GeneFisher2
an AJAX based implementation of GeneFisher

Bachelor Thesis
in Bioinformatics and Genome Research

by

Daniel Hagemeier

August 16, 2006

Revisors: Dipl.-Inf. Alexander Sczyrba
Prof. Dr. Robert Giegerich

CONTENTS

1 Motivation **2**

2 GeneFisher - Classical web-application for degenerate primer design **4**
2.1 Exploration of GeneFisher 4
2.2 Limitations of GeneFisher 8

3 AJAX - Basics **11**
3.1 AJAX - Basics 11
3.1.1 XMLHttpRequest 12
3.1.2 JavaScript and DOM 14
3.2 A simple AJAX example 15
3.3 AJAX - Disadvantages 16

4 GeneFisher2 - AJAX based Implementation **19**
4.1 GeneFisher2 - The Improvements 19
4.2 GeneFisher2 - Realisation of GeneFisher with AJAX 20
4.2.1 AJAX - Not always the right choice 23
4.2.2 GeneFisher2 - The Input Validation in Detail 24
4.3 Outlook 26

CHAPTER 1

MOTIVATION

One of the most important technologies in biology is the polymerase chain reaction (*PCR*). The reaction allows to generate manifold copies of a desired DNA fragment. Primers, short oligo nucleotides, mostly between 15-30 nucleotides long, play a major role in this reaction. These primers anneal at selected positions on a template DNA strand that contains the target DNA fragment. The primers frame the targeted fragment and build the starting point for the activity of the polymerase enzyme. Outgoing from the annealed primers this enzyme synthesises a new complementary copy of the template DNA strand. After some cycles determined by temperature changes that induce the primer annealing, the elongation by the polymerase and the denaturation, where the synthesised strands are separated from their template, the desired fragment is duplicated many times.

Specific Primers have to be designed before every new PCR experiment. Because of its versatility and reliability in primer design *GeneFisher* (4) is a very popular tool. *GeneFisher* calculates primers complementary to a single DNA sequence. It also implements an approach for finding a primer pair matching to a postulated DNA fragment or gene of a not yet sequenced organism.

Primer design is necessarily a highly interactive process, because primers must fullfill the expectations of the biologist running the PCR. The length of the amplified product, the GC content of the primers or the primer length are only a few examples of the parameters which need to be adapted to PCR experiments. *GeneFisher* gives biologists the possibility to arrange primers according to their requirements. The adjustment of many parameters is possible in *GeneFisher*. A disadvantage of such classical highly interactive web applications are pauses in the workflow between requests to the server and the upload of a new website to the browser. Users had to check states of server requests before they could go on with the application.

The task of this bachelor thesis was to keep the functionality of *GeneFisher* and to make the application more comfortable to users. The result is *GeneFisher2* a new implementation using *AJAX*. *AJAX*, short for *Asynchronous Java-*

Script and XML, allows to create interactive web applications with a desktop-application-like behaviour where pauses and state checks by hand are no longer necessary.

CHAPTER 2

GENEFISHER - CLASSICAL WEB-APPLICATION FOR DEGENERATE PRIMER DESIGN

2.1 Exploration of GeneFisher

To facilitate the change to an improved user friendliness, a closer look at *GeneFisher* was required. A general overview of *GeneFisher's* workflow had to be carried out first.

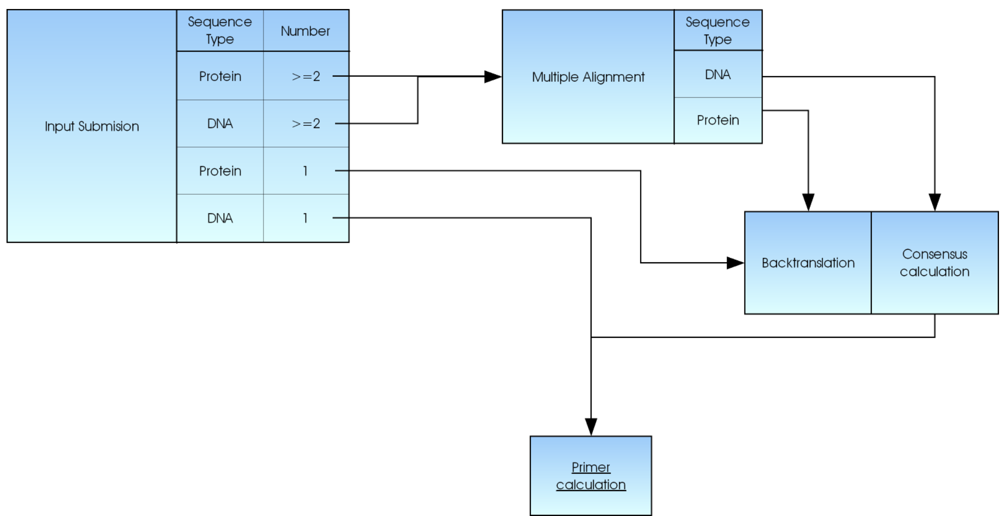

Figure 2.1: Logical Workflow of GeneFisher

One or more, DNA or protein sequences are the required input for the *Gene-Fisher* tool. The primer generation for a single DNA sequence, the template

strand for a PCR, is started directly after sequence submission. To amplify the sequence of a postulated template, *GeneFisher* supports the construction of degenerate primers. Therefore *GeneFisher* takes sequences of organisms closely related to the target as input to generate fitting primers. A multiple alignment of these sequences allows the establishment of a consensus sequence that contains the common information of all involved sequences. The consensus sequence is then handled like a single DNA or portein strand.

Degenerate primers fitting to such a consensus sequence are primers where some nucleotide positions are not specified clearly. For a PCR experiment this means that a mixture of primers showing a different nucleotide in this position is required. For the calculation of primers fitting to amino acid sequences additionally a backtranslation is required. A backtranslation means that the sequences are retranslated into their coding nucleotide sequences. Figure 2.1 shows all mentioned operations of *GeneFisher* in a general overview.

For the new implementation of *GeneFisher* the re-use of some of its binaries was favored. A data flow chart given in the dissertaion of SCHLEIERMACHER C.(5) was not sufficient to achieve this aim. Therefore an analysis of the dependencies between scripts and binaries was necessary. This "reverse engineering" allowed to get detailed knowledge about the *GeneFisher* program. The result is a detailed chart of the CGI scripts, the binaries and their interactions. An internal modularity of *GeneFisher* is determined by this examination. According to figure 2.2 the detailed workflow of *GeneFisher* will be described in the following section.

gf_submit

The `gf_submit` script is responsible for the sequence submission. The HTML page generated by `gf_submit`, offers two possibilities for it. A textarea that allows to paste selected sequences into it, and a file upload field. In this field, the URL of a locally saved files can be entered. Users execute the submission by clicking a button. The `gf_parseseqinput` script is started.

gf_parseseqinput

The submitted input is saved and checked by `readseq` which is executed by this script. The `readseq` output is used to classificate the input as amino acid or nucleotide sequence. `gf_parseseqinput` counts the submitted sequences and guesses by comparing the sequence lengths and searching for gaps if the submitted sequences are aligned or not. Depending on these results, the next executed script is determined.

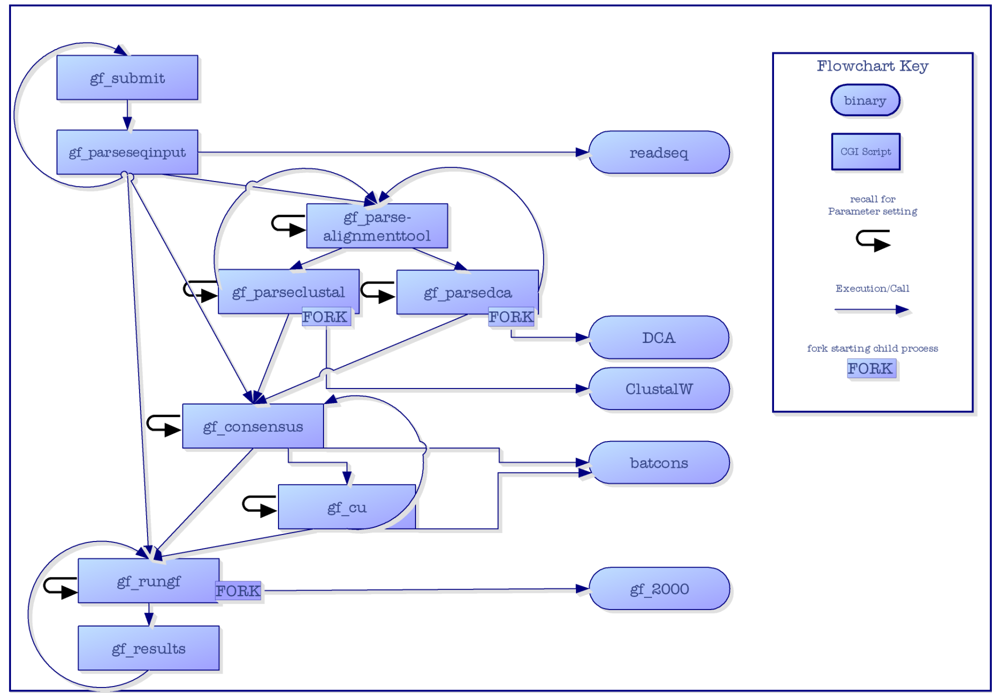

Figure 2.2: GeneFisher Flowchart

gf_parsealignmenttool

An unaligned input with more than one sequence is treated next by the `gf_-parsealignmenttool` script. On the first diplayed page the alignment tool can be selected. Users have the choice between *ClustalW* and *DCA*. In the next step the script executes either `gf_parsedca` or `gf_parseclustal` depending on the selected alignment algorithm on the page.

gf_parsedca and gf_parseclustal

Both scripts generate HTML sites allowing a parameter adaption for the calculation of a multiple alignment with the selected algorithm.

After the parameter settting the user submits them. The script then calls itself to assemble a query string with the input. A `Fork` command starts the alignment binary ClustalW or DCA in a child process. The main process continues with the generation of new HTML page. On this page, the users cooperation is required to avoid mistakes in the further GeneFisher workflow. A detailed description of these difficulties will be given in section 2.2.

The results of the alignment are presented on another page by button click. Users now can decide to continue and go to consensus calculation or to recalculate the alignment with different settings or a different algorithm. In case of a re-calculation `gf_parsealignmenttool` is executed again. The `gf_consensus` script is responsible for the consensus establishment.

gf_consensus

After submission of already aligned sequences or after an alignment in the GeneFisher workflow, a consensus sequence construction in `gf_consensus` is the next step. The HTML document provided by the script allows a parameter adaption for the consensus calculation. In case the input are amino acid sequences, the HTML page additionally offers two methods for a backtranslation to nucleic acids. These methods are the maximum redundancy method and the codon usage method. If the codon usage method for backtranslation is selected `gf_cu` is target of the sended HTTP request.

The maximum redundancy method is provided by the `BatCons` binary, which is also responsible for the consensus calculation. After the form submission by button click, the `BatCons` binary is executed with the appropriate parameters. `BatCons` evaluates the consensus sequence and performs a possibly required backtranslation. The results are treated like in `gf_parsedca` and `gf_parseclustal` and causes the same difficulties. For a necessary recalculation `gf_consensus` calls itself again.

gf_cu

The `gf_cu` script executes a Java applet. This Java applet allows the backtranslation of amino acid sequences to DNA sequences with the codon usage method. After backtranslation the script also executes the `BatCons` binary to get a consensus sequence.

gf_rungf

The `gf_rungf` script starts the primer calculation in the workflow. This script porvides an HTML page to set parameters for the primer selection. After the submission of possibly adapted parameters by the user, the `gf_2000` binary for primer calculation is started in a child process. The main process produces a page with a single button leading to the results.

gf_results

This script is last part in the workflow and responsible for the presentation of the calculated results, the primers. The beside the results, the page contains two buttons. One button leads back to gf_rungf to apply different primer settings. The other one executes gf_submit the initial *GeneFisher* script.

2.2 Limitations of GeneFisher

The interactive nature of *GeneFisher* is responsible for the complicated structure of this classical web application. In such a program many steps are necessary to reach the final result, because after every choice of the user a new page has to be generated.

No Parameter Validation

Maybe to keep the user from going through more pages *GeneFisher* makes no validation of the submitted parameters. This means entering letters instead of numbers (integer values), will not be recognized by the program.

The binaries DCA or ClustalW for example are executed without a previous validation of the entered parameters. In the protocols of these binaries warnings like "Bad option for /topdiags (must be integer)" are listed, but which user cares for this output in a long list of cryptical messages of a program?
In such a case the binaries produce a result but it is not determined by the parameters the user adapted.

The parameters for the BatCons binary are not validated either before the execution. The program sets its own standard parameters that are different to those given as suggestion in GeneFisher. Listing 2.1 shows a log file of BatCons executed with false parameters. The only hint that the entered parameters were not accepted by the program is the word corrected and a list of the used parameters. Again there is no explicit warning.

Another place of user interaction in *GeneFisher* is the page where the primer calculation parameters can be adapted. Here the execution of the gf_2000 binary with inappropriate parameters leads to no results in primer calculation. Again, there is no instruction for the users pointing at the wrong parameters.

Slight Sequence Validation

A check of the input sequences is initialised after submission of the input to gf_paresseqinput. The sequence validation is commited by readseq. This

Listing 2.1: Protocol of BatCons Binary

```
1
2  BiBiServ Tool Log
3  Use the Reload button to update.
4
5  *** batcons V1.0
6
7  Program Settings
8   Result file: /spool/genefisher/_1155292236_18515/alignment.cons
9   Unique (corrected): 50%
10  Noise: 0%
11  Min./Max. Length of 22 Sequence(s): 255/255
12
13 Autodetected sequence type is DNA (0% unambiguous aa).
14 TCTGCCTCCGTCCTTnnTTCTCACAGCAATGAATTTTGCAATCTGAACCCAAGTGAAAAAAAAnnnnnn...
```

validation is not very restrictive so that invalid sequences can get into the further process and lead to bad results.

Uncomfortable Handling

In *GeneFisher* the binaries `readseq`, `ClustalW`, `DCA`, `BatcCons` and `gf_2000` are executed by the appropriate CGI script. The results of the binaries are stored under a previously defined filename. A connection back to *GeneFisher's* scripts does not exist. This means that the invoked calculation is not traceable by *GeneFisher* itself. The users have to check the state of the calculation and their results manually.

Especially after the execution of `DCA`, `ClustalW` or the `BatCons` binary the cooperation of the users is required. Users have to click a button to open a separate new window. It contains the progress file of the calculation like the one in listing 2.1 that shows the output of `BatCons`. In these files users have to search for a certain output. The alignment binaries require the message "GDE-Alignment File created" and `BatCons` has to show the consensus sequence on the bottom of the displayed page. An empty page is not an unusual result, because the calculations often take some time. Then the users have to refresh the page showing the progress file manually by clicking on the reload button of their browser until the necessary answer of the executed binary appears. There is no automatic page refreshing. If the users do not follow these instructions they may invoke the next script too early. This leads to unsatisfying results later, because in the chain of scripts the results of every step are required to get a proper final result.

Generally, the handling of errors caused by wrong input like described earlier, or errors during the binary execution are not handled by *GeneFisher* directly. Users can ignore the error messages in the binary output. This also can lead to unsatisfying or no results in the worst case.

After every execution of the mentioned binaries the user has to click a button to see the results of the calculation. The demanded results are also displayed on a new page. This can be annoying when the results do not come up to the user`s expectations. The adaption of the parameters has to be done on the previous page. Especially a strict choice of parameters, which is often required for primers can lead to a multiple back and forth in the application.

A possibility to reduce or remove these mentioned "limitations" of GeneFisher is the adoption of techniques in recent web development. The AJAX technique is a promising approach to encourage a user-friendly interactive behaviour of web applications. In chapter 3 the basics of AJAX are discussed.

CHAPTER 3

AJAX - BASICS OF A NEW TECHNIQUE IN WEB DEVELOPMENT

3.1 AJAX - Basics

AJAX is a shorthand for Asynchronous Javascript and XML(1). The term *Web 2.0* is also associated with this technique used in recent web development. Web applications using AJAX have a lot of advantages regarding conventional [1] internet programs.

By using *AJAX* it is possible to leave the client server communication in the background of web applications unnoticeable to the user. Elements of HTML pages can be changed, adapted or added by using server data without loading a completely new page. This can reduce the server-load because only relevant parts of a page have to be calculated. Users get a faster feedback directly after a request. A desktop-application-like behaviour of web applications can be implemented in this way. It is not necessary to provide a program, like a Java-Applet, to all users to achieve this interactive behaviour, when using *AJAX*. In contrast to classical HTTP requests, the *AJAX* requests are asynchronous. This means that during a request, for example a server side validation of form entries, users can still work on the displayed HTML page.

The problems of *GeneFisher* listed in section 2.2 can be avoided with an AJAX based implementation. For example the back and forth from website to website to adjust parameters and watch the calculated result, is no longer necessary when using AJAX. It also allows to establish a connection to the server without an explicite user command. A simple click, move or keypress is sufficient. So a user input like entered or changed parameters can be validated by an adequate server side script or program immediatly, without any further user activity.

[1] Conventional internet programs in this bachelor thesis are always ment to be HTML documents and CGI scripts interacting directly using HTTP like *GeneFisher* does. The considered applications do not use JavaScript for the communication with the server side.

Unbelievable that*AJAX* is no new technology. It is based on several already existing technologies like HTML, JavaScript and the DOM interacting in a new way.

Google maps[2], flickr[3] and ajaxWrite[4] are only a few examples of interactive web applications that already use AJAX. In the following sections the parts giving AJAX its functionality are introduced. Afterwards a short example will explain the interactions of of these parts.

3.1.1 XMLHttpRequest

The `XMLHttpRequest` object is the main part of the *AJAX* approach. It is the technical component that makes the asynchronous client server communication possible. The object is created in JavasScript like in Listing 3.1.

Listing 3.1: Creating an instance of the `XMLHttpRequest` Object

```
1  var xmlHttp;
2  function createXMLHttpRequest(){
3          if (window.ActiveXObject) {
4                  xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
5          }
6          else if (window.XMLHttpRequest) {
7                  xmlHttp = new XMLHttpRequest();
8          }
9  }
```

The `XMLHttpRequest` object is available in all current browsers. The source code in Listing 3.1 shows differences between Microsoft's internet explorer and other browsers. In Microsoft's browser the `XMLHttpRequest` is an ActiveX object in contrast to others where it is a normal JavaScript object. Thus there is the check for an ActiveX object. Some differences between the objects of the varying browsers are existing, but they share a common subset of Methods and Properties that are sufficient for many applications.

The Methods

Very important for the implementation of an AJAX web application are the methods `open()`, `send()` and `setRequestHeader()` listed in table 3.1.

[2]http://maps.google.com/
[3]http://www.flickr.com/
[4]http://www.ajaxlaunch.com/ajaxwrite

Table 3.1: Standard `XMLHttpRequest` Methods(2)

Method	Description
`abort()`	Aborts the current request
`getAllResponseHeaders()`	Returns all the response headers for the HTTP request as key/value pairs
`getResponseHeader(name)`	Returns the value of the specified header
`open("method","url",["syncFlag"])`	Sets the stage for the call to the server. Method can be: GET, POST or PUT. The optional syncFlag allows to switch between syncronous or asyncronous client/server communication
`send(content)`	Sends the request to the server
`setRequestHeader("header","value")`	Sets the specified header to the supplied value. open() must be called before attempting to set any headers

The `open()` method determines the HTTP request method by choosing GET or POST as parameter. Another parameter of the `open()` method is the URL of the target for the request on server side. The kind of communication is selected by an optional flag. It can be TRUE for asynchronous and FALSE for synchronous communication between server and client side. By default the flag is TRUE. A FALSE only makes sense in cases where input has to be validated and the user should not continue with the application. The `setRequestHeader()` method is necessary to specify the header of the query, which is submitted as parameter of the `send()` method.

The Properties

Additionally to the above mentioned methods, there are some useful properties of the `XMLHttpRequest` object taht are listed in table 3.2. These properties allow the client side to process server states and the response.

The `onreadystatechange` property of the `XMLHttpRequest` object is an important part of an AJAX implementation. The property is ususally combined with a JavaScript handler function - The callback function. The porperty stores a pointer to this function and `onreadystatechange` calls it after every change of the server, or the `XMLHttpRequest` object state . In this callback function these states can be checked with other properties. The `readyState` prop-

Table 3.2: Standard XMLHttpRequest Properties(2)

Property	Description
onreadystatechange	Event handler that fires at every state change. Usually calling a JavaScript handler function
readyState	The state of the request: 0=uninitialised, 1=loading, 2=loaded, 3=interactive, 4=complete
responseText	The response of the server as string
responseXML	The server response as XML
status	Http status code from the server
statusText	The server status as text

erty returns the state of the XMLHttpRequest object. The possible states are listed in table 3.2. A server status check can be performed with the status property. The returned states are the HTTP-status-codes[5], that inform about successful requests or allow an appropriate error handling. If server and request are in the desired state the response of the server can be extracted by the responseText or responseXML properties of the XMLHttpRequest object.

With the presented methods and properties the core of the AJAX technique, the asynchronous communication between client and server is possible. But there is something more to AJAX than the XMLHttpRequest object. In an interactive web application the content of the page invoking the AJAX request has to be send to the server and refreshed with the response of the server. The methods to achieve such a behaviour of websites are provided by JavaScript and the DOM.

3.1.2 JavaScript and DOM

DOM[6] is the abbreviation for Document Object Model. It is a platform and language neutral interface that allows programs and scripts to dynamically access and update contents, structures and styles of XML and HTML documents. With the DOM methods available in JavaScript it is possible to read the content of HTML Tags containing the information for a query string, or to put new

[5] Status Code Definitions, http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html, (August 2006)

[6] http://w3.org/DOM, (August 2006)

elements into the existing document. Changes to the document caused by DOM methods are displayed immediately. The selection of a single element in the tree structure of a valid HTML document can be a tricky exercise.Therefore the commonly used DOM method is `getElementByID("CSS-ID")`. The method selects the element of the document with the CSS-ID which was given as argument. In combination with porperties like `innerHTML` or `value` the contents of the selected elements in a document can dynamically be changed or read.

3.2 A simple AJAX example

After describing the single parts of the AJAX technique an explanation how these structures work together to create an interactive website, follows. The AJAX example in listing 3.2 is a simple HTML document. This page contains a single button that starts an asynchronous request to the server by clicking on it. The server responds by sending the requested file, a static XML-file. The content of the response is then displayed on the already loaded HTML page, without a reload.

The bottom of listing 3.2 contains the HTML-tags responsible for the look of the displayed page. Inside the `<input>`-tag a JavaScript `onclick()`-event handler listens to clicks on the generated button.

A button-click forces the handler to call the `startRequest()` function. The function is part of the embedded JavaScript Document. Inside an instance of an `XMLHttprequest` object is created like in listing 3.1. The `open()` and the `send()` method of the object are used to send a request to the server. The `open()`-method gets the URL of the target for the request as second parameter. In this case a simple static XML file. The first parameter of the `open()` method, in the example set to `GET`, determines the HTTP-request method for the asynchronous request. The asynchronous behaviour of the request is explicitely defined by the boolean `TRUE` as third optional parameter. The `send()` method of the object actually sends the request.

When the request is placed, the server side prepares the appropriate response. In this example the requested file `simpleResponse.xml` containing the line of text `"Hello WORLD!!"` is the answer of the server. But the server response must not be static, it can be a server side script in any language that can handle HTTP-requests allowing to generate the response according to parameters submitted as query string.

By using the `onreadystatechange` property of the `XMLHttpRequest` object it is possible to react at server states. `onreadystatechange` executes the callback function `handleStateChange()` after every change in the internal states of the object. After every state change there is the possibility to retrieve

Listing 3.2: Simple AJAX Example

```
<!DOCTYPE html [...]>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Simple AJAX Example</title>
<script type="text/javascript">
var xmlhttp;

function createXMLHttpRequest(){
        **For details look at listing 3.1**
}

function startRequest(){
        createXMLHttprequest();
        xmlHttp.onreadystatechange=handleStateChange;
        xmlHttp.open("GET", "simpleResponse.xml", true);
        xmlHttp.send(null);
}

function handleStateChange(){
        if(xmlHttp.readystate == 4){
                if(xmlHttp.status == 200){
                        document.getElementById("response").innerHTML=xmlHttp.responseText;
                }
        }
}
</script>
</head>

<body>
        <form action="#">
                <input type="button" value="Start Request" onclick="startRequest();"/>
        </form>
        <div id="response"></div>
</body>
</html>
```

the state of the object`s and the sever`s status, by checking two properties. A `readyState` of 4 means that the request is "complete". The HTTP `status` of 200 returned by the server means "OK". In the example these states allow the extraction of the server response. With the DOM method `getElementByID("response").innerHTML` the content of an element of the HTML page with the CSS-ID "response" can be changed. The `<div>`-tag carrying the ID will be filled with the response. Therefore the `XMLHttpRequest`s property `responseText` is used to read the content of the submitted file.

Figure 3.1 shows the output of the example code in a Mozilla Firefox browser.

3.3 AJAX - Disadvantages

After the discussion of the advantages of *AJAX* it is also necessary to take a look at the limits and negative sides of this technique.

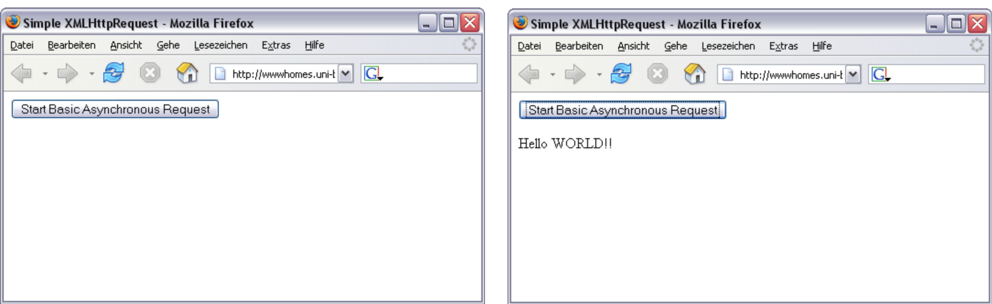

Figure 3.1: Simple AJAX Example

A first important point is that *AJAX* applications need browsers with activated JavaScript. So a service with many visitors not using JavaScript should not apply it. Some useful methods for *AJAX* still differ from browser to browser or are not supported at all. Tests with different browsers are therefore necessary.

The responsiveness of *AJAX* websites is not only a positive nature. Users can be irritated by the unusual behaviour of the AJAX applications. Clicking a button to start any server activity is still standard in internet applications nowadays. Additionally the handling of most websites is very similar and thus very easy. Therefore, forms without a corresponding submit-button are possible when using *AJAX*, but undesireable in most cases. The users do not profit when the handling of web applications is changed in this way.

The unrecogniseable data submission possible with *AJAX*, especially when using forms, can also lead to a increased feeling of insecurity. Users do not allways know when something and what happens to their entered data. Therefore users need a feedback that something is running in the background of the application. The server action has to be visualized. For the user it must be obvious that chages on an HTML document occured, when there was a modification by *AJAX*.

Many *AJAX* applications have a problem with the functionality of the browser-back-button. This button used to go back to the previously displayed page, does not work properly with *AJAX* because everything is happening on one HTML document. The changes caused by user activities do not have an effect on the browser history. Using the back-button then will lead the user to a prevously visited page, but not back in the current application. Meanwhile some solutions for this problem are available. A simple print of a HTML document that was changed by *AJAX* is not possible, either for the same reasons. The internet browser will only print the information of the page in the startup state,

when loaded from the server.

An implementation of a file upload to a server is not easy to realise with *AJAX* support, too. Browser incompatibilities are also responsible in this case.

The listed problems are mostly important for user experience of implemented applications. The reasons do not really speak against the use of the AJAX technique. These problems with *AJAX* only show that it only should be applied where it is meaningful and where it does not lead to an exclusion of many users.

CHAPTER 4

GENEFISHER2 - AJAX BASED IMPLEMENTATION OF THE SUCCESSFULL PRIMER DESIGN TOOL

4.1 GeneFisher2 - The Improvements

After the detailed view inside *GeneFisher`s* workflow in Chapter 2 and the survey of the AJAX technique, the realisation of *GeneFisher2* is now discussed in detail. With the implementation of *GeneFisher2* the possabilities of the classical *GeneFisher* program were kept but the user demanding situations described in section 2.2 were avoided. Users now make the same steps to calculate their results, but the porgram got an increased usability. The *AJAX* technique is very important for the realisation of the user-friendly *GeneFisher2*.

Figure 4.1 is a chart of the displayed HTML pages of *GeneFisher* and *GeneFisher2* during a calculation of primers. The input for this example are multiple DNA sequences. These sequences are going through an alignment, the consensus sequence generation and then to the final primer calculation.

At first glance it becomes obvious that there is a strong reduction of pages a user has to go through, in *GeneFisher2*. Where *GeneFisher* took three, four or even five HTML page for partial result, the calculation of every important step takes place on only one HTML Document, in *GeneFisher2*. This explicit condensation is possible by using the AJAX techique.

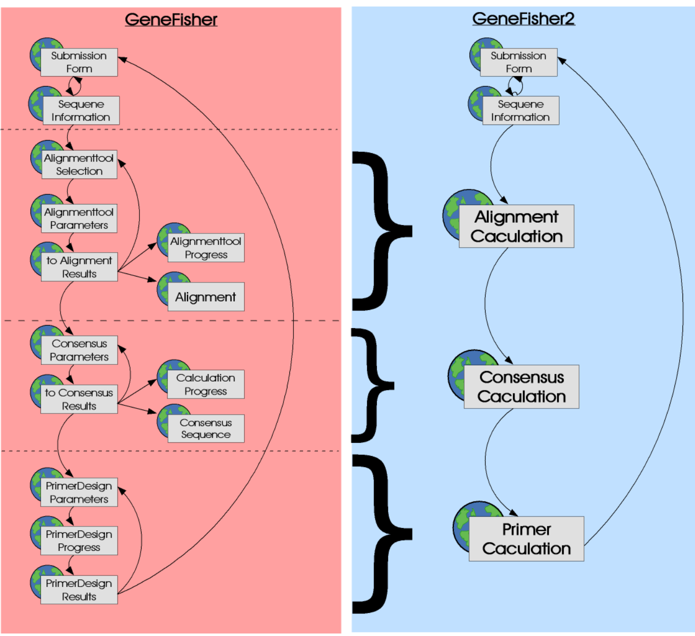

Figure 4.1: HTML Pages of GeneFisher and GeneFisher2 displayed during Primer-Design progress with multiple DNA sequences

4.2 GeneFisher2 - Realisation of GeneFisher with AJAX

After a general overview of the improvements in *GeneFisher2* an observation of its scripts shows how it was realised in detail.

Figure 4.2 is a flowchart of the *GeneFisher2* program. In contrast to figure 2.2, showing the details of the *GeneFisher* workflow, the chart contains JavaScript documents as additional structure. The JavaScripts are responsible for the improved communication between HTML pages in most cases generated by CGI scripts and server side programs or scripts. They use the `XMLHttpRequest` object of AJAX illustrated in chapter 3.

In *GeneFisher2*, `batcons` and the `gf_2000` binary were used that are already known from *GeneFisher*. The modularity of *GeneFisher* supported the re-use of these parts in the new implementation. Espacially the `gf_2000` binary is important for the functionality of *GeneFisher* and *GeneFisher2* because it is responsible for the reliable primer calculation. `readseq`, `ClustalW` and `DCA` were not used in *GeneFisher2*. These binaries were replaced by webservices that produce similar or better results.

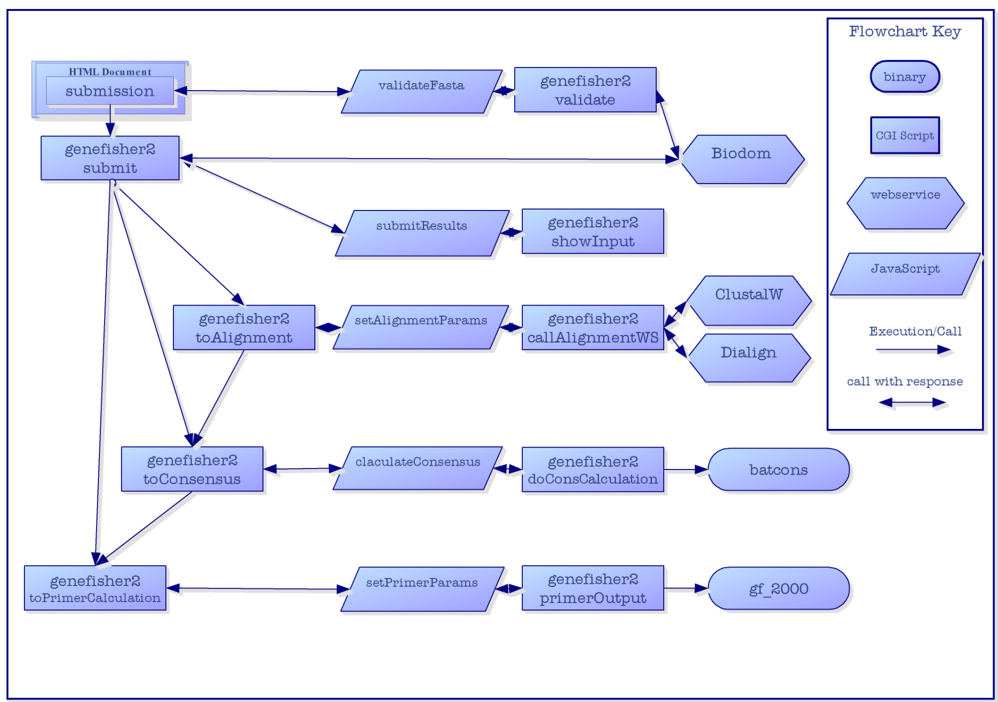

Figure 4.2: GeneFisher2 Flowchart

submission.html

GeneFisher2 starts with the HTML document `submission.html`. On this web document, a first sequence validation with AJAX support is carried out. Sequences submitted by using the textarea of the HTML page are validated directly. Users get an immediate feedback about the legitimacy of their input on the submission page.

For the sequence verification the `BioDOM`[1] webservice took the role of `readseq` in *GeneFisher*. The `BioDOM` webservice converts the native sequence input like the required FASTA format to XML formats that can be validated against XML schema definitions. Therefore the validation is more restrictive and reliable than the slight validation made by `readseq` in *GeneFisher*.

[1] http://bibiserv.techfak.uni-bielefeld.de/biodomws/

genefisher2_submit

In the next step through *GeneFisher2* the classical HTTP request approach is used. The `genefisher2_submit` script is called by the `submission.html` page. This conventional step is necessary because the file upload with AJAX as mentioned in section 3.3 is complicated and not every browser supports it in the same way. Thus files entered in the file-upload form of `submission.html` are uploaded and validated by `BioDOM` in the next step.

genefisher2_showInput

After all submitted files were transformed to an appropriate XML format like `SequenceML` or `AlignmentML`, they are displayed with help of XSLT scripts. AJAX is used to change the view form a tight version of the input, to a detailed version showing the whole sequences and vice versa. The JavaScript document `submit Results` and the Perl script `genefisher2_showInput` implement this function.

According to the `BioDOM` results and the informations collected by the script the taget script is chosen. Like in the *GeneFisher* workflow, *GeneFisher2* decides if the input were aligned, unaligned, multiple, single, amino acid or nucleotide sequences and determines the next step.

genefisher2_toAlignment

`genefisher2_toAlignment` is the next station for two or more unaligned sequences. On the website generated by the CGI script users can choose the alignment algorithm for the multiple alignment of their input. At the moment `ClustalW` and `Dialign` are available. The webservices of both algorithms can handle the sequenceML format provided by the `BioDOM` webservice that is used for validation. The selectable parameters corresponding to every alignment webservice are adapted by AJAX. A slightly validation of the entered parameters of the users is made by the JavaScript document `setAlignmentParams`, too. The user invokes the AJAX call to the CGI script that makes a SOAP request to a selected webservice. Because of the long running times of an alignment calculation an animated GIF is displayed to give the user a feedback that the calculation is still going on. Changing messages after a certain number of recalls to the webservice are shown to support this. The results of the multiple alignment are put on the HTML page containing the parameters. If the alignment does not match the expectations a recalculation is possible instantly.

genefisher2_toConsensus

genefisher2_toConsensus is the following CGI script, right after the alignment or when the entered sequences have been aligned before. The constructed HTML page of this script, allows the user to set the parameters for the consensus calculation or additionally in cases of an amino acid input the backtranslation parameters, too. The entered parameters are checked by the JavaScript calculateConsensus. The batcons binary is executed by the genefisher2_doConsCalculation script. It is the server side script of an XMLHttprequest object that returns the consensus sequence and the multiple alignment in a HTML table.

genefisher2_primerCalculation

The last step is the call of the genefisher2_primerCalculation. The script is the basement for the primer calculation. The parameter seletion for the following calculation is possible on this page again. A slight validation of the adapted parameters is implemented by the JavaScript setPrimerParams. The JavaScript document also starts the primer calculation by calling the genefisher2_primerOutput CGI script. The script parses the parameters according to the requirements of the gf_2000 binary. The script then executes the binary and reads the results out of a predifined file of the project. The results are added to the parameter setting HTML page. The user can check his results again directly on the page where the calculation parameteres are set and the calculation is started. If the results are not satisfying another calculation with different parameters can be started immediately.

4.2.1 AJAX - Not always the right choice

The calls in figure 4.2 going directly from one *GeneFisher2* script to another, like for example the call from genefisher2_submit to genefisher2_toAlignment, are allways classical HTTP requests. These synchronous calls help to keep a structure in the logical workflow (Figure 2.1). The calculation of multiple alignments, the consensus and the primer calculation are the three logical sections of the program. Every important section due to AJAX now takes place on a single page. If one section is completed the user can continue with the next step meaning that a new page will be loaded. Figure 4.1 shows that combining AJAX with classical HTTP requests also improves the program`s structure. Another advance of conventional HTTP requests is, that the users are not exposed to a completely strange behaviour of the application, caused by the new AJAX technique.

4.2.2 GeneFisher2 - The Input Validation in Detail

Listings 4.1, 4.2 and 4.3 contain excerpts of the source code from `submission.html`, `genefisher2_validate` and `validateFasta.js`. These three documents are performing a sequence validation with AJAX support. It is an example of how *AJAX* is applied in the *GeneFisher2* implementation.

The shortened HTML document `submission.html` in listing 4.1 only contains a textarea and a button, as visible elements. The JavaScript file `validateFasta.js` is integrated into the document head. A click on the button the `onclick()`-event of JavaScript calls the function `checkFasta()` contained in the embedded JavaScript-document `validateFasta.js`.

The function `checkFasta()` calls the `createXMLHttpRequest()` to generate an instance of the `XMLHttpRequest` object, like in listing 3.1. A query string is assembled with help of the DOM method `getElementByID()`. Query strings of HTTP requests are key/value pairs. Keys and values are parted by the equals sign. Different key/value pairs are divided by an ampersand. The text contained in the textfield of the HTML document 4.1 is the value of this pair. In this example the `open()` method of the `XMLHttpRequest` object uses the "POST" method for the HTTP request to the specified URL. Accordingly the `send()` method of the object gets the query string as parameter. The target of the `open()` method is the `genefisher2_validate` script.

`genefisher2_validate` is a server side perl-script. The script uses the CGI-module. Supported by this module the submitted value of the query string can be extracted, simply by using the apropriate key. Using the SOAP::Lite perl-module and the BioDom WSDL, a soap request is send to the `BioDom` webservice. The request contains the submitted sequences. The Biodom-Webservice is fast and the response comes right after the submission. The response of BioDom is validated against regular expressions. Unsuccessful webservice calls respond an error-message where a description of the error can be extracted and printed. The response of successful calls leads to the message "Your input is valid FASTA DNA-Sequence". These messages are printed to STDOUT.

The `onreadystatechange` method of the `XMLHttpRequest` object in listing 4.3 carries avpointer to the callback function defined in the JavaScript document. After every change in the state of the server side script called by the `open()` method the callback function `handleResponse` is executed. In this function the state and the status property of the `XMLHttpRequest` object are checked. A `readyState` of 4 means "complete" and the HTTP `status` of 200 means "OK". Other possible states allow a error- and status-handling of server-requests, not interesting in this case. The response of the server can be extracted with the `responseText` property of the object. The DOM method `getElementByID()` and the property `innerHTML` put the reponse of the server

Listing 4.1: HTML Document - submission.html

```
1  [...]
2  <html>
3  <head>
4          [...]
5          <script src="validateFasta.js" type="text/javascript">
6          [...]</script>
7  </head>
8  <body>
9          [...]
10         <textarea name="text" id="text" rows="8" cols="70">
11         [...]</textarea>
12         [...]
13         <button type="button" id="checkFasta" onclick="checkFasta()">
14         Check Fasta</button>
15         [...]
16         <div id="isvalidFasta"></div>
17         [...]
18 </body>
19 </html>
```

Listing 4.2: Perl Script - genefisher2_validate

```
1  #!/bin/perl5.8 -w
2  use CGI;
3  use SOAP::Lite;
4  [...]
5  my $query = new CGI;
6  my $textfile = $query->param('text');
7  my $uriSequence = URI->new("http://bibiwsserv.techfak.uni-bielefeld.de/biodomws/axis/SequenceML?wsdl");
8  
9  print $query->header();
10 
11 my $result = SOAP::Lite->service($uriSequence)
12     ->readable(1)
13     ->outputxml(1)
14     ->fromFasta($textfile,4);
15 [...]
16 if ($result=~/.*<faultstring>(.*)<\/faultstring>.*/i){
17     print "<b>$1</b>";
18     exit;
19 }
20 [...]}
21 if ($result=~/.*<nucleicAcidSequence>(.*)<\/nucleicAcidSequence>.*/ig){
22                 [...]
23                 print "Your␣input␣is␣valid␣FASTA␣DNA-Sequence";
24                 exit;
25 }
26 [...]
```

Listing 4.3: `JavaScript - validateFasta.js`

```
1  var xmlHttp;
2
3  function createXMLHttpRequest(){[...see Listing 3.1...]}
4
5  function checkFasta(){
6  [...]
7  var url="/cgi-bin/genefisher2_validate";
8  [...]
9  var queryString = "text="+document.getElementById('text').value;
10 xmlHttp.open("POST",url,true);
11 xmlHttp.onreadystatechange=handleResponse;
12 xmlHttp.send(queryString);
13 }
14 function handleResponse() {
15         if (xmlHttp.readyState==4){
16                 if (xmlHttp.status==200){
17                         var response = xmlHttp.responseText;
18                         document.getElementById('isvalidFasta').innerHTML=response;
19                         if (response!=""){
20                         [...]
21                         document.getElementById('Submit').type="submit";
22                         [...]
23 }
```

directly into the desired element of the HTML document shown in listing 4.1. Without refresching the page he results are added to the document.

This example points out again, how usefull AJAX can be in web-applications. The immediate feedback to server-requests without a page reload is one of its most important features.

4.3 Outlook

Adding AJAX to other programs validation on server side? Some changes to the User interface still necessary New primer design binary replacing `gf_2000`.

BIBLIOGRAPHY

(1) *Ajax: A New Approach to Web Applications.* Garrett J.J., adaptive path, 2005

(2) *Foundations of Ajax.* Asleson R. and Schutta N.T., Apress, 2006

(3) *AJAX(Web 2.o in der Praxis).* Gamperl J., Galileo Computing, 2006

(4) *GeneFisher - Software Support for Detection of Postulated Genes.* Giegerich R. et al, 1996

(5) *Algorithmic Support for PCR and Genome-wide Repeat Analysis.* Schleiermacher C. 2001

(6) *Mastering AJAX.* McLaughlin B., IBM developerWorks, 2005

(7) *Ein Framework für die Entwicklung asynchroner WebServices auf dem BiBiServ.* Mersch H., 2004