is used for rendering the page. A simple example of this method, which forwards to *responseJSFPage.jsf*, is:

```
public String process() {
    return "responseJSFPage";
}
```

A more comprehensive article about JSF page navigation can be found at [56]

JSF pages and JavaScript or CSS code In some situations, it is desired to generate JavaScript or CSS code dynamically. This works exactly in the same manner as in XHTML markup using Expression Language. Thus, this line of code generates a JavaScript variable with data:

```
var dataToPlot =
    #{samplingFormController.results.flotChartData};
```

It is obvious, that the data in *samplingFormController.results.flotChartData* must be a string having a valid JavaScript expression as the JSF framework is not able to detect syntactic correctness of expressions in other languages. Generating CSS code or any other text-based code is realized by using the same mechanism.

2.5.3. Java Enterprise Edition Runtimes

In order to deploy any Java code and MATLAB® components for the web, a Java EE runtime has to be employed. The most popular Java EE runtimes are Tomcat, GlassFish and JBoss. Those runtimes are briefly compared and a choice will be made for deploying the MATLAB® function.

Tomcat is a servlet container, which is used solely to deploy and run web applications, that follow the Java servlet specification. It is the simplest and most lightweight Java EE runtime. In order to run, it only requires a Java Runtime environment. Its current version is licensed under the Apache License version 2 and can thus be used free-of-charge for private, educational and commercial projects. The server is configured by a web based console, which does not work if the server is operated from Eclipse IDE. In that case, this feature is, however, not needed, because the deployment is done much easier using Eclipse's tools.

For software development purposes, Tomcat can easily be integrated into Eclipse IDE's Java EE perspective by adding it into the *Server* tab on the bottom of the Eclipse IDE. Once integrated there, the server can conveniently be started, restarted and stopped using the toolbar button of the *Server* tab. When the code deployed on Tomcat including the MATLAB® component changes, it gets redeployed automatically onto the server if the JVM is in debug mode and class hotswapping has been enabled.

Since Tomcat lacks many features, that a fully-blown JavaEE application server ships, the TomEE project [57] has been created to fill that gap, while still allowing Tomcat usage. TomEE ships with Apache MyFaces and others.

GlassFish is the Java EE reference implementation created and maintained by Oracle. Its current version 4 supports, but also requires Java EE 7's Java Development Kit (JDK). In addition to Tomcat's support for servlets, GlassFish supports the full Java EE 7 specification [58]. Since the source code of it's servlet engine is derived from Tomcat, in general, it runs servlet based web applications as in Tomcat. GlassFish allows for a comprehensive web based configuration of all of its components and features. Own experiments with the current version have however shown, that problems occur if additional native libraries need to be loaded, which is the case for deployed MATLAB® functions.

Another popular Java EE runtime is JBoss created by Red Hat [59]. Besides the ability to execute servlets, it also ships EJB[32], Hibernate, CORBA support, Rich Faces and others.

Since only JSF pages are created and additional features like Java Persistence or Enterprise Java Beans are not needed for this simple application, TomEE has been chosen due to its lightweightness and simplicity. Additionally, The MathWorks provides good documentation on integration of MATLAB® functions into web applications using Tomcat. TomEE has been chosen because it is smaller than a Tomcat installation, and it ships with Apache MyFaces, a JavaServer Faces, library.

[32] Enterprise Java Beans

3. DEVELOPMENT OF THE LEARNING PLATFORM

The development of the learning platform is performed using different software tools. In this chapter, these tools, the materials used to develop the learning platform along the development process are described.

3.1. Starting Point for the Thesis Work

In this section the currently used methods for teaching in the departments of Signal Processing and Communication Systems are introduced. This section thus purposes on introducing the context, in which the here developed e-learning platform, is placed.

At Tampere University of Technology and especially at the above mentioned departments, the traditional classroom teaching method is widely applied. Traditional classroom teaching consists of lectures, exercises, examinations and optionally homework assignments. Such homework assignments are done in order to upgrade the final course grade, upgrade the amount of credit points earned from that course or as a prerequisite to pass the course.

Lectures are usually given in lecture halls using PowerPoint® slides, overhead projector slides and lecture notes. Further explanations of the subject are given on the blackboard as this allows to react individually to student's questions. However, as a matter of fact, the time during lectures is limited, which does not allow to answer all student's questions completely and as deeply as needed. Attendance to the lectures is in the vast majority of cases voluntary, however it is strongly recommended by the teachers.

Exercises are given in order to give the students the possibility to recapitulate the knowledge, that has been taught during the lectures. For some courses, attendance or even active participation to the exercises is mandatory in order to pass the course. Exercises are also used to upgrade the final grade of the course. During exercise classes tasks are given to the students, which are solved using pen and paper, on the whiteboard or using MATLAB®. Some exercise classes require to solve the assignments beforehand. Laboratory-based exercises consist of pre-lab questions, which are answered before the exercise session, the actual laboratory exercise and a report, which is written after the exercise session and has to turned in after a rather short deadline. In the departments of signal processing and communication systems, also laboratory work is done mainly using MATLAB®, whereas C/C++ or Java are only used rarely. Teachers at the signal

processing and electronics department are very well skilled in MATLAB®. Programming skills in JavaScript or Java are less developed.

In addition to the exercise sessions, homework assignments are given to the students. They usually go deeper into the topic than the exercises or even stimulate the students to conduct additional studies.

Courses in English are offered to international and Finnish students. In order for new international degree students to enroll at TUT, they need to present a proof of sufficient language skills in English. This makes sure, that students are able to follow the lectures and to solve the exercises, homework assignments and questions in the examinations[60]. It also reduces the requirement of internationalization for the learning platform to English as many courses are taught in English nowadays.

As already mentioned earlier, on successful completion of a course, students are awarded with ECTS[1] credit points. The amount of credit points is based on the average time and work effort needed in order to complete the course. One credit equals approximately 27 hours of work [61]. Courses last at most one semester and at least one teaching period[2]. This ECTS credit point system allows to measure and compare the effort done by students EU-wide. It also allows to slice the students problem[3] of obtaining enough credit points to fulfill the degree requirements into manageable chunks of courses rather than having few huge courses like in problem-centered education[4]. Such huge courses put higher risks on the students in that sense, that failure leads to spending much time and effort while not getting any credits at all out of the problem-centered course. Additionally, courses with more work load prevent students from taking other courses in parallel.

At the TUT's department of Electronics, a learning platform, which is part of the Invocom teaching project, is existing. It is based on MATLAB® WebServer, whose support was discontinued by The MathWorks [62]. This led to a situation, which does not allow teachers to generate new learning material and especially interactive, demo-based exercises.

MATLAB® WebServer was discontinued, because it had used the CGI interface of a web server. Those interfaces are resource-intensive as for every request they start a new process instead of a new thread. Also security considerations concerning the CGI interface led The MathWorks to the decision to cease support for this technology with MATLAB® version 2006b.

There is, however, still a need to create hands-on-demo exercises for students to understand learning subjects from a different point of view than just the mathematical formulas presented in the lectures as well as the MATLAB® programming based exercises as mentioned in the beginning of this subsection.

---

[1] European Credit Transfer and Accumulation System

[2] A teaching period corresponds to seven weeks of teaching and one exam week and a semester consists of exactly two teaching periods.

[3] Here, the term problem refers to the whole study process.

[4] This term has been introduced in chapter 1.

3.2. Development Tools and Process

Since the learning platform is written in Java for a Java EE application server, a suitable IDE has to be sought. The requirements for such a IDE are syntax highlighting for Java, XML and XHTML as well as JavaScript code, easy integration of a Java EE application server, code assistance features and Git integration. Additionally, due to zero budget of the thesis, the IDE has to be available free-of-charge.

Richly featured Java IDEs are Jetbrains IntelliJ IDEA [63], Oracle's Netbeans IDE [64] and Eclipse [65]. IntelliJ IDEA is not available free-of-charge and thus not suitable for this project. Netbeans IDE is free-of-charge and comes with a wide range of features. The same applies for Eclipse. The software development itself has been carried out using Eclipse IDE for Java EE Developers [66], because it has a good support for Java EE development by integrating Tomcat into its user interface. Eclipse IDE also ships an easy to use, but comprehensive and verbose debugger interface.

As mentioned in chapter 3.1., many exercises in signal processing and communication engineering courses are given using MATLAB®. Thus, instructors have a very good knowledge of MATLAB® and feel comfortable employing it also for e-learning platforms. Thus, allowing instructors to express the underlying mathematics using MATLAB® functions is a natural choice.

To deploy MATLAB® code to a web server, MATLAB® Builder JA[5] has been chosen as it is capable of translating and packaging MATLAB® functions into Java classes. Java code itself can be processed by a Java EE servlet container like Tomcat [67], a Java-based web server capable of processing HTTP requests sent by web browsers.

Since the programmer has not much experience with integration of MATLAB® Builder JA components and the JavaServer Faces technology and the goals of the thesis project were not clear in the beginning, an agile software development method is used. This reduces the risks of setting too ambitious plans, which in the end causes the thesis project to last too long or even to fail. Agile methods also allow to specify new features[6] and a new look and feel on the fly rather than making complicated plans. It also allows to reduce the documentation work in the beginning of the software development process and to push the documentation, which is really required for the end-user or other software developers, to the end of the development process. In parts of the project, SCRUM has been employed in order to allow the stakeholders to follow the progress and in order to estimate the work and time remaining.

Based on the possible refinement due to the usage of agile methods, first the Fast Fourier transform exercise has been implemented. There were implementation iterations using plain Java Servlets, JavaServer Pages and finally JavaServer Faces. After the right UI technology has been determined, the UI components were developed. Out of those

[5] MATLAB® Builder JA has been introduced in section 2.4.

[6] Features are also referred to as use cases.

components, the exercises, which form the introductory signal processing course, have been developed. As a last step, the navigation UI elements were implemented.

Due to the agility of this thesis project, long theoretical studies have not been conducted. Thus, the theoretical part of this thesis mainly consists of material needed to understand the basics of the conducted work.

As a basis for the programming work, the Java EE specification by Oracle has been extensively used. For the solution of specific small problems, programmer platforms like StackExchange and JavaRanch have been sought. Additionally, HTML, JavaScript, CSS and XML specifications by W3C and tutorials by SelfHTML, W3Schools and tutorialspoint have been used.

The layout of the user interface has been developed with the existing, but legacy Invocom learning platform in mind. User interface mock-ups have been created using the Pencil UI software package [68].

3.3. Use Cases

Use cases describe how and by whom the learning platform is supposed to be used. In this chapter, the use cases for the learning platform are developed.

3.3.1. General Considerations

The learning platform to be developed within this thesis project serves the goal of supporting lectures and exercises. This means, that it does not replace teachers or teaching assistants, but gives the student a possibility to understand the subject from another point of view.

Students, as the main actors of the learning platform, shall be enabled to experiment with different fundamental aspects of digital signal processing and communication theory. This already poses some important requirements to the learning platform, which are listed in what follows.

An important role in a user-friendly learning platform is taken by the navigation. For the here developed learning platform, a bread crumb-based navigation system is needed, because it allows the user to go back to a page situated higher in the navigation hierarchy.

Since the learning platform is web-based and web-based applications are capable of displaying and following hyperlinks, links to related content within the learning platform as well as links to related and interesting pages in the WWW are displayed in the upper part of the pages. Such links shall be shown on the top of the page, where they can be found easily.

The user interface queries all parameters needed to perform the desired DSP tool from the student. This gives the student free hands on experimenting with those parameters to see the outcomes. Naturally the student is able to set multiple signal and algorithm

parameters at a time allowing her to study the effect of all parameters using the learning platform. The required user input parameters are obtained from the formula or the block diagram of the DSP tool or algorithm. This realizes the concept of online laboratories[7].

As with all software, user inputs need to be validated by the system. This has to be done before the algorithm of the DSP tool is invoked in order to avoid attacks to the system, overloading it leading to poor response times or even system outages. Poor response times lead to user frustration, whereas system outages render the system completely unusable requiring support staff to start the system up again. It is clear, that during weekends and in the evening hours such reboots are not feasible. However, during the weekends and in the evening hours the most percentage of total users is expected the use the learning platform.

After the student has set all the parameters for the DSP tool, she clicks the process button, which triggers the algorithm to be performed. When the algorithm has completed, the student gets presented the output using two dimensional charts. The charts are placed in the corresponding blocks. This allows the student to easily see the signal corresponding to its place within the signal processing algorithm. All charts need to be zoom-able along the x-axis by marking the ROI[8] using the left mouse button. The y axis is left in the initial, automatically configured state based on the minimum and maximum values of the displayed output and thus not affected by the zoom. A cross-hair mark follows the mouse pointer along the x-axis while it is on the diagram displaying the x and y values at the current mouse position. In order to enable the student to include the chart into exercise reports, it must be possible to save to chart to local disk.

The interpretation of the results is completely left to the student. However, a link to questions, which give hints for the interpretation and thus guides the student towards the learning goals is embedded. When used in conjunction with exercises, questions can be employed to aid writing reports are self-study notes, which themselves are possible to be taken into account when grading the course.

3.3.2. Case study: Fast Fourier Transform Page

This subsection presents an example of the points mentioned above on the example of a Fast Fourier Transform (FFT) demonstration page. The FFT example has been chosen due to its simple structure and because its block diagram only consists of three blocks. This reduces work when a change on the page is required. Such changes are needed often due to interactive nature of this project[9].

First, the input parameters adjustable by the user, are obtained using the well-known formula of the FFT:

[7] Online laboratories have been introduced in chapter 1.
[8] Region of interest
[9] The project type has been introduced in section 3.2.

$$F_m = \sum_{k=0}^{2n-1} x_k e^{-\frac{2\pi i}{2n} mk} \quad m = 0, ..., 2n-1 \tag{1}$$

Using this formula, it is determined, that the following parameters need to be given to the FFT algorithm. These are the signal x and the length of the FFT n.

It is worth noting here, that the signal x_k is entirely undefined at this point. Thus it needs to be defined using signal types and their corresponding parameters. This allows to study the effects of the FFT on different signal types. As with the DSP algorithm itself, the formulas of the signals reveal the parameters and are thus used the find them.

This learning platform uses sine, cosine, boxcar, sinc, impulses, Gaussian bell curves and constants as input signals as these are the most typical signals encountered in introductory signal processing and communication theory teaching. The user shall be able to select the signal type and gets presented with the correct set of input fields to define the signal further. These signals are discussed in what follows.

All signals have some parameters in common. First, the length of the x axis is defined. The learning platform thus queries the minimum and maximum values on the x axis along with the sampling frequency. These three parameters define the length of the signal in samples and thus the amount of data to be processed by the algorithm. Since every computer system has only limited amount of computational power and the user expects results after a reasonable amount of time, the length of the generated input signal has to be limited. Own experiments have shown, that a maximum input signal length of 2000 samples is a reasonable trade-off between performance and studying signal effects.

Sinusoidal signals are either sine or cosine signals. Their input parameters are easily derived from their formulas as shown by equations 2 and 3, respectively.

$$x(t) = A\sin(2\pi ft + \phi) \tag{2}$$

$$x(t) = A\cos(2\pi ft + \phi) \tag{3}$$

In the above mentioned formulas, A denotes the signal's amplitude, f the signal's frequency and ϕ the phase offset. These three parameters are the signal parameters and are thus queried from the user. Boxcar signals, also referred to as rectangular functions, have an amplitude and a width of their rectangle. Thus, amplitude and width are queried from the user if she chooses to generate a boxcar signal. Constant signals posses a constant value over the whole signal duration. Here, the user only has to set the value of the constant. Sinc signals, which are created when taking with DFT of a boxcar signal, have the following formula shown by equation 4.

$$x(t) = Asinc(2\pi ft + \phi) \tag{4}$$

As with the sinusoidal signals, the sinc formula reveals, that the frequency f, the amplitude A and the phase offset ϕ have to be queried from the user as input parameter. Gaussian bell curves are exponential signals, which are defined by the standard deviation σ and mean μ. The formula of such signals is shown by figure 5.

$$x(t) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \tag{5}$$

Equation 5 reveals, that the required signal parameters for Gaussian signals are standard deviation and mean.

After the student has set all required parameter, she clicks the process button in order to start the algorithm. After the button has been pressed, the given input parameters are validated. If the validation fails, an error message is printed and the algorithm is stopped.

As mentioned earlier in this chapter, the generated input signal and the resulting signals are displayed using charts. The outputs are separated using a horizontal line. In the first block, the input signal, which was generated according to the parameters given by the student, is displayed in a chart. In the remaining blocks, the charts required to fully depict the output of the algorithm, are shown. For the FFT case, the amplitude spectrum as well as the phase spectrum are required to be displayed.

In the initial state, the charts always show the entire signal leaving it to the user to zoom in for getting a more detailed view of the signals. The y-axis has been scaled to always allow 20% excess space from the minimum and maximum value. This does not only allow to embed the legend into the chart, but also ensures, that the signal is shown entirely in the chart without clipping or reaching the upper border of the plotting area. An example chart showing a sinusoidal input signal is shown in figure A.2.

4. THE LEARNING PLATFORM

This chapter introduces and discusses all findings from the user interface design, the software design and the software development phase. First, the prototype exercise is briefly introduced. This is followed by the development of the software architecture and the choice of software tools and libraries. Afterwards, the development process of the prototype exercise page is discussed. This is followed a detailed description of the user interface components. Out of these a model for creating more exercises is developed and evaluated using the "Introductory Signal Processing" course.

4.1. The Prototype Exercise

To study the process of integrating a MATLAB® Builder JA component into a dynamically generated web page, a demo for the Fast Fourier Transform has been created. In order to investigate its spectrum and phase characteristics, the Fast Fourier Transform transforms an input signal to its frequency representation. It is one of the most fundamental tools in signal processing and is thus essential for students to understand in order to carry on with their signal processing and communication studies.

The demonstration developed in this section consists of a web page, which allows the user to select different input signal types[1] and adjust their parameters[2]. After the user has made her choice, she clicks on the submit button, which triggers the back-end code. After the computation has been finished, the page presents the input signal, its amplitude and phase spectra plotted in separate line plots.

The user can repeat this process over and over again to study the properties of the Fourier Transform. An attached question sheet guides the student towards the learning goals. Hyperlinks in the "Related Content" section lead to material related to the exercise and thus exploit the nonlinearity of the student's learning process and the nonlinearity of the WWW.

[1] In this example, sinusoidal, impulse, constant, sinc, boxcar and Gaussian bell curved signals are employed. These are defined in section 3.3.1..

[2] Adjustable parameters in this example are depending on the input signal type: sampling frequency, amplitude, signal length, signal frequency, mean, standard deviation and boxcar width.

4.1.1. Design of the Pages

The goal of the page design is to create components, that can be reused for other demonstration pages. This saves disk space and reduces the amount of work needed to create more exercises.

When the page is retrieved for the first time, the user is presented with preset input parameters. The user can simply accept them by clicking submit or change them according to her needs. The input parameters are collected in a HTML form, which is grouped into logical blocks according to the block diagram of the DSP tool. The desired signal type is selected by a HTML selector. Based on its current selection, the signal-type specific groups are switched in a way, that only the input fields used for the selected signal type is displayed and the others are hidden. The fields for the common signal parameters like minimum and maximum values on the x axis and the sampling frequency are, however, shown for all signal types.

As with every web application, user input has to be verified and checked for plausibility in order to avoid security holes. This is discussed in section 2.5.2.

Results are displayed in their corresponding blocks using basic two-dimensional line charts. The units shown on the axes of the charts depend on what is displayed in the chart. The y axes is scaled automatically based on the minimum and maximum values of the corresponding axis and the x axis based on the current zoom. The user has the possibility to zoom into the diagram. This zoom, however, is restricted to the x axis in order to always be able to show the curve fully in its vertical extend.

Based on the considerations in section 3.3., the first mock-up of the user interface for the FFT page has been created. It is shown in figure A.1. The user interface mock-up has been iteratively refined using the results obtained from the development of the FFT prototype page. It is created using the Pencil UI software package, which allows to directly insert UI components rather than drawing them as it is done using basic graphics processing software packages like MS Paint, GIMP or Adobe Photoshop.

4.2. Software Architecture

In this subsection, the software architecture for the teaching pages is developed. The software architecture serves as a base for creating a model for adding more exercises to the system later.

The Model-View-Controller pattern (MVC) as shown in section 2.3. can be employed in web applications as well. Yet, it keeps the system easily maintainable and understandable. Thus, the MVC pattern is chosen for the learning platform, which is developed in this thesis project.

The general structure of the demonstration is introduced here. It is visualized by figure A.4.. The visual part of the teaching page is the view. It is realized as a XHTML page

having CSS style sheets and JavaScript code embedded into it in order to improve the interaction with the page and to improve its look and feel. Apart from the visualization, the view is also responsible for collecting the input parameters from the user.

In order to interact with the rest of the application and to forward the user input, the view has its own data model, the View Data Model, associated with it. This data model is responsible for validating and storing the inputs of the user.

When the user triggers the algorithm, the controller is invoked. For this to function properly, the controller must be associated with the view. The task of the controller is to trigger the conversion of the view's data model to the one needed by its associated Data Access Object (DAO).

The DAO is then invoked by the controller and invokes the MATLAB® component associated to the it. After the DAO has received the results of the algorithm from the MATLAB® component, it creates the data model, which holds the results. This results data model is then queried by the view in order to present the results to the user.

Next, the general work flow mentioned above is evaluated on the Fast Fourier Transform demonstration page[3]. Applying the model introduced above to the FFT demonstration page directly is principally possible, but leads to issues when thinking of reusing view components and code created for this module in other learning units as the pages will share many UI components.

Thus, it carefully has to be thought on how to split the view and its data model in a way, that as many reusable components as possible are generated, which can be reused in order learning units.

As shown in the use case definition in section 3.3., input signals have to be selected and their parameters have to be defined. When creating more of these demo pages, users will need the ability to define those signals and parameters there as well. Thus, it is a natural choice to separate the view into field sets for the different signals as concepted in section 3.3.2. Figure A.6. shows the division of the input field sets to different fragments. From that figure, it can be seen, that for each signal type a separate fragment is created, which resembles an input field set. Based on the signal type selection, its associated field set is shown to the user.

In addition to the parameter required by the input signals, the FFT algorithm also requires the FFT length as an additional parameter. Such parameters are easily embeddable into the FFT UI block.

In order to exploit the division of the input field sets into fragments while still maintaining the MVC pattern, data models are created for each fragment. The data models are initialized using default values, which are empirically obtained. Care is taken upon model definition, that the data is consistent and does not overload the server with too complex computation requests. This has been done by allowing only a signal length of 2000 samples as shown in section 3.3.1. If the data model validation fails, the exact reason for

[3]The FFT demonstration page is introduced in section 4.1.

failure is printed below to the form and the actual algorithm is not invoked. In order to leave freedom to the user to experiment with the parameters, restrictions on other parameters than the signal length have not been imposed. The relationship between the view and its data models is depicted by figure A.7.

This results data model holds the signals obtained by the FFT algorithm. It is used to create charts displaying the results of the algorithm. In the case of the FFT, it contains the input signal, its amplitude and power spectra.

In order for the FFT MATLAB® DAO to be able to easily invoke the FFT MATLAB® component, the data has to be in a format, which follows the parameter list of the deployed FFT MATLAB® component and consists only of basic Java types and strings. Thus, a MATLAB® parameter model has been introduced, which allows to abstract the user interface data model from the parameter formats of the underlying MATLAB® function.

A button, which triggers the processing action of the user supplied parameters, is associated with the view. Upon click, the view data validation is triggered first. If this operation succeeds, the controller is invoked. Its tasks are to transform the view data model into the FFT MATLAB® parameter model. The controller then invokes the FFT MATLAB® DAO with the transformed data model. After the algorithm has completed, the DAO has returned the results as a results data model object, which is forwarded to the view. Before it is able to perform this operation, it transforms the data model values into native MATLAB® Builder JA objects, which are then passed to the FFT MATLAB® component.

After the call to the MATLAB® Builder JA component has returned, the DAO unwraps all results from the returned native MATLAB® Builder JA objects and generates a result data model object out of it. This object is then passed through the controller back to the view, where it is displayed in charts. Furthermore, the FFT MATLAB® DAO is responsible for initializing and destroying the MATLAB® component as well as creating and destroying native MATLAB® objects allocated by the DAO itself.

The MATLAB® component is invoked by the DAO and performs the actual FFT algorithms and returns the input signal as well as its amplitude and phase spectra. This component is discussed in more detail in section 4.3.1.

4.3. Creation the Prototype Exercise Page

This section discusses how the Fast Fourier transform exercise page has been created and deployed. This page serves as a field to study different software libraries and to develop the software architecture developed in section 4.2. and a model to create more exercises by the teaching staff, which is developed in section 4.7.

4.3.1. Creating the MATLAB® Function

First, it needs to be defined, what the MATLAB® function is supposed to do and what are its input and output parameters. The function has to carry out the Fast Fourier transform (FFT) on one-dimensional signals. In order to input a signal to a DSP tool, the signal first has to be created. This is performed by the *inputSignalGeneratorStruct* function. As mentioned in section 3.3.2., one type of signals is sinusoidals and sinc signals having a given frequency, amplitude, length and are sampled at a given sampling frequency. Another type are impulses with a given amplitude and offset from $t = 0$. Constant signals with a given length and value as well as boxcar signals with a given length, amplitude and duration of that amplitude and Gaussian bell curved signals with a given length, standard deviation and mean can also be generated by this function.

The function outputs the input signal and its amplitude and phase spectra as an array of double values. The first double arrays holds the input signal fed into MATLAB®'s *fft* function, whereas the second arrays holds the amplitude and the third array holds the phase spectrum of the given signal. Thus, the signature of the MATLAB® function is the following:

```
function [x, y, fftXAxis, fftAbs, fftPhase, funcEquation,
   diagramMetadata] = FourierTransformDemoStruct(
   inputSignalParameters, NFFT, showFftLogScale)
```

Using *deploytool*, the above mentioned function is compiled into a jar file, which is deployed on an ordinary desktop computer or within a servlet container. The user interface of *deploytool* prompts for embedding the function into a Java class. In this demonstration, the Java class is named *FourierTransformDemoStruct*.

The compilation process requires Java Development Kit (JDK) to be installed onto the computer and it's *bin/* folder to be on the *PATH* environment variable. After the compilation has succeeded, the jar package with the *FourierTransformDemoStruct* classes in it has been created. Its prototype is the following:

```
public class FourierTransformDemoStruct {
    public FourierTransformDemoStruct(); // the constructor
    // the actual MATLAB function
    public Object[] FourierTransformDemoStruct(
        int numberOfOutputParameter,
        Object... inputParameters);
}
```

This component is then deployed with all libraries, on which a web application[4] depends, to the *WEB-INF/lib* folder, from where Tomcat loads everything up once the servlet is initializing.

[4] A web application is a collection of static HTML pages, JavaScript files, style sheets, servlets and JSP files

Since MCR depends on its own native libraries, also those libraries are reloaded every time the servlet initializes if they were placed in the *WEB-INF/lib* folder. Such a reload of native libraries, however, results in a *java.lang.UnsatisfieldLinkError*, because native libraries can only be loaded once. Thus, the *javabuilder.jar* file of the MCR has to be deployed into a location, where it gets loaded only once upon server start. Such locations are the path of Tomcat's common class loader or Tomcat's shared class loader. Those paths are adjustable in the *catalina.properties* file. The directory, where the native MCR libraries are stored, are given to the JVM running Tomcat either as environment variables or as the *-Djava.library.path* parameter.

In order to abstract from the work needed to access the MATLAB® component, a data access object (DAO) is created. Such DAOs are widely used to abstract from low-level database or file reading and writing operations or invoking other data generating components like the MATLAB® Builder JA components. The responsibilities of the MATLAB® DAO are discussed in section 4.2..

4.3.2. Creating the Java Code

In the first part of this subsection it is discussed how to create a servlet based sample application, that interacts with the above created MATLAB® component. The second part of this subsection covers how this sample application is created using JavaServer Faces.

A new servlet class named *FftDemoServlet* is created as part of a web application named *TomcatTest*. The servlet class imports the MATLAB® component class as well as all classes needed from the MCR. In order to optimize performance, the MATLAB® component classes is instantiated in the servlet's *init()* method. This avoids reinstantiation of the component class every time the servlet handles a new request.

For testing and demonstration purposes, it is best to have all parameters to the transfered servlet using the HTTP GET method[5]. This allows modifying them on the fly in the web browser's address bar. Later on in production, HTTP POST will be used in order to hide the parameters from users. This avoids the possibility of easily overloading the server by adding huge numbers into the GET request string.

Servlets handle HTTP GET requests in their *doGet(HttpRequest request, HttpResponse response)* method. The request parameter object contains all given parameters, which the *FftDemoServlet* retrieves and checks for consistency. The consistency check avoids sending erroneous and too huge parameter values to the MATLAB® component. After all parameters have passed the consistency check, they are wrapped into MCR objects like *MWNumericArray* or *MWCharArray*.

Now, that all input parameters are ready and the MATLAB® component has been instantiated, the MATLAB® function is called. The results of the call are collected into

[5]The HTTP protocol has been introduced in section 2.1.2.

the returned array of *Object* named *results*. The first input signal WebFigure is unpacked from the *result* array using:

```
MWJavaObjectRef ref = (MWJavaObjectRef)results[1];
WebFigure inputSignalPlot = (WebFigure)ref.get();
```

The amplitude and phase spectra plots are unpacked in the same way. As a next step, the WebFigures are attached to the session, so that they are retrievable by the displaying JSP page. This is demonstrated by the following code snippet.

```
HttpSession session = request.getSession();
session.setAttribute("inputSignalPlot",inputSignalPlot);
session.setAttribute("inputSignalPlotBinder",new
   MWHttpSessionBinder(inputSignalPlot));

session.setAttribute("outputSignalPlot",outputSignalPlot);
session.setAttribute("outputSignalPlotBinder",new
   MWHttpSessionBinder(outputSignalPlot));

session.setAttribute("phasePlot", phasePlot);
session.setAttribute("phasePlotBinder", new MWHttpSessionBinder(
   phasePlot));
```

As a last step, all native MATLAB® resources have to be freed. This includes all instances of *MWArray* and its subclasses as well as the *MWJavaObjectRef* objects and the *results* array. This is done by calling *MWArray.disposeArray(...)* on the object in question. When the lifetime of the servlet itself has ended, the instance of the MATLAB® component class is disposed in the servlet's *destroy()* method.

During the vast majority of operation, errors can occur. These are indicated by Java exceptions thrown back to the calling function. They have to be properly handled. Some errors come from erroneous user input, others are generated by misconfigured and wrongly used software components. In the phase of software prototype creation, exceptions are only logged to the console of Tomcat.

The above mentioned cleanup operations like freeing memory from native MATLAB® resources have to be done regardless whether an exceptions has been thrown or not. Thus those operations are performed in a *finally* block, which is always executed.

Next, it is discussed how a JavaServer Faces application using the above generated MATLAB® component is created. Based on the parameters needed for the called MATLAB® function, the data models are created as they serve as a base for the UI and the DAO. As a second step, the DAO is created for the MATLAB® function and tested. Then, the UI is designed[6] and the JSF XHTML markup is created[7]. Finally, the controller, which wires up everything, is generated. By separating the responsibilities to all of those

[6] See section 4.1.1.
[7] See section 2.5.2.

components, the source code stays easily readable and understandable. Furthermore, the well-known MVC pattern is realized allowing other software experts to understand the system easily.

As shown in figure A.6., the view is composed of XHTML fragments in order to improve reusability of code. Thus, the main XHTML file of the FFT view only includes the necessary CSS style sheets and JavaScript files and defines spaces for the charts displaying the output signals. The input signal and FFT parameter blocks are included from other XHTML fragment files.

The application consists of more than one data model. This is done in order to reflect the different domains, in which the data is used. The data model of the view's input fields is responsible for collecting the users inputs and validating them. The data model of the MATLAB® DAOs holds all data needed for the invocation of its MATLAB® function. Finally, the result data model has all data prepared to be displayed to the user as charts and their labels and axes labels. The advantage of these different data models is, that the views or DAOs do not need to transform any data. The transformation is solely triggered by the controllers and performed by the corresponding data models, which are part of the business logic.

4.3.3. The Charts

Since the charts display the outputs of the algorithms given the parameters inputted by the user, special attention has to be payed to the output visualization as this is a prerequisite for the interpretation of the results by the end-user[8].

Since MATLAB® Builder JA, which has been introduced in section 2.4., is used, a natural choice is to use its build-in WebFigure feature. On the other side, JavaScript chart libraries can be employed to embed visually appealing and interactive charts. As an example, the Flotcharts library is introduced in section 2.2.3..

In this subsection, the WebFigures produced by MATLAB® Builder JA [42] and the JavaScript figures produced by the Flotcharts and Highcharts libraries as well as the workflows with both libraries are compared.

Workflow for MATLAB® WebFigures The workflow for retrieving a WebFigure and attaching it to the current session is described in section 4.3.2. As mentioned in section 2.4., a MATLAB® WebFigure is embedded into a web page by using the JSP tag library supplied by the MATLAB® Builder JA software package or by requesting HTML markup from MATLAB® Builder JA.

If the view is realized using a JSP page and a servlet has forwarded the request to the JSP page using a *RequestDispatcher* object, the WebFigures JSP tag library is a natural choice as it only requires to insert JSP markup. Thus, JSP scriptlets are avoided. First,

[8] More on the results interpretation is written in section 3.3.1.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
       pageEncoding="UTF-8"%>
<%@ taglib prefix="wf" uri="http://www.mathworks.com/builderja/
   webfigures.tld" %>
<html>
<head>
<title>MATLAB Builder JA WebFigure</title>
</head>
<body>
    <wf:web-figure name="outputSignalPlot" scope="session" width
       ="600px" height="500px" />
</body>
</html>
```

Figure 4.1. *JSP code snippet displaying a MATLAB® WebFigure object*

```
WebFigureHtmlGenerator generator =
    new WebFigureHtmlGenerator("WebFigures",
                               getServletContext());
String webFigureMarkup =
    generator.getFigureEmbeddedString(outputSignalPlot,
                                      "outputSignalPlot",
                                      "session",
                                       null,
                                       null,
                                       null);
```

Figure 4.2. *Java code retrieving HTML markup for embedding a MATLAB® WebFigure into a web page*

the tag library has to be declared in the beginning of the JSP file. In this example, as the tag prefix, *wf* is used. After its declaration, it is used later on using the <*wf:web-figure*> tag. This tag needs the name of the WebFigure object defined in the servlet, which has attached it to the scope. Additionally, the scope in which it has been attached to the servlet context, is required. Optionally, the size of the WebFigure on the screen is defined in the <*wf:web-figure*> tag. This is demonstrated by the code snippet shown in figure 4.1.

Another possibility to insert MATLAB® WebFigures into a web page is to create HTML markup and use this directly in the web page created by a servlet or a JSP file. This is achieved by using MATLAB® Builder JA's *WebFigureHtmlGenerator* object. This object creates a string, which embeds the WebFigure[9] into a HTML iframe object. Assuming, that the WebFigure is already attached to the context of the current servlet, markup generated for it by the code snippet shown by figure 4.2.

[9]To be more precise, the WebFigure is technically another servlet, to which the web browsers sends HTTP requests using AJAX.

From that code snippet, it can be seen, how the "WebFigures" string tells the markup generator, that the WebFigures servlet is mapped to the *WebFigures* path and the *outputSignalPlot* object refers to a WebFigure object.

Additionally, in [42] it is also mentioned the possibility to include static images of the WebFigure into a web page. Since this does not allow for interactivity, this option is not considered within this work.

Workflow for Flotcharts Flotcharts rely on data in JavaScript format. Thus, the data returned by the MATLAB® component has to be transformed into a JavaScript array suitable for Flotcharts. A workflow creating a simple chart using Flotcharts is described in section 2.2.3.

For the FFT example, the data is transformed into a string processable by Flotcharts and inserted as JavaScript code into the XHTML markup generated by the JSF page[10]. Compared to the simple example shown in section 2.2.3., for the FFT example, there is a need to supply options to Flotcharts plot function in order to display a chart according to the requirements stated in the use case section 3.3. The configuration options are listed in the JavaScript code snippet in figure A.12. In order to make the cross-hair[11] feature working, the *jquery.flotcharts.crosshair.js* and *jquery.flotcharts.navigation.js* plugin files have to be included into the main HTML document, along the jQuery and Flotcharts library JavaScript files.

Adding axes captions and chart titles is not yet implemented into the Flotcharts library. Thus these need to be embedded in a 3x3 table. The chart's <*div*> container itself is placed in the center cell, whereas the y axis caption is placed in the left cell of the middle row, whereas the x axis caption is placed in the middle cell of the lower row. The caption of the chart is placed in the middle cell of the upper row. Thus, this problem is solvable without much effort.

Workflow for Highcharts As with the above mentioned Flotcharts library, charts are produced by the Highcharts library from JavaScript double arrays. This leads to the same principal workflow for Highcharts charts as with Flotcharts. Thus, for the general workflow is referred to the previous paragraph.

Flotcharts and Highcharts, however, differ in the format of their configuration option object. This requires to create two new functions, which create the required JavaScript code fragments. A major code refactoring is, however, not necessary to switch between Flotcharts and Highcharts.

Compared to Flotcharts, Highcharts reduces the programming work needed in order to get the zoom along the x axis, the axes captions and chart title, the horizontal

[10] JSF, JavaServer Faces are introduced in section 2.5.2. and the workflow for creating the JSF based FFT demo is introduced in section 4.3.2.

[11] A cross-hair refers to a thin vertical line following the mouse pointer in x direction over the chart.

zoom and the crosshair feature working. These features are simply enabled through the *highchartsOptions* object.

Additionally, the save to disk feature in Highcharts works significantly more reliable than with Highcharts. This achieved by a better implementation of the canvas2image[12] feature. If the browser is not capable of HTML5 canvas, the chart can be send to a third-party HTTP server, which takes care of converting the chart to a downloadable image.

Comparison of MATLAB® WebFigures and JavaScript based chart frameworks Both chart libraries allow to generate the line charts for the FFT amplitude spectrum. If the chart is not zoomed at all, both charts look similar and are visually appealing.

Problems, however, occur when charts are zoomed. MATLAB® WebFigures only zoom equally along the x and y axes. Both axes, however, are dropped out of the chart if the chart is zoomed in much. This leads to the fact, that the user is no longer able to read the axes and thus does not know at which values the plot currently is situated. In an application like this learning platform, hiding axes from and thus disabling the user to properly read and interpret the results renders the charts unusable. This is especially a problem as the zoom behavior of MATLAB® WebFigures cannot be influenced neither by the programmer nor by the user of the learning platform.

The JavaScript based chart frameworks like Flotcharts, on the other side, allow for zoom configuration. This means, that the Flotcharts and Highcharts frameworks allow to configure the zoom method. It can either be configured to zoom equally along the x and y axis, but it is also possible to zoom only along the x axis. Flotcharts also allows to insert multiple plots into a chart. Additionally, the chart always show the axes when zoomed, which allows the user to read the axes tick labels. It is also possible to insert a crosshair into the chart. Along with the cross-hair, a legend can be inserted, which shows the values on the x and y axes at the current mouse pointer position.

To sum up the comparison, it has to be said, that MATLAB® WebFigures is easier from the programmer's point of view as it requires only MATLAB® knowledge. Since, as mentioned in section 3.1., the MATLAB® skills of the teacher are very good, this appears as the natural choice to implement the learning platform. The JavaScript based chart libraries, on the other side, require skills in JavaScript programming language. They, however, offer much more configuration options, are very user-friendly and intuitive charts can be produced.

Due to the severe drawbacks of MATLAB® WebFigure's user interface compared to its JavaScript based counterparts, for the learning platform to be developed within this work, Highcharts has been chosen. The Flotcharts framework has not been chosen due to its missing direct support for axes labels and chart titles as well as their unreliable implementation of the save to local disk feature as discussed in section 2.2.3.

[12]The canvas2image feature allows to retrieve the chart as an image using HTML5 canvas.

4.4. Directory Layout

In this section, the directory layout for the learning units is developed. JavaServer Faces projects follow a specific directory layout in order to keep everything simple and manageable. Further requirements are also imposed by the usage of Java packages.

All web resources including the JSF markup files are stored under the *WebContent/* folder. Here, the fragments of the UI are stored in a subfolder *BuildingBlocks/* in order to separate them from the parent view files[13], which are served to the browser. The building blocks are the parts out of which the user interface is assembled. This implies, that those files cannot be used alone. Examples for UI building blocks are the input signal parameter block and the FFT block.

The actual web pages including the exercises are stored below the *courses/* folder. For each course, a representing subfolder is created. Below these course subfolders, folders for lecture material and exercises are created. Below these folders, the lecture units are stored.

The JavaScript files and CSS style sheets are stored in the *resources/* folder. As mentioned in section 2.5.2., this allows the abstraction from their actual storage using <*h:outputScript ...*> tags in the JSF markup. Thus, those files can be moved to another location if needed without the need to change the paths in the <*h:outputScript ...*> tags.

Within the *resources/* folder, subfolder *style/* is created to separate the CSS style sheets from the JavaScript files, which are stored in the *js/* subfolder. The JavaScript libraries are further separated from each other using another subfolder hierarchy.

The *WEB-INF/* folder is needed by the servlet container in order to run the web application. It contains the deployment descriptor[14] *web.xml* and all additional Java libraries deployed alongside the web application. The compiled Java classes are stored in the *WEB-INF/classes* folder. Within this folder, for each separate Java package a subfolder is created. Id est, a class *fi.frohling.learningplatform.datamodel.BoxcarParameterModel* is stored in *WEB-INF/classes/fi/frohling/learningplatform/datamodel/BoxcarParameterModel.class*.

4.5. Navigation

In this chapter, the method how navigation elements are included into the learning platform, is discussed. The navigation back to parent pages is realized by bread crumbs. Bread crumb refers to a hierarchical list of navigation items, that leads to the currently opened page.

Since the JavaServer Faces technology does not provide a mean to generate such lists, an own solution is created and described here. Such a hierarchical list is described using

[13] These parent view files are the exercise units and the index pages.

[14] An example deployment descriptor for JSF based web application is shown in section 2.5.2.

a tree data structure. In order to make it easily editable from the outside, a XML file is used. Its name is *breadcrumb.xml*. It holds the entire navigation tree as a list of *TreeNode* elements. Each *TreeNode* element has links to its parent and its child nodes as well as links to pages related to it. The bread crumbs contain as payload the URI of the node and a descriptive text as well as links to their parent and child nodes.

The class *BreadcrumbList*, which is part of the learning platform's data model, is responsible for reading in the XML file and converting it into the corresponding Java data structure, *List<Breadcrumb>*. The corresponding XHTML view files query the list of breadcrumbs based on the URI of the current request.

The homepage of the learning platform as well as the homepages of all courses stored in the learning platform need to display a list of links of courses and learning material, representatively. For this purpose, the above mentioned bread crumb data structure is employed as well. In this case, the *children* attribute is used to store all bread crumbs, which represent content below the currently viewed page. The view XHTML file simply queries this list and displays it as a HTML unordered list[15].

As mentioned in chapter 3.3., material related to the currently viewed learning unit is valuable. Thus, a mechanism to include such links is needed along the bread crumb navigation. For this purpose, the data structure in the XML file *breadcrumbs.xml* is extended by adding the links to related pages to the data structure. After reading in the file by an object of class *BreadcrumbList*, the view calls a function, which returns links to all related pages of the currently viewed URI. The view code displays them sorted by internal and external links on top of the page.

4.6. User Interface Components

In this section, the user interface components of the learning platform are introduced. First, the basic layout of the UI and then the actual components are introduced.

As mentioned in section 3.3.1., the UI components are displayed as blocks. This is done to mimic the block layout, which allows to grasp the signal flow much easier rather than using mathematical formulas.

All inputs and outputs related to the blocks are collected and displayed within those in order to provide an intuitive user interface to the learner. These blocks can be collapsed and expanded in order to give the learner a chance to easily see the whole block layout on smaller screens.

All components have to be reusable in order to facilitate their quick and easy use in the actual exercise pages. This is achieved by passing the reference to their backing bean[16] as a parameter using the JSF tag

```
<ui:param name="mybean" value="#{controller.dataModel.object}">
```

[15] In HTML, an unordered list is defined using <ul> tags. Its list element are defined by <li> tags.

[16] In Java Enterprise Edition, the term backing bean refers to the data model.

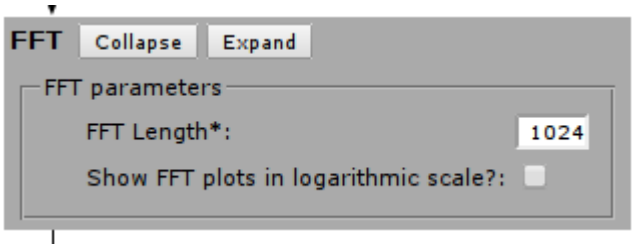

Figure 4.1. *Fast Fourier transform block*

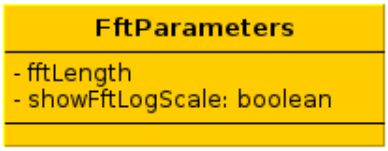

Figure 4.2. *Fast Fourier transform data model*

and by assigning a unique name to them using

```
<ui:param name="myname" name="myname">
```

For the exercises of the course "Introductory Signal Processing", the following blocks were created: Fast Fourier transform block, Matrix input and output block, Multiplicator block, arrows and nodes, summation block, sampler block, quantization block, chart display container and input signal parameter query container. These are briefly introduced in the remainder of this section.

The Fast Fourier transform block is used to collect the length N as a parameter of the FFT and whether the result shall be displayed in linear and logarithmic[17] scale. The FFT block is shown in figure 4.1.

The convolution block is a passive block, because as such it does not collect any input from the user nor displays any results. Thus, it does not need any backing bean assigned to it. The convolution block simply consists of a mathematical convolution sign. The convolution block is displayed in figure 4.3.

In order to be able to collect scaling factors, which are used exempli gratia in the FIR[18] filtering exercise, the multiplicator block has been developed. This block consists of a multiplicator symbol and an input field, which queries the scaling factor from the user. Since data is queried by the multiplicator block, a backing bean has to be provided to the component. A screenshot of this block is shown in figure 4.4.

Figure 4.3. *Convolution block*

[17] For the logarithmic scale, decibel (dB) is used.
[18] Finite Impulse Response

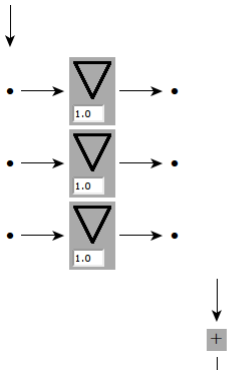

Figure 4.4. *Multiplicator blocks used in a FIR filter*

Figure 4.5. *Multiplicator block data model*

Arrows and nodes are employed to visually connect the building blocks of the user interface. They are technically not related to any functionality and thus they do not need any backing bean being provided. Arrows come as vertical arrows pointing downwards and horizontal arrows pointing in both direction. Nodes are simple black dots.

The summation block is used to indicate, that summation of multiple signals occurs at the point being. Since summation does not need any parameters, no backing bean is provided for this block. Visually, this block only contains of a plus sign. The block is shown in figure 4.6.

The sampler is a block, which collects the sampling frequency as input parameter from the user. Thus, a backing bean has to provided to this component. The sampler component is shown in figure 4.7. and its data model is shown in figure 4.8.

The quantizer block is similar to the sampler block having one input field. This input field collects the resolution of the quantized signal in bits. Since it collects user input, a backing bean is needed to be provided to the component as a UI parameter. The quantizer block is depicted in figure 4.9. and the data model is shown in figure 4.10.

After results are computed by the deployed MATLAB® function, results are displayed using the chart block. This block requires as UI parameters the ID of the DOM container,

Figure 4.6. *The summation block*

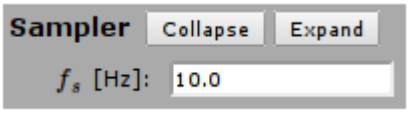

Figure 4.7. *The sampler block*

Figure 4.8. *The data model of the sampler*

which holds the chart, the customization options of the chart[19], the data series to be plotted and the caption of the chart[20].

The main element of the chart block is the chart itself. The charts are realized using the Highcharts JavaScript library[21]. The charts can be hidden and recalled by pressing the Hide and Show buttons on the top. The chart is resizable like ordinary browser frames and by using the plus and minus buttons on top of the chart. A copy of the chart can be printed or saved to disk by clicking the disk icon on the top right of the widget. Horizontal zoom is performed by marking the desired area inside the chart. The Reset zoom button allows to return to the full data view. An example of the chart block is shown in figure 4.11.

The input signal block is a complex UI component used to collect all required input signal parameters from the user. It is backed by a bean, which is structured the same way as the UI component in order to make this complex structure easily understandable by both end-users and programmers. The first part, which is always visible, is the common signal parameter query group. It is used to select the type of the signal and x axis range and the sampling frequency of the signal to be generated. The sampling frequency used in the input signal block must not be mistaken with the sampling frequency used for the sampler component. The common signal parameter block also checks via an AJAX call whether the length of the generated signal is short enough in order to avoid high load on the server caused by a demanding request.

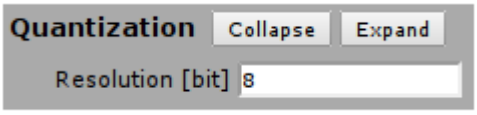

Figure 4.9. *The quantizer block*

[19]The chart is customized using the deployed MATLAB® function in order to provide an unique plotting interface to the creators of the learning units.

[20]This is usually the equation of the displayed function, but in principle any free text is possible to be provided here.

[21]The Highcharts library is introduced in section 2.2.3.

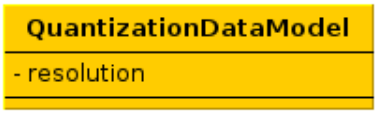

Figure 4.10. *The quantizer data model*

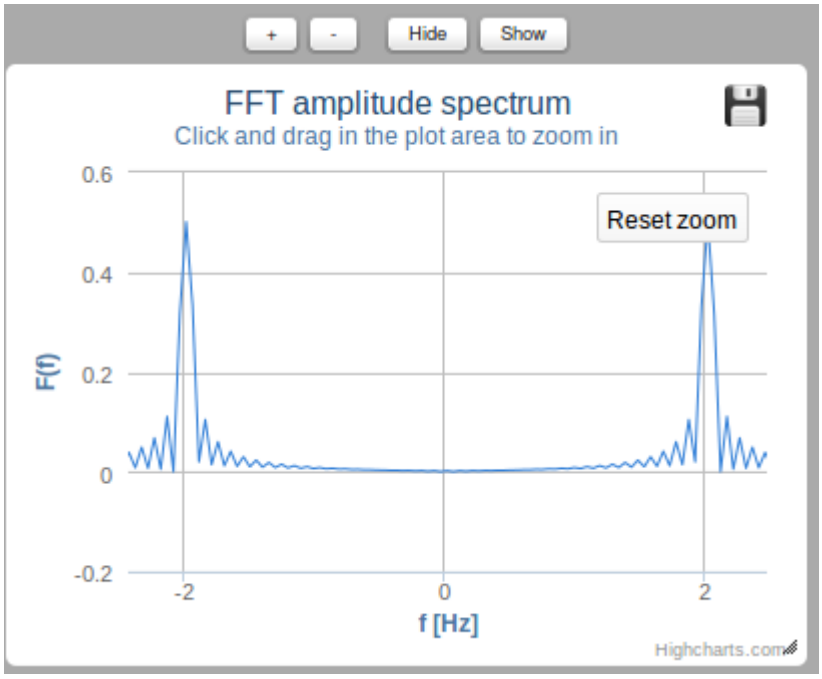

Figure 4.11. *The chart block*

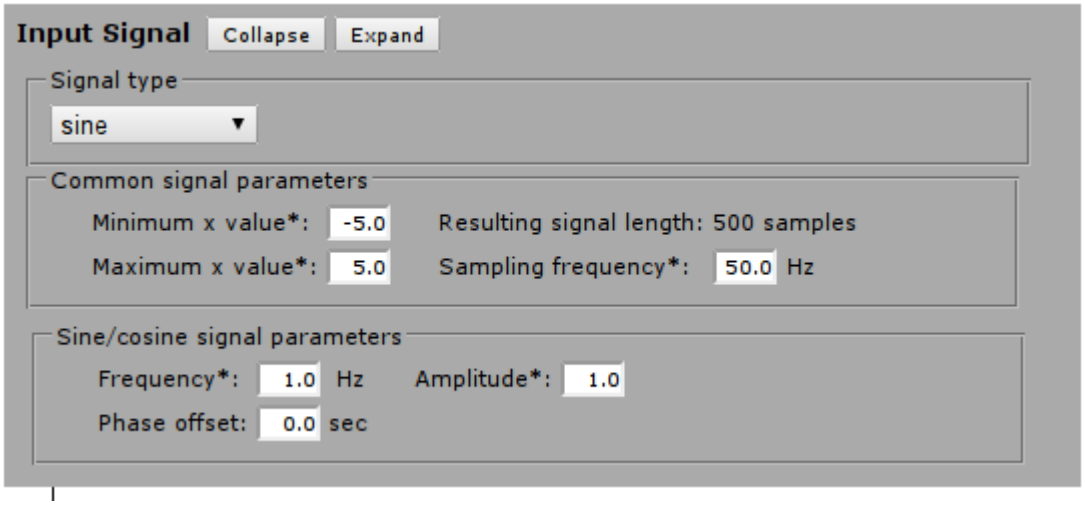

Figure 4.12. *Input signal block querying parameter for sinusoidal signal*

Based on the signal type selection, the group querying the signal-specific parameters is shown, whereas all the remaining groups are hidden. If sine or cosine signal types are selected, input fields for signal frequency, amplitude and phase offset are displayed. Selecting boxcar as signal type results in displaying input fields for signal amplitude and width in samples, where the signal exhibits the inputted amplitude. Sinc signals need as a parameter the frequency, the amplitude and the phase offset. The Gaussian signals are created using mean μ and standard deviation σ. Finally, for constant signals only the value is queried. An example of the input signal block querying parameters for a sinusoidal signal is shown in figure 4.12. and the underlying data model is shown in figure A.5.

4.7. Model for Creating Exercises

Based on the conceptual and technical considerations in sections 3.3., 4.6. and 4.2., in this section, a model of the workflow for creating learning units is derived. This process is illustrated by figure A.14.

Before a learning units can be created, its learning outcomes are defined. As mentioned in chapter 1., a powerful tool to achieve this is to define questions, which the student has to answer in order to see whether she fully understands the subject to be taught by this unit.

Since the learning platform is based on the Model-View-Controller pattern, after the learning goals and the question sheet have been defined, the data model[22] is created. It defines all input and output parameters of the learning unit. The data model is then programmed into Java classes under the *fi.frohling.learningplatform.datamodel* package. In order to separate the XHTML page form data from the MATLAB® input parameters and

[22]The data model describes the domain of the application.

the results, data models are created for each of those. The results models must implement the *ResultsModel* interface in order to achieve a unified calling interface.

As a next step, based on the data model, the building blocks of the user interface are identified. Those are either already existing UI components or a component, which is created from scratch.

After that, the back end functionality is created. The first step to achieve this is to program, test and compile the MATLAB® function. For calling a MATLAB® function, a data access object (DAO) is created under the *fi.frohling.learningplatform.dao* package. As with the results data model, in order to have a unified calling interface, all DAO classes must implement the *MatlabDao* interface.

After the DAO is created, the controller class, which calls the DAO object, is programmed. It is worth noticing, that an object of the controller class owns an object of the DAO class as well as an object of its corresponding results model. The controller provides a method, which is linked as an action to a command button in the user interface. The task of the controller's method is to create a new object of its corresponding MATLAB parameter class, assigns this object to the DAO object and calls the *perform()* method of the DAO object.

4.8. The Introductory Signal Processing Course

In this section, the exercises, of which the Introductory Signal Processing course is composed of, are introduced. After selecting the course from the course listing, the student gets presented with a list of learning units associated to the course. The course is composed of the following exercises: Fast Fourier transform, Convolution, Sampling and Quantization, Windowing, FIR Filtering and Convolution theorem.

The Fast Fourier Transform Exercise The purpose of this exercise is to give the student the possibility to study how different signals behave in frequency domain. Thus, the exercises need to consist of an input signal block, a Fourier transform block and two charts showing the generated input signal and its Fourier transform.

The Convolution Exercise This exercise demonstrates the principles of convolution. This requires to place two input signal blocks and two charts displaying the generated input signals. These two signals are fed into the convolution block. The result of this operation is shown in another chart on the right of the page.

The data flow starts after all required inputs are made and the Process button has been pressed. The *process()* function of the managed bean[23] of class *ConvolutionController* is called. It creates an object of *MatlabConvolutionParameterModel*[24], feeds it into the

[23] Java Beans are introduced in section 2.5.2.

[24] Objects of this class prepare data to be fed in parameters to the deployed MATLAB® function.

ConvolutionMatlabDao object and calls the DAO's *perform()* method. The DAO calls the convolution MATLAB® function, unpacks the resulting data needed for the charts and creates an object of *MatlabConvolutionResultsModel*. This object is used by the view to display the charts.

The Sampling and Quantization Exercise Sampling refers to discretization of a continuous signal along the time axis, whereas quantization refers to discretization of a signal along the y axis. Both method are needed to be performed for an analog-to-digital conversion.

Also for this exercise, an input signal is needed and thus the UI has to be equipped with an input signal block. The sampler block collects the sampling frequency used for this exercise and the quantization block queries the resolution of the quantized signal in bits. Charts are used to display the generated input signal and its Fourier transform, the sampled signal and its Fourier transform as well as the quantized signal. Due to the developed framework, the flow of data principally works the same way as with the Fourier transform and the convolution exercise except that objects of the classes used for the sampling and quantization exercise are used.

The Windowing Exercise Signals encountered in the real world last infinitive time. Thus, for studying and processing signals in a computer in digital domain, signals need to be cut of for a specific amount of time. If the signal is simply cut of, the Gibbs phenomena occurs due to the sharp transition. Thus, windows have been introduced in order to alleviate the distortions caused by the Gibbs phenomena.

Since the exercise allows to study the effects of windowing on a signal, the input signal block is needed in this exercise as well. Additionally, the windowing block, which queries the window type and its length from the user, has to be included into the exercise user interface.

After processing, charts for the input signal and its Fourier transform, the window function, its Fourier transform and the resulting windowed function and its Fourier transform are displayed. Also for this exercise, the data flow is principally the same due to the developed framework.

The Convolution Theorem Exercise The convolution theorem is used to save computational power when conducting convolution. In mathematical terms, the convolution theorem, is expressed as:

$$z = x * y = F^{-1}(F(x)F(y)) \tag{1}$$

where F denotes the Fourier transform and $*$ denotes convolution. Since convolution has a computational complexity of $O(n^2)$ and FFT of $O(n \cdot logn)$, applying the convolution theorem has a significant computational advantage.

As with the convolution exercise, two input signals and thus two input signal blocks are needed. To depict the convolution theorem, a convolution block, three FFT blocks[25] and a block for point-wise matrix multiplication are needed.

Charts display the input signal, the convolved and the signal, that has been processed using the convolution theorem. Also, a chart showing the residual signal and a text block showing the summed mean-squared residual is placed into the block diagram of the exercise. Again since the data flow is determined by the developed framework, except that the instantiated classes are suitable for the convolution theorem exercise.

The FIR Filtering Exercise Another exercise, which makes use of the block structure of the user interface, is the FIR filtering exercise. This exercise serves to demonstrate the basic properties of low- and highpass filtering.

Here, an input signal is fed into an array of multiplicator blocks. The number of multiplicator blocks, which resembles the filter order, is configured on top of the page. The results are finally summed up and the resulting signal is shown in a chart. Also the generated input signal is shown in another chart. Internally, the backing data model of the UI is composed of an array of Multiplicator class instances[26], an input signal object.

[25] Here, two FFT blocks for the forward FT are needed and one FFT block for the inverse operation are needed.

[26] The multiplicator objects themselves encapsulate the multiplicator as a double value.

5. DISCUSSION OF THE RESULTS

This chapter sums up and discusses the results, which were obtained by the previous chapter. The results are compared with the existing approaches from chapter 1.

As mentioned in section 3.1., the legacy Invocom learning platform used the discontinued MATLAB® WebServer toolbox to compute and visualize mathematic calculations. The here developed software, however, uses MATLAB® Builder JA, which offers the possibility to deploy MATLAB® functions to a Java EE application server. Such Java EE applications allow the deployment of Java code, which generates HTML markup. MATLAB® Builder JA offers, compared to the legacy MATLAB® WebServer, better integration into distributed software systems. Additionally due to the fact, that the CGI interface is not used anymore, security and performance have been improved.

One of the goals of this project is to create a remote laboratory. Such systems have been briefly introduced in chapter 1. Remote laboratories are very beneficial to students as they allow for hands-on-experience on the topic with inputs and parameters, which they are able to choose freely. The here developed block layout with the parameter input directly inside the blocks of the user interface allows to grasp the main elements of the demonstration unit easily.

This also leads to student-centeredness. As described in section 3.3., the learning platform has been designed and developed with the student as the main actor in mind. As mentioned earlier, students are able to choose all parameters of the subject to be studied freely, so that they get to experiment with the matter rather than studying dry mathematical formulas from the blackboard or the lecture slides.

The concept of nonlinearity in learning is realized by adding a “Related Content” section to the exercise page. Into that section, links to internal and external resources related to the topic of the exercise, are inserted. This motivates the student to stroll through the learning platform or the WWW finding information presented from different points of view.

As the student plays the central role in this learning platform, her learning is fostered towards the clearly defined learning goals by offering the possibility to attach a question sheet to the exercise. This question sheet is printable, which allows the learner to individually take notes on the sheet. On the other side, the questions allow to monitor the learning process of the students by offering a possibility to collect the sheets and to evaluate them. Such question sheets also offer the possibility to reduce exam pressure by allowing the students to collect bonus points from those exercises. The objective of the question sheet

is to guide the student towards the learning goals. This is refereed to as questions- and test driven model. On the other side, forcing students to answer the questions on those sheets is possible with this learning platform as well. Such a force is id est introduced by requiring the student to work through the question sheets in order to pass a course. This is refereed to as task-driven model. Such a model is useful, if an examination can be skipped. This should, however, be well thought of. Instead, it is possible to extend the course over another teaching period or to lower the amount of ECTS credits awarded for the successful completion of the course in question.

As the main result of this thesis project, a MVC software architecture has been developed, which allows to create and integrate reusable user interface components according to the building block principle. The challenge of making MVC to work over the WWW has been overcome by using the JavaServer Faces framework.

The problem of integrating the deployed MATLAB® functions into the learning platforms, which requires much custom Java code to be written, has been separated from the user interface code by developing DAOs for the MATLAB® functions. If another data source is desired later in the future, the UI code does not need to be changed, but only the DAO class has to be developed newly. This allows to easily integrate another numeric computation system like GNU Octave or SciLab if MATLAB® ceases its Builder JA product or if the teaching is switched to another numeric computation system.

Apart from visualizing the problem setting of the exercise using a building block UI layout, results also need to be visualized in a meaningful way. This requires charts, which are interactively zoom- and panable. MATLAB® Builder JA ships WebFigures, a framework to show ordinary MATLAB® figures on web pages. These, however, lack interactivity as their zoom functionality is not capable of zooming along axes and axes disappear if zoomed. Thus, another solution has been sought and found as JavaScript chart libraries. These allow charts to be zoomed and paned along axes, add crosshair navigation aids including values at the current crosshair position to be shown in the legend. In contrast to MATLAB® WebFigures, JavaScript-based chart libraries do not cause network traffic upon zooming or panning due to the fact that the chart logic and the data is transferred entirely into the browser. As all components of MATLAB® are proprietary and thus closed-source, the shortcomings of MATLAB® WebFigures cannot be fixed by the open-source software developer community. On the other side, since the JavaScript chart libraries are open-source, bugs are fixed quickly and new features are added if requested by the user community. Since one of the requirements was to have a chart configuration as close to MATLAB®'s figure and plot functions, an output parameter as a MATLAB® struct has been added, where the creator of the MATLAB® function has the possibility to configure the appearance of the JavaScript chart. Since the underlying technologies differ much, it is clear, that the possible usable properties are different from the ones MATLAB® uses.

In contrast to the existing, legacy Invocom learning platform, the layout of the new platform's blocks is not as good as with the Invocom platform. This is due to the fact, that the new platform's blocks are laid out using HTML tables. This leads to an adaption of the table columns width according to the block, which takes the most horizontal space. If the difference of the blocks widths is huge, this leads to arrows not starting or ending at the blocks edges. This problem has been overcome by the Invocom platform by implementing a pixel-exact layout. Such a layout, however, does not allow for exercise pages, which are assembled out of those building blocks using JavaServer Faces's *<ui:include>* mechanism.

Another possibility mentioned in chapter 1. is to integrate this learning platform into existing systems like Moodle. Such an approach requires the programmer to familiarize herself with such platforms. Additionally, if the teaching institution decides to switch to another system, huge efforts are needed to port the learning platform to the new system. Thus, this learning platform has been implemented as a stand-alone system. Nowadays, the possibilities offered by cloud storage are promising as well. This pushes the data and responsibility from the teaching institution to an external service provider. The hardware and service maintenance costs are lowered. In case of errors or software updates, however, the external service provider has to be contacted and asked for assistance. This introduces dependency. Also the load on the servers owned by the teaching institution caused by the learning platform is rather low. Thus, the learning platform can be installed on a machine, which is already used by other services. Thus, there are no significant benefits of outsourcing the learning platform to a cloud provider.

In chapter 1., also the concept of cooperative learning has been introduced. This concept is promising in that sense, that individual understanding of the topic by the students in the learning group is combined. This fosters discussions on the topic beneath the students and helps them to understand the subject taught using the other group members. At Tampere University of Technology, solving exercises to be graded in groups is not desired. Thus, the learning platform has been developed with the individual student in mind. It is, however, possible to adapt the question sheets to foster cooperative learning.

6. CONCLUSIONS

In this chapter, the results are summed up and their importance as well recommendations for their practical application are discussed. Furthermore, suggestions for future research based on this thesis are given.

This thesis project has been an innovative research and development project in the field of applied software development. Starting from the already existing learning platforms, possible learning strategies useful for the needs of newly developed platform, have been taken into account to create use cases. These use cases have been realized by a JavaServer Faces based web application prototype. Based on this prototype, the software architecture and a model for creating more exercises have been developed.

In the beginning of this project, a need for an interactive, easy-to-use and easy-to-extend e-learning platform for signal processing and communication theory students following the hands-on and question-driven learning principles has been identified. The use cases developed from this need define, that the student shall get the possibility to freely generate input signals and configure the parameters of the DSP algorithms. Input, intermediate and output signals shall be depicted as time series charts, which shall be zoom-able, pan-able and which can be saved to local disk in order to include them into exercise reports. Exercise questions are attached to the exercise page as a printable document. This allows students to individually make notes in written form on the printout, but also serves as a template for the exercise report. The questions are created based on the learning goals of the exercise.

The interesting project results from the teacher's point of view is the model on how to add new exercises to the learning platform. This is an important thesis project outcome as it allows teachers to create new exercises in a framework, which requires less Java and JavaScript coding, but lets teachers perform most of the work using HTML and MATLAB®[1]. Limiting the programming language requirement to HTML and MATLAB® allows teachers from the departments of signal processing and electronics to create new exercises, because in these departments a high proficiency in HTML and MATLAB® is present, but the Java and JavaScript proficiencies are rather lower in these departments. If the idea of integrating MATLAB® into this learning platform is thought further, researchers can get the possibility to demonstrate the results of their research using the learning platform. All it needs to be done is to create the MATLAB® function and the

[1]The complex mathematics are formulated in MATLAB® functions, which are used by the learning platform.

presentation page along with some Java code wiring everything up as shown in section 4.7.

From the software engineering point of view, the software architecture, which has been developed throughout this project, is of interest. It has been shown, how MATLAB® functions are integrated into a modern web application using state of the art JavaServer Faces, jQuery and Highcharts front end technologies. It has also been shown once more, that the well-known MVC pattern for UI application development is easily applicable for web applications too. In the learning platform, the MVC pattern allowed to clearly separate presentation matters from the data model and the actions needed to perform the whole task of retrieving the user-supplied parameters, running the DSP algorithm and presenting the results in a meaningful way to the user. Below this MVC layer, a data access layer has been deployed. Its responsibility is to abstract from the actions required to interact with the deployed MATLAB® functions. Since interaction with this layer happens through an interface, over which the parameters needed for the math function call are passed, it is possible to realize the mathematic functions using another technology such as GNU Octave, SciLab or as a Java numerical computing library.

The current version of this learning platform is a stand-alone distributed software system, which requires to be installed on a server maintained by the teaching institute. On the other side, this makes the teaching institute independent from third-party providers, such as cloud operators. Since the learning platform is not integrated into a framework such as Moodle, the teaching institute is free to change its general teaching framework software without the need to modify the learning platform developed in this project.

The current version of the learning platform, which has resulted from this thesis project, implements the work with one dimensional signals. Future projects have, based on this research, the possibility to investigate how the learning platform can be extended for two dimensional signals such as images and three dimensional signals such as videos. This allows the student to study how the DSP algorithms work with images or videos and what influences the results. Here, attention shall be payed to the data models and how the object-oriented design can be used to reuse the code already existing for the one dimensional case. Another case to study is the realization of simple vector- or matrix based exercise to study the DSP algorithms such as convolution, correlation and Fourier transform on these very simple examples. In this case, it is of special interest whether and how the existing MATLAB® functions can be used for these examples, too. Further research can also assess the improvements in learning caused by this learning platform. For this, a group of students takes a signal processing course with and a control group without the aid of the learning platform. Results of the examination and course feedback from the students shall be taken into account when performing the assessment.

References

[1] Y. Tang and A. Fan, "An integrated approach to self-regulated learning platform enhanced with Web 2.0 technology," *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*, pp. 236–239, 2011.

[2] L. Yin, B. Tan, Z. Li, and Y. Wang, "Web-based interactive learning platform of multimedia technology course," *2009 International Conference on Computational Intelligence and Software Engineering*, pp. 1–4, Dec. 2009.

[3] G. a. Krudysz and J. H. McClellan, "Web-based platform for problem-centered learning in DSP," *2011 Digital Signal Processing and Signal Processing Education Meeting (DSP/SPE)*, pp. 402–407, Jan. 2011.

[4] J. Yong, "Workflow-based e-learning platform," *Proceedings of the Ninth International Conference on Computer Supported Cooperative Work in Design, 2005.*, pp. 1002–1007 Vol. 2, 2005.

[5] F. Wen, J. Zhang, and Y. Tian, "Design and application of an e-learning platform for various learning groups," *Network Infrastructure and Digital Content, 2009. IC-NIDC 2009. IEEE International Conference on*, 2009.

[6] S. Jin, "Design of an online learning platform with Moodle," in *Computer Science Education (ICCSE), 2012 7th International Conference on*, July 2012, pp. 1710–1714.

[7] "Moodle.org: open-source community-based tools for learning," Moodle Ltd., 2013, [accessed on 14.03.2013]. [WWW]. Available at: http://www.moodle.org

[8] A. Asif, "Multimedia learning objects for digital signal processing in communications," *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03).*, vol. 3, pp. III–781–4, 2003.

[9] M. Llopis and F. Llopis, ".NET e-learning platform," *2008 Eighth IEEE International Conference on Advanced Learning Technologies*, pp. 664–665, 2008.

[10] I. Kastelan and M. Katona, "Students perspective of the embedded engineering learning platform - a case study in digital design," *2011 Proceedings of the 34th International Convention MIPRO (2011)*, pp. 1178–1182, 2011.

[11] D. Chandran and S. Kempegowda, "Hybrid e-learning platform based on cloud architecture model: A proposal," *2010 International Conference on Signal and Image Processing*, pp. 534–537, Dec. 2010.

[12] "edX," EdX, 2013, [accessed on 14.03.2013]. [WWW]. Available at: https://www.edx.org

[13] "Coursera," Coursera, 2013, [accessed on 14.03.2013]. [WWW]. Available at: https://www.coursera.org

[14] C. Giovannella, P. Selva, L. Serafini, and a. Bruni, "Conceptual learning assessment and content management in e-learning platform by means of conceptual maps," *Proceedings 3rd IEEE International Conference on Advanced Technologies*, pp. 400–401, 2003.

[15] F. Lerro, S. Marchisio, S. Martini, H. Massacesi, E. Perretta, a. Gimenez, N. Aimetti, and J. Oshiro, "Integration of an e-learning platform and a remote laboratory for the experimental training at distance in engineering education," *2012 9th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, pp. 1–5, Jul. 2012.

[16] "Mathematical markup language (MathML) version 3.0 2nd edition," W3C, [accessed on 18.04.2014]. [WWW]. Available at: http://www.w3.org/TR/MathML3/

[17] "MathJax," MathJax, [accessed on 18.04.2014]. [WWW]. Available at: http://www.mathjax.org/

[18] C. H. Chen and H. C. Lin, "A distance e-learning platform for signal analysis and measurement using FFT," *Computer Applications in Engineering Education*, vol. 19, no. 1, pp. 71–80, Mar. 2011. [WWW]. Available at: http://doi.wiley.com/10.1002/cae.20292

[19] W. Junfeng, "Application of interactive teaching in digital signal processing courses," *seiofbluemountain.com*, pp. 450–452, [accessed on 08.02.2013]. [WWW]. Available at: http://www.seiofbluemountain.com/upload/product/201006/2010jyhy03a14.pdf

[20] "MATLAB Builder JA - MathWorks Nordic," The MathWorks Inc, 2013. [WWW]. Available at: http://www.mathworks.se/products/javabuilder/

[21] B. Sturm and J. Gibson, "Signals and systems using MATLAB: an integrated suite of applications for exploring and teaching media signal processing," *Proceedings Frontiers in Education 35th Annual Conference (2005)*, pp. 21–26, 2005.

[22] G.-J. Hwang, C.-H. Wu, J. C. R. Tseng, and I. Huang, "Development of a ubiquitous learning platform based on a real-time help-seeking mechanism," *British Journal of Educational Technology*, vol. 42, no. 6, pp. 992–1002, Nov. 2011.

[23] G.-H. Hwang, B. Chen, H.-C. Chu, and Z. S. Cheng, "Development of a Web 2.0-based ubiquitous learning platform for schoolyard plant identification," *2012 IEEE Seventh International Conference on Wireless, Mobile and Ubiquitous Technology in Education*, pp. 259–263, Mar. 2012.

[24] "Internetworking technology handbook," Cisco Inc, [accessed on 09.11.2013]. [WWW]. Available at: http://docwiki.cisco.com/wiki/Internetworking_Basics#Open_Systems_Interconnection_Reference_Model

[25] "Hypertext transfer protocol – HTTP/1.1," Internet Engineering Task Force (IETF), [accessed on 09.11.2013]. [WWW]. Available at: http://www.ietf.org/rfc/rfc2616.txt

[26] S. R. Leonard Richardson, Mike Amundsen, *RESTful Web APIs*. O'Reilly Media, 2013.

[27] "XHTML™1.0 the extensible hypertext markup language (second edition)," W3C, [accessed on 01.04.2014]. [WWW]. Available at: http://www.w3.org/TR/xhtml1/

[28] "CSS specifications," W3C, [accessed on 01.04.2014]. [WWW]. Available at: http://www.w3.org/Style/CSS/specs

[29] "CSS reference," W3Schools, [accessed on 13.11.2013]. [WWW]. Available at: http://www.w3schools.com/cssref/default.asp

[30] "JavaScript and HTML DOM reference," W3Schools, [accessed on 13.11.2013]. [WWW]. Available at: http://www.w3schools.com/jsref/

[31] "AJAX tutorial," W3schools, [accessed on 13.11.2013]. [WWW]. Available at: http://www.w3schools.com/ajax/default.asp

[32] D. Sureau and xul.fr, "AJAX tutorial," [accessed on 29.08.2013]. [WWW]. Available at: http://www.xul.fr/en-xml-ajax.html

[33] "jQuery," The jQuery Foundation, [accessed on 13.11.2013]. [WWW]. Available at: http://jquery.com/

[34] "jQuery tutorial," W3Schools, [accessed on 13.11.2013]. [WWW]. Available at: http://www.w3schools.com/jquery/default.asp

[35] IOLA and O. Laursen, "Flot: Attractive JavaScript plotting for jQuery," [accessed on 29.08.2013]. [WWW]. Available at: http://www.flotcharts.org/

[36] "Tampere - wikipedia," [accessed on 13.11.2013]. [WWW]. Available at: http://en.wikipedia.org/wiki/Tampere

[37] "Highcharts - interactive JavaScript charts for your webpage," Highsoft, [accessed on 02.04.2014]. [WWW]. Available at: http://www.highcharts.com/

[38] "Highcharts API reference," Highsoft, [accessed on 02.04.2014]. [WWW]. Available at: http://api.highcharts.com/highcharts

[39] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view controller user interface paradigm in Smalltalk-80," *J. Object Oriented Program.*, vol. 1, no. 3, pp. 26–49, Aug. 1988. [WWW]. Available at: http://dl.acm.org/citation.cfm?id=50757.50759

[40] J. Deacon, "Model-view-controller (MVC) architecture," 2009, [accessed on 28.08.2013]. [WWW]. Available at: http://www.jdl.co.uk/briefings/MVC.pdf

[41] A. Leff and J. Rayfield, "Web-application development using the Model/View/Controller design pattern," In: Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International, 2001, pp. 118–127.

[42] "MATLAB Builder JA 2 user's guide," The MathWorks, [accessed on 29.08.2013]. [WWW]. Available at: http://www.scribd.com/doc/52391015/JA-builder

[43] J. W. Eaton, "GNU Octave," [accessed on 07.03.2014]. [WWW]. Available at: http://www.gnu.org/software/octave/

[44] K. Hansen, "JavaOctave: Wiki: Home - project kenai," [accessed on 07.03.2014]. [WWW]. Available at: https://kenai.com/projects/javaoctave/pages/Home

[45] "The W3C markup validation service," W3C, [accessed on 07.03.2014]. [WWW]. Available at: http://validator.w3.org

[46] "JavaServer Faces community," [accessed on 07.03.2014]. [WWW]. Available at: https://javaserverfaces.java.net

[47] "Oracle mojarra JavaServer Faces," [accessed on 29.08.2013]. [WWW]. Available at: http://javaserverfaces.java.net/

[48] "Glassfish server," Oracle, [accessed on 29.08.2013]. [WWW]. Available at: http://glassfish.java.net/

[49] "MyFaces - welcome to the apache MyFaces project," http://myfaces.apache.org/, Apache Software Foundation, [accessed on 29.08.2013]. [WWW]. Available at: http://myfaces.apache.org/

[50] H. Bergsten, "Improving JSF by dumping JSP," http://www.onjava.com/pub/a/onjava/2004/06/09/jsf.html, September 2004, [accessed on 28.8.2013]. [WWW]. Available at: http://www.onjava.com/pub/a/onjava/2004/06/09/jsf.html

[51] "JSF 2.1 view declaration language," Oracle, [accessed on 28.8.2013]. [WWW]. Available at: http://docs.oracle.com/javaee/6/javaserverfaces/2.1/docs/vdldocs/facelets/

[52] "JavaBeans spec," Oracle, [accessed on 07.03.2014]. [WWW]. Available at: http://www.oracle.com/technetwork/java/javase/documentation/spec-136004.html

[53] "Expression language - the Java EE 6 tutorial," Oracle, [accessed on 28.08.2013]. [WWW]. Available at: http://docs.oracle.com/javaee/6/tutorial/doc/gjddd.html

[54] "JSF life cycle," tutorialspoint, [accessed on 02.11.2013]. [WWW]. Available at: http://www.tutorialspoint.com/jsf/jsf_life_cycle.htm

[55] "FacesContext (javaserver faces API (2.0))," Oracle, [accessed on 02.11.2013]. [WWW]. Available at: http://docs.oracle.com/cd/E17802_01/j2ee/javaee/javaserverfaces/2.0/docs/api/javax/faces/context/FacesContext.html

[56] "JSF - page navigation," tutorialspoint, [accessed on 02.11.2013]. [WWW]. Available at: http://www.tutorialspoint.com/jsf/jsf_page_navigation.htm

[57] "Apache TomEE," The Apache Software Foundation, [accessed on 06.11.2013]. [WWW]. Available at: http://tomee.apache.org/apache-tomee.html

[58] C. Humble, "Oracle officially launches Java EE 7 and Glassfish 4 today," 2013, [accessed on 22.07.2013]. [WWW]. Available at: http://www.infoq.com/news/2013/06/ee7-launch

[59] "JBoss enterprise application platform features," Red Hat, 2013, [accessed on 22.07.2013]. [WWW]. Available at: http://www.redhat.com/resourcelibrary/articles/jboss-enterprise-application-platform-features

[60] "Language requirements," Tampere University of Technology, [accessed on 07.03.2013]. [WWW]. Available at: http://www.tut.fi/admissions/applying/master-s-programmes/language-requirements

[61] "Study guide for degree students," Tampere University of Technology, 2012-2013, [accessed on 07.03.2013]. [WWW]. Available at: http://ttyhakuinfo2.mediacabinet.fi/hakuinfo/filebank/1217-TTY_degree_students_2012-13_www.pdf

[62] "Options for deploying MATLAB applications via the Web - MathWorks Deutschland," MathWorks Deutschland, [accessed on 18.04.2014]. [WWW]. Available at: http://www.mathworks.de/products/new_products/webserver_discontinued.html?s_cid=r2006b_webserver

[63] "IntelliJ IDEA - the best Java and Polyglot IDE," JetBrains, [accessed on 18.04.2014]. [WWW]. Available at: http://www.jetbrains.com/idea/

[64] "Welcome to NetBeans," Oracle Corporation, [accessed on 18.04.2014]. [WWW]. Available at: https://netbeans.org/

[65] "Eclipse - the Eclipse open source community website," The Eclipse Foundation, [accessed on 29.08.2013]. [WWW]. Available at: http://www.eclipse.org

[66] "Eclipse - the Eclipse foundation open source community website," The Eclipse Foundation, 2013, [accessed on 12.04.2013]. [WWW]. Available at: http://http://www.eclipse.org/

[67] "Apache Tomcat - Welcome!" The Apache Software Foundation, 2013, [accessed on 11.04.2013]. [WWW]. Available at: http://tomcat.apache.org/

[68] "Home - Pencil project," Evolus, [accessed on 18.04.2014]. [WWW]. Available at: http://pencil.evolus.vn/

APPENDIX 1: ADDITIONAL FIGURES

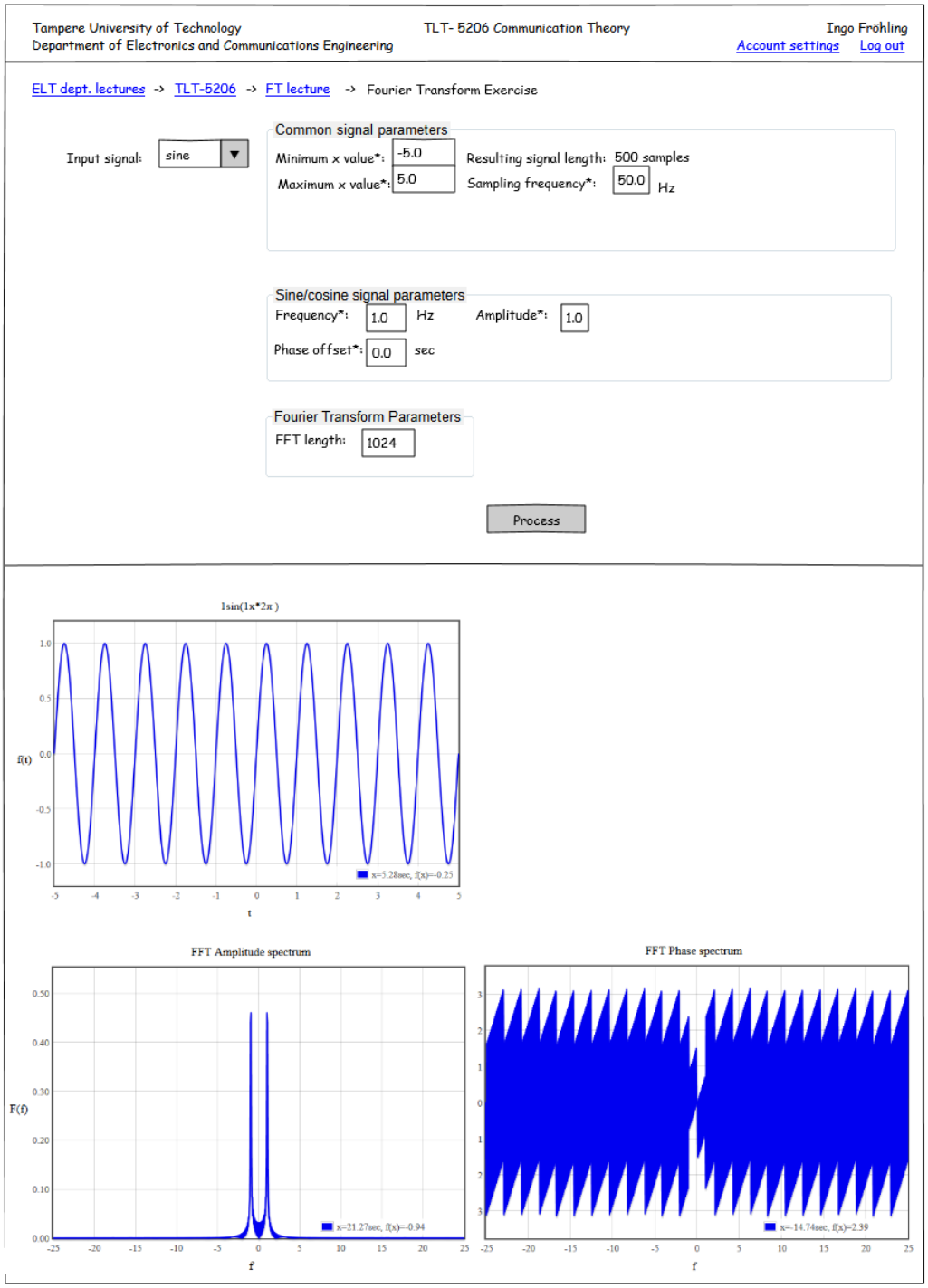

Figure A.1. *JSF FFT User interface*

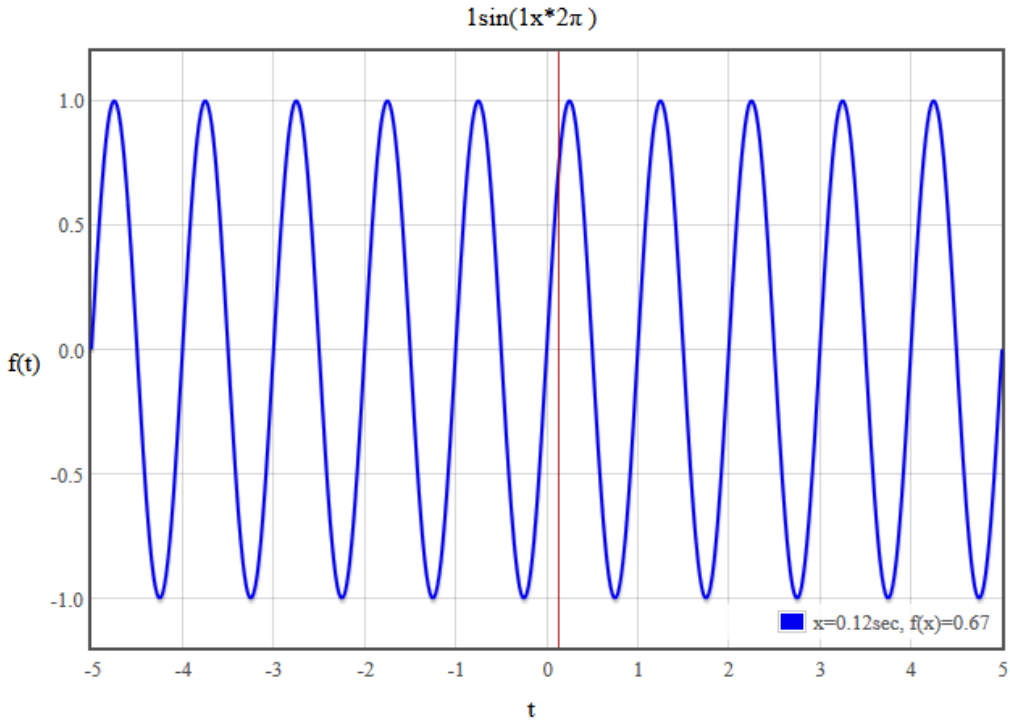

Figure A.2. *Chart showing Generated input signal*

```
// Retrieve the input values from the form
var minimumX = document.getElementById('minimumX').value;
var maximumX = document.getElementById('maximumX').value;
var fs = document.getElementById('fs').value;
var signalLengthElement =
    document.getElementById('signalLength');

// Prepare the AJAX request
var request = '/signallength?minimumX='+minimumX+'&maximumX='+
   maximumX+'&fs='+fs;
var ajaxRequest = new XmlHttpRequest();

// Define the action to be performed
// when the request successfully completes
ajaxRequest.onreadystatechange = function() {
    if (ajaxRequest.readystate === 4 &&
        ajaxRequest.status === 200) {
        // Assign the response of the AJAX request
        // as the text of the signalLength DOM element
        signalLengthElement.innerHTML =
            ajaxRequest.responseText;
    }
};

// Send the request to the server
ajaxRequest.open('GET', request);
ajaxRequest.send();
```

Figure A.3. *Plain JavaScript code of an AJAX request*

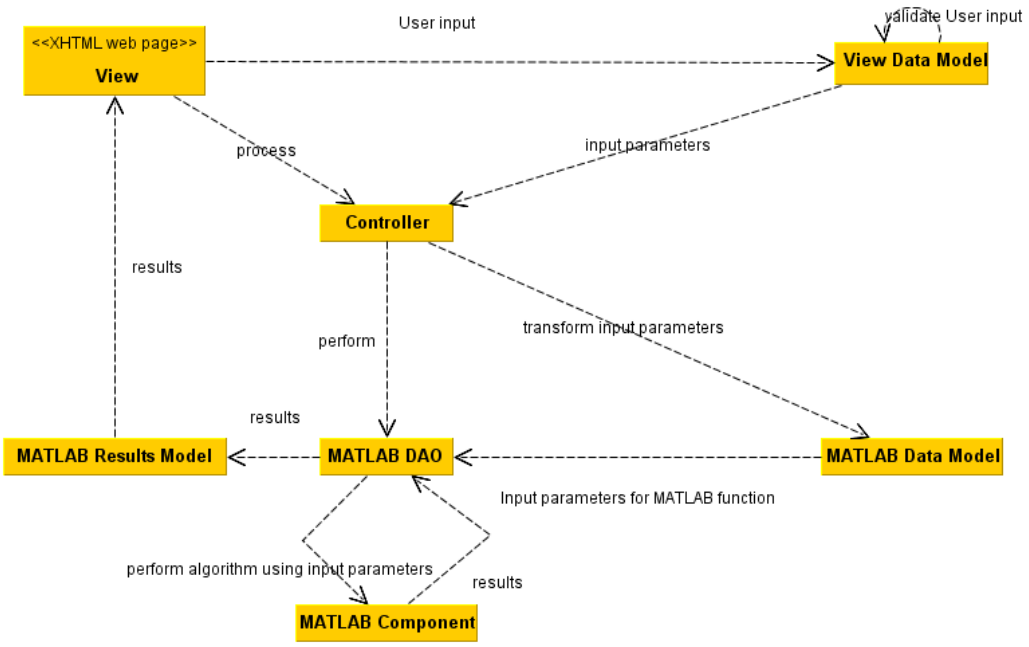

Figure A.4. *General software architecture*

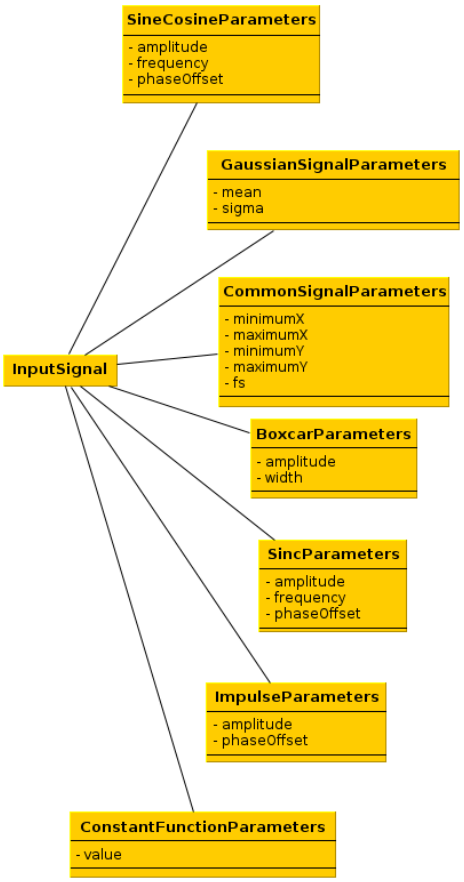

Figure A.5. *Data model of input signal block*

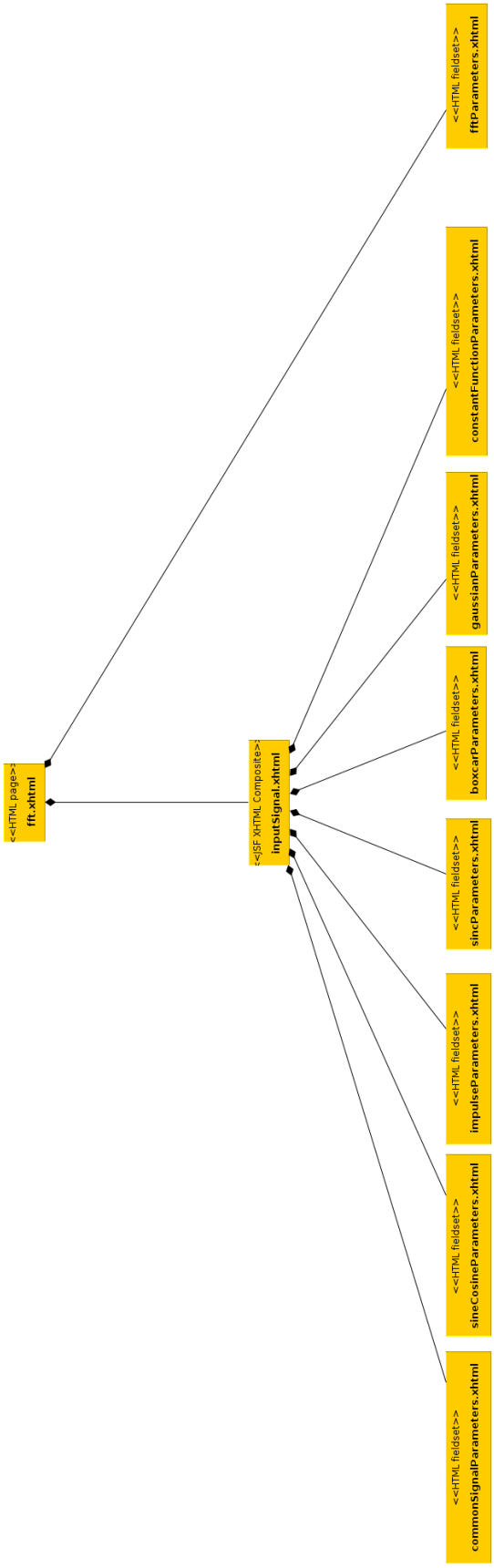

Figure A.6. *FFT view components*

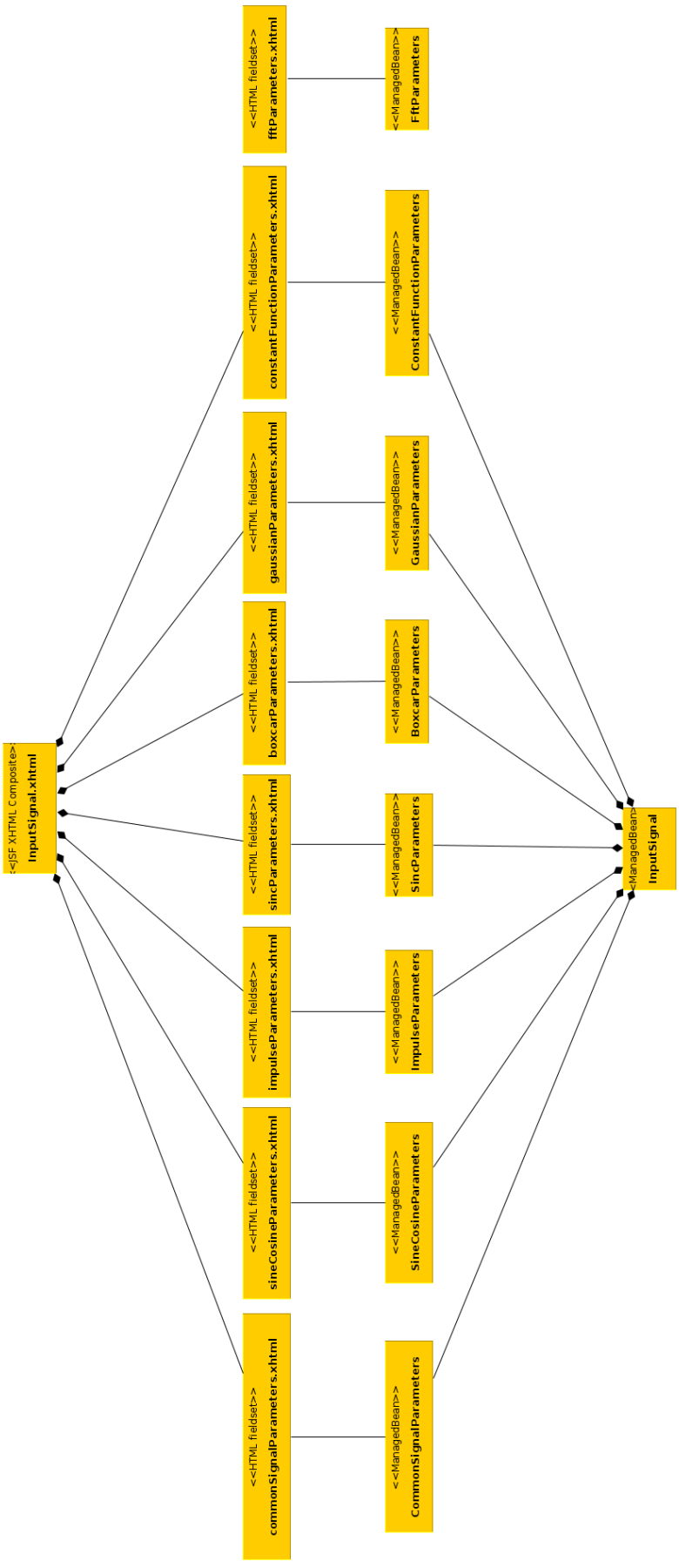

Figure A.7. *FFT view and view data model association*

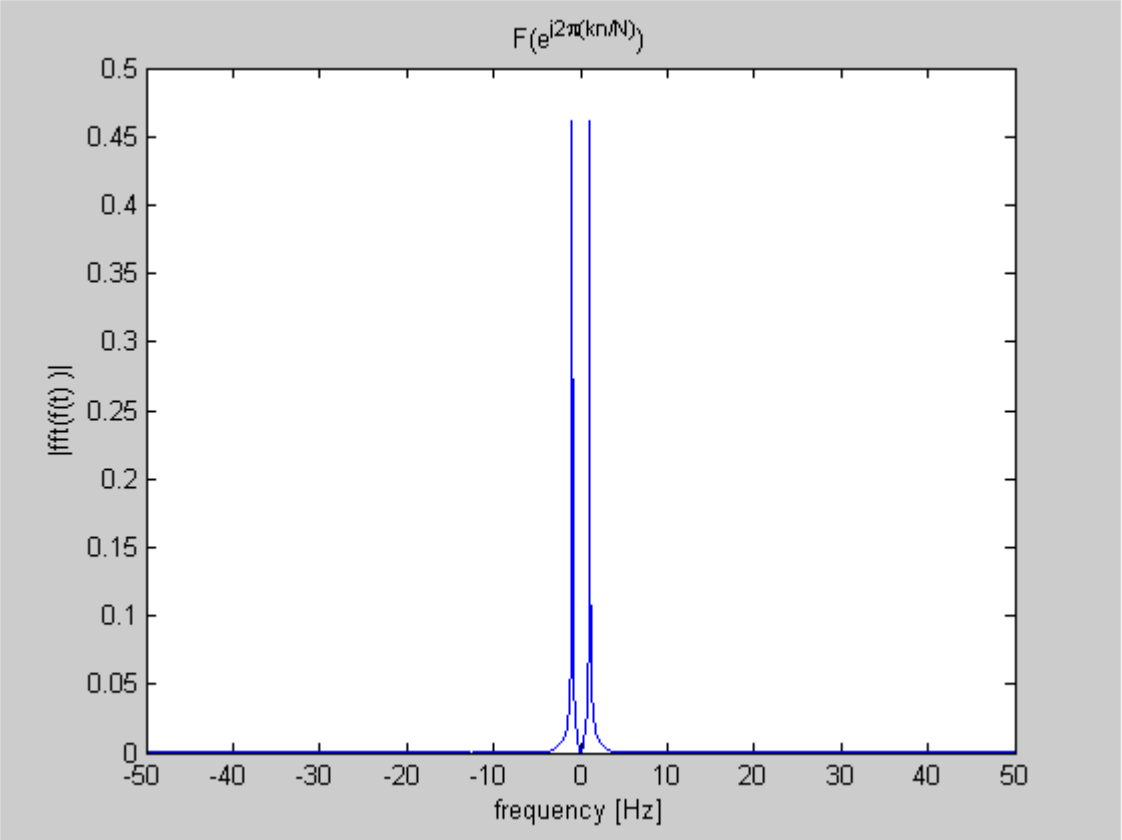

Figure A.8. *MATLAB Builder JA WebFigure*

Figure A.9. *MATLAB Builder JA WebFigure zoomed in*

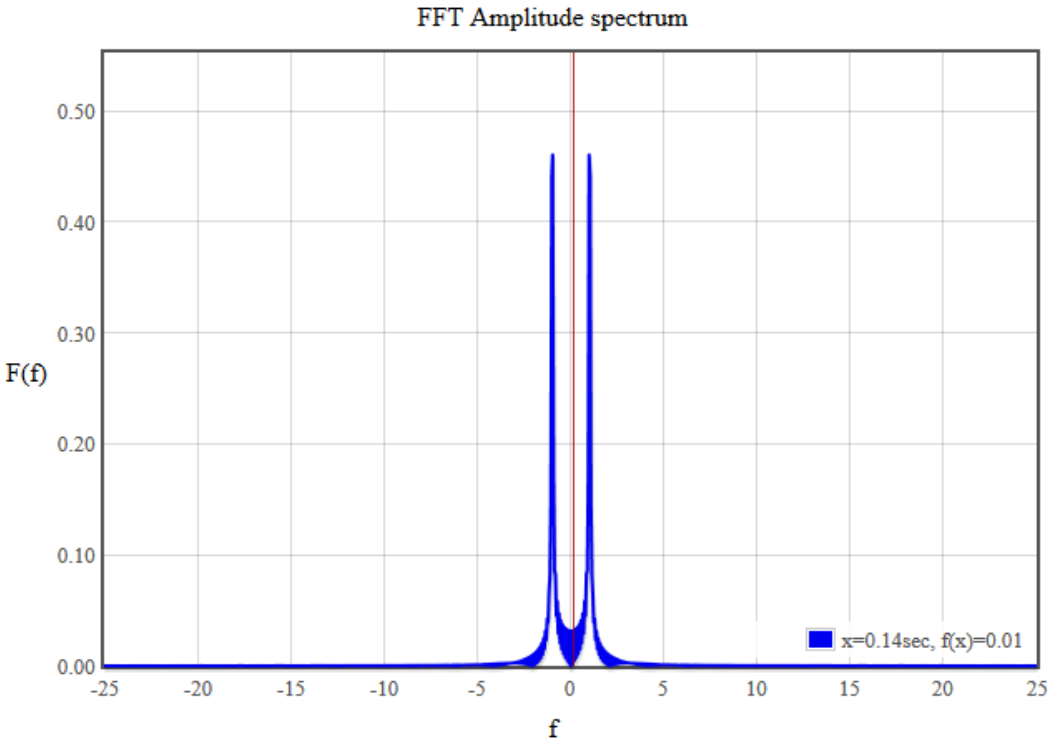

Figure A.10. *Flotchart*

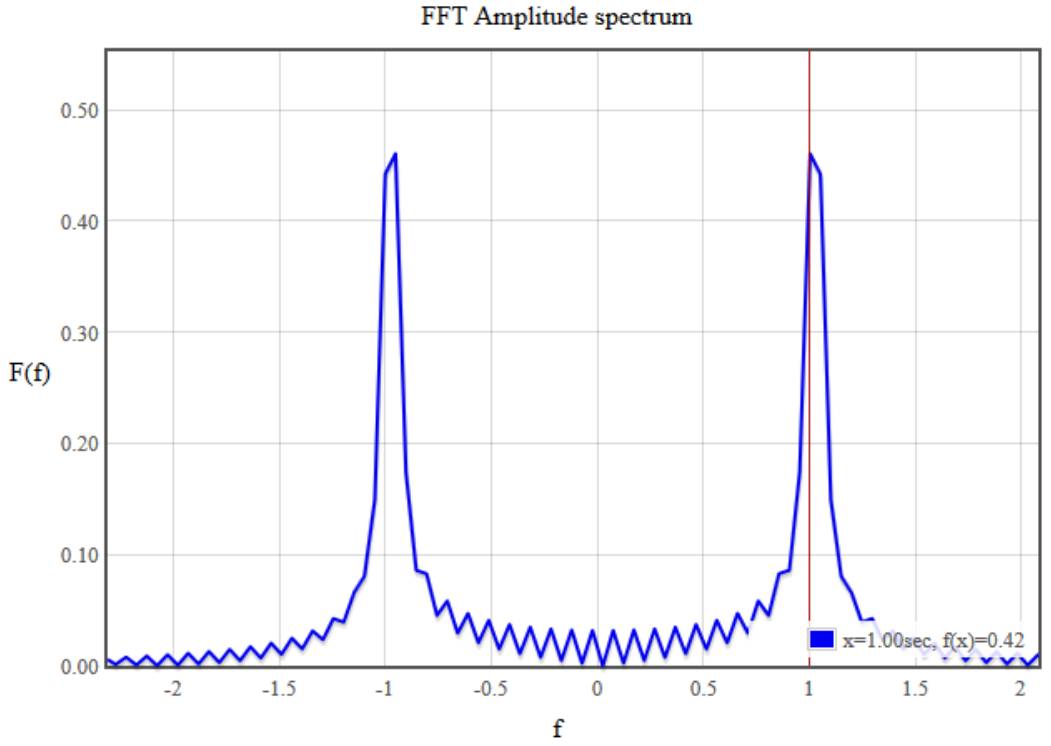

Figure A.11. *Flotchart zoomed in*

```
var flotPlotOptions = {
    grid: {
        clickable: false,
        hoverable: true,
        autoHighlight: false
    },
    xaxis: {
        zoomRange: [0.01, maximumXAxisVal-minimumXAxisVal],
        panRange: [minimumXAxisVal, maximumXAxisVal],
        ticks: tickFun
    },
    yaxis: {
        min: yAxisMin,
        max: yAxisMax,
        // do not zoom on the y-axis
        zoomRange: [maximumYAxisVal-minimumYAxisVal,
           maximumYAxisVal-minimumYAxisVal],
        panRange: [minimumYAxisVal-0.2, maximumYAxisVal+0.2]
    },
    zoom: {
        interactive: true
    },
    pan: {
        interactive: true
    },
    crosshair: {
        mode: "x"
    },
    legend: {
        position: 'se'
    }
};
// Create the plot
flotPlot = $("#container").plot(data, plotOptions);
```

Figure A.12. *Program listing showing Flotchart options used for the FFT demonstration page*

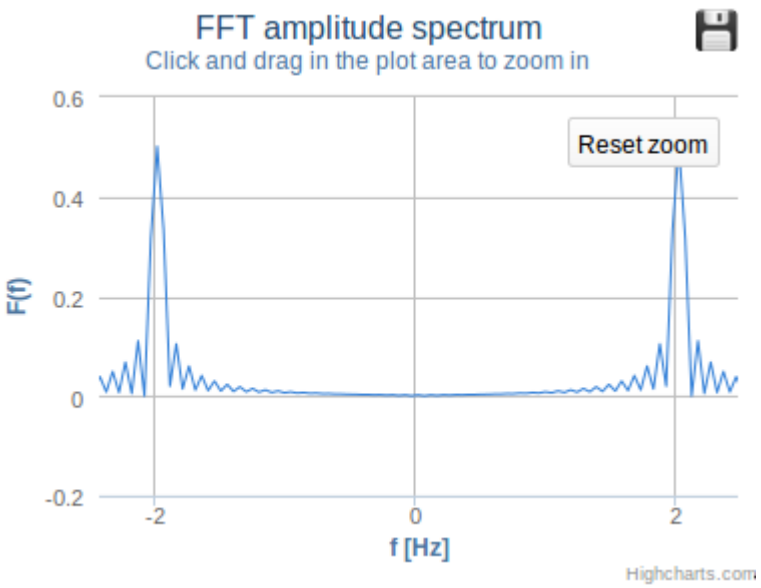

Figure A.13. *A chart generated by the JavaScript Highcharts library*

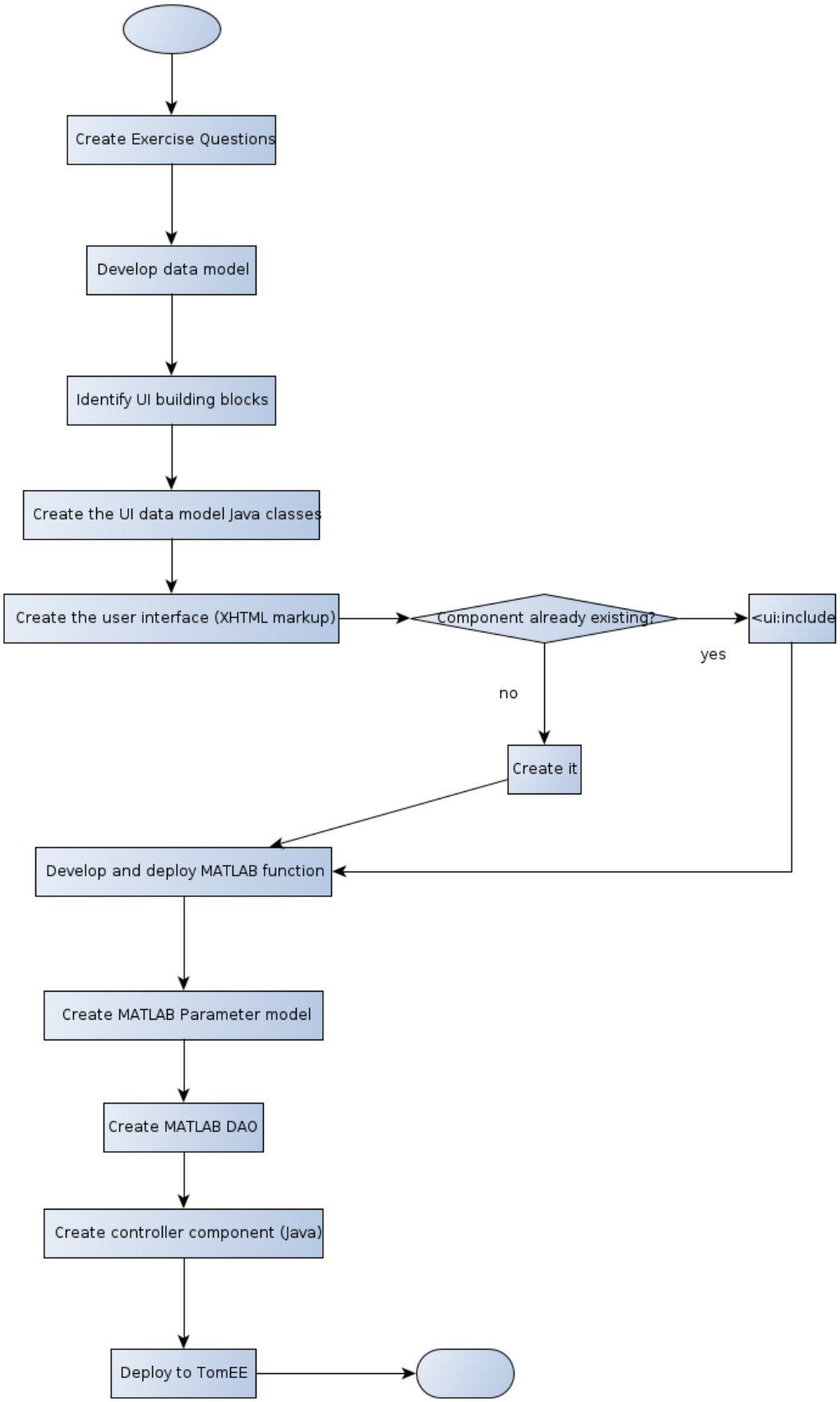

Figure A.14. *Workflow for creating exercise units*