Browser Based Remote Control of Hearing Aids

Marno van der Maas[1]

***Abstract*—Due to uncertainties about user-preferences and unforeseen changes in the environment, real-time processing systems like hearing aids require frequent fine-tuning moments. It is interesting that these adjustments are many times made very infrequently and only by professionals who have to stop the system completely for the adjustment. The architecture described in this paper makes it possible for users to control a hearing aid algorithm from their smartphone's browser and makes it possible to adjust the parameters of a processing system in real time. The implications of such an architecture is that hearing aid algorithms can eventually adjust themselves to the patient's preferences by using machine learning techniques.**

***Index Terms*—hearing aid, remote control, HTML5, jQuery, mobile computing, remote monitoring, parameter adjustment.**

I. Introduction

REAL time parameter adjustment is important in processes like hearing aid algorithms and video feed enhancement. As can be seen in [2] hearing aids are currently adjusted by professionals and not by the patient. It means that every time a patient has a problem they need to visit their audiologist for adjustment. Additionally, the possibility to solve common problems using software already exists according to [4]. This software can solve problems according to the descriptors the patient uses to characterize their experience. Besides time-efficiency, real-time parameter adjustment enables machine learning to be introduced to hearing aids, because desired output can be matched to the input. The future could therefore hold a hearing aid which can be adjusted to the patient's listening preferences and ultimately the patient does not even have to go to the audiologist if their hearing loss changes.

In order to make a first step in the vision described above, a remote-control application that governs the adjustment process is needed.

Problem statement: The paper investigates an architecture of a communication medium between a hearing aid algorithm and a user input device.

More specifically the task of the architecture is to set up a communication channel between a hearing aid algorithm running on MATLAB and an HTML5 browser on a user's smartphone.

The above problem can be split up in a number of sub problems.

- First, the MATLAB algorithm must run continuously and so there must be a way to create a separate thread for the server to run on the PC.
- Second, there must be a way for the MATLAB thread to communicate with the server, which can be split up in what data will be communicated and over what medium.
- Third, there must be a communication channel between the PC and the browser, which should make available the browser application itself and should provide the possibility of communicating parameter values.

The above sub problems should all be solved with two important ideas in mind, which are the real-time property of the communication and the versatility of the architecture.

- The communication latency should be minimized for the sake of user comfort and machine learning techniques that can potentially be used in conjunction with this system.
- Secondly, the whole architecture should be designed to be versatile in the sense that it can work with any parameter library and thus not be tailored to a specific algorithm.

A versatile architecture is important, because this property is what will make it possible to generalize the techniques that are used in this specific application to what any parameter-adjustment application should do. The architecture could potentially also be used for real-time video enhancement or other remote control applications like TV's or computer games.

In the next section the design decisions of the architecture are illustrated, which is followed by a section exemplifying the architecture by means of a specific implementation. The specific implementation will be used as a validation that the architecture fulfills all the aforementioned requirements. The requirements in conjunction with the future research opportunities will be highlighted in the discussion.

II. Research Methods

The architectural design is described in this section in accordance with the sub problems stated in the introduction. The architecture is viewed from the perspective of hearing aid algorithm development, but the hearing aid algorithm can easily be replaced by another process that provides tuning parameters.

In Figure 1 one can see a high-level, schematic representation of the main elements of the architecture and the communication between these elements. The main communication sub problems are between the Java server and MATLAB algorithm and the communication between the browser and the server. The questions of what medium of communication to use and what data objects to communicate are made more complex by the fact that three different programming languages have to be used. These programming languages are JavaScript, Java and MATLAB code. The reasons for splitting the system up in these four blocks and for choosing the specific technologies are presented in the following sections.

A. Application Environment

The task calls for a user controlled application to be developed, which is the bottom block in Figure 1. The most

Manuscript received June 11, 2013

[1]M. van der Maas is a Bachelor of Science student at the Department of Electrical Engineering, Eindhoven University of Technology, e-mail: (m.h.j.v.d.maas@student.tue.nl), ID number: 0733769.

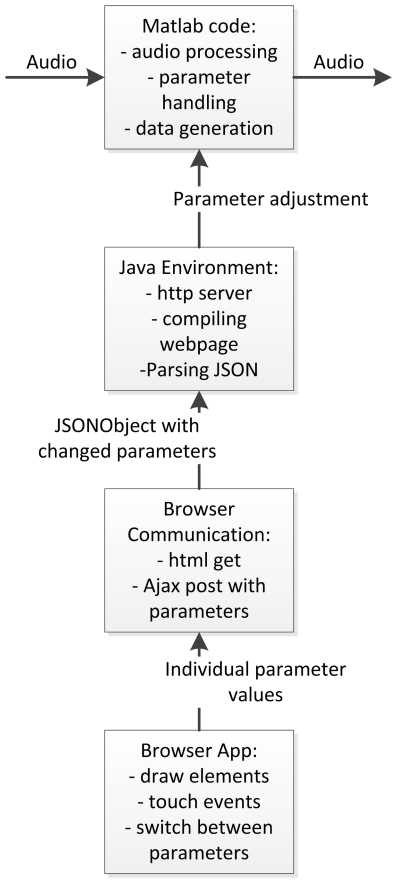

Fig. 1. High-level design of the system so that it fulfills the task description.

natural choice for user input device that supports graphical user interfaces is the smartphone, because it is not desirable to manufacture a custom remote control for each hearing aid (like the ones described by [8]). On the smartphone, the browser is one of the most pervasive mobile application environments available on which to develop a remote-control, parameter-adjustment application. According to [5] mobile app usage has increased steadily over the years and the penetration of browsers is just as high as that of applications. However, applications are platform specific, while browser applications are universal to all platforms or can be made universal with little effort, which is outlined in [19]. The only disadvantage to browser applications versus native applications, according to [19], is that it cannot use hardware as efficiently in order to increase performance. However, the client-side of the application is not computationally heavy, since the hearing aid algorithm and server operations do not run on the client, so this increased performance is superfluous. One reason to not use a browser on a smartphone is if Bluetooth is required, due to power consumption as described in [7]. However, since in the problem description the communication is with a PC, using Wifi is not seen as a problem. The browser on a smartphone is the chosen environment on which to develop a hearing aid adjustment app.

B. Concurrency with MATLAB

For this task, the environment on which the algorithm runs is fixed, which is the top block in Figure 1. However, MATLAB does not automatically provide HTTP server capabilities and the hearing aid algorithm must run on a separate thread than the server so that the audio processing can run uninterrupted. A positive aspect of MATLAB is that it has a built-in Java Virtual Machine, which means that a Java based HTTP server could run on a Java thread, parallel to the MATLAB main thread. Java has an HTTP server in its standard library, so this makes it possible to communicate with the browser identified in the previous section.

C. MATLAB Communication with Java Server

We have argued that because the top block is based in MATLAB it is logical to have the second block to be an HTTP server running in Java. However, communication between two threads is not a trivial task, especially not when these threads are in two different programming languages.

1) Medium of communication: An asynchronous communication medium must be set up between MATLAB and Java. This is accomplished by making a queue of parameter objects that is designed for concurrent use. This queue serves as a waiting line of packages that the server can fill up when requests come in and MATLAB can read out whenever the audio processing activities can be safely interrupted.

2) Object of communication: The object of communication between the Java HTTP server and the MATLAB environment should be an object that both environments can interpret. The most straight forward approach is to create a Java class that contains all the attributes that can be sent over the communication channel. The Java environment can then fill in the attributes of the object before placing it in the queue and MATLAB can read out the attributes when it is time to read from the queue.

D. Browser Communication with Server

In section II-A, the browser is chosen as the user-side communication end-point and in section II-B it is made clear that an HTTP server running on the Java platform is used.

1) Protocol: Browser communication is based on the HTTP protocol. However, there are many other technologies available to handle server-side events, as listed in [1]. There are two main categories of server-client communication: One consists of only client-initiated connections (client-pull model) and another also involves server-initiated connections (server-push model) [10]. Server-push models, however, would cause extra complexity on mostly the server side because HTTP connections have to be kept open as opposed to the traditional web communication of only client pulling [18]. Besides complexity, memory may also become a problem since for each client a new connection needs to be maintained. Also, unless streaming or real-time results are necessary, [17] states that client pulling is always preferred over server pushing. Since the server software will probably have to run on a hearing aid device, server-side complexity, memory and processing should be

limited in order to preserve battery charge. The only problem with a client-pull model is that if the hearing aid wants to send information back, a polling system must be used. This has more latency than other techniques like websockets as stated by [16], whereas other sources like [15] say that the speed improvement is limited. However, the communication from server to client is not implemented in the current application and it is found that latency is not a problem for communicating simple feedback data. Lastly, Ajax requests (a common form of client-push communication) are automatically supported in an HTTP server and in browsers. This is why HTTP Ajax post requests are chosen as the medium of communication instead of a persistent connection between server and client, which websockets establish [3]. The server will then only be active when the client sends or requests information.

2) Package format: Besides deciding what protocol to use for communicating between the browser and the server, it must be decided what data should be sent over the link. JavaScript automatically displays its objects in JSON, which is "a text format that is completely language independent" [9]. A reason to use JSON instead of, for example, XML is because "XML is not well suited to data-interchange" because it "carries a lot of baggage" [14]. JSON in comparison is very light-weight and Java has ample library support for parsing JSON strings. The Java library that is used in this project is Json-lib. Because JSON "is a lightweight data-interchange format" that is "easy for humans to read and write" and "easy for machines to parse and generate" [9], it is used as the means of formatting data over the client server connection.

E. Versatility

As can be seen in section II-C2 and II-D2 the data objects that are sent over the communication channel make no assumptions about the format of the parameter data. This means that the architecture can be easily extended to include additional parameter types. Important to note is that the Java server is completely independent of the MATLAB environment, which means it can easily be used in another system. In general each block in Figure 1 should be as independent of the other blocks as possible. Versatility is important in parameter adjustment architectures because they can then be applied in numerous circumstances with little or no adjustment. The last point that should be made to increase versatility is that the application should be compiled when the server starts with the tuning parameter library that the hearing aid provides. Making adjustments to the tuning parameter library will then be no problem for the application.

F. Libraries versus Custom Code

The last block in Figure 1 that has to be identified is the third from the top. JavaScript is a logical choice for browser scripting, especially due to its support, see [6]. However, there are many API's available for JavaScript. This parameter-adjustment application should use a standard library in order to realize communication and user interfaces. In [12] the advantages of using the standard library jQuery are stated as being; ease of use, the size of the library, available documentation and Ajax support. All of these are reasons to use jQuery instead of a custom implementation. It is important that jQuery handles all the nitty-gritty browser differences (especially for functionality like Ajax post requests), which programmers would otherwise need to implement on their own. It makes setting up a connection between client and server simple and reliable. The two disadvantages that are named are that functionality may be limited and that the JavaScript file needs to be loaded. The limited functionality is not a problem, since additional functionality is realized by including other compatible libraries like jQuery-UI. The disadvantage of the browser needing to load the JavaScript files of these libraries is not very relevant, since clients usually cache these files and thus they only have to be loaded once. Also the files are relatively small as stated in [13]. (The version that is used in this project is the 2.0.0 minified version.) Besides the main functionality of jQuery, extensions like jQuery UI and touch-punch were used to make the user interface. Additionally, jQuery is the most popular JavaScript library according to [11], the extendibility of jQuery by using jQuery UI and touch-punch are reason enough to use jQuery instead of other JavaScript libraries. Using the jQuery API and libraries that work in conjunction with it, is the most time-efficient way to create a browser-based hearing aid application that is compatible with all major browsers [6].

III. Research Results

This section describes the realized application, which can be used in conjunction with developing algorithms in the MATLAB environment.

A. Types of User Input

The users should be able to input different types of parameters. Currently, there are three types of parameters that are implemented. A one-dimensional slider outputs a value between zero and one, which could be deployed for volume control. Adjusting high and low frequency hearing impairment simultaneously can be accomplished using a two-dimensional plane and can be used in conjunction with parameter projection on the MATLAB side. And finally, a discrete selector to choose from items in a list, such as a number of source audio files.

Besides parameter adjustment the user can also tell the MATLAB algorithm to play the output audio over the sound card, to pause the audio or to stop the system completely. Stopping also means the server is terminated.

Listing 1 shows the skeleton of a JSON object with all the possible keys and can be used in conjunction with Table I to know how the JSON string is formatted for each of the implemented input types. Only playback control is missing in the table, which sends a JSON object with only the 'playback' key.

```
{
  'parameter': name,
  'x-coord': x,
  'y-coord': y,
  'value': val,
  'selected': select,
  'playback': control
}
```

Listing 1. Format of a JSON object

TABLE I
TABLE SUMMARIZING THE ATTRIBUTES OF THE JSON OBJECT FOR EACH OF THE DIFFERENT TYPES OF INPUTS THAT ARE SENT OVER THE AJAX REQUEST

Attribute	Type	1D input	2D input	Discrete input
parameter	String	✓	✓	✓
value	double $[0, 1]$	✓		
x-coord	double $[0, 1]$		✓	
y-coord	double $[0, 1]$		✓	
selected	String			✓

B. Detailed Description of the System Workings

The main workings of the current system are shown in Figure 2. It has two main parts; The first is an initiation phase, in which the hearing aid is started and the web application is compiled according to the parameters the hearing aid accepts. The second phase is a potentially-infinite loop, in which the hearing aid algorithm runs and periodically checks a queue for parameter adjustments. Concurrently the server is accepting any page requests and parameter adjustments that the user passes on. Besides parameter adjustments the user can turn on and off the audio by sending a 'play' and 'pause' signal respectively. The system can be stopped completely by sending a 'stop' playback control command.

The system consists of four objects and an actor. The GUI object consists of a UI part in HTML and a communication part in JavaScript. The server acts as a go between for the browser and the hearing aid operations, which are regulated by the demo. The demo sets up all necessary parts of the system and decides when the hearing aid checks the queue in the server. The aid simply provides the tuning parameters that it accepts and runs the algorithm, which is a black box from the perspective of the designed system. The patient is the user that provides input to the system and is an actor because the system reacts on the input that it provides.

1) The GUI: is compiled in the server by inserting each parameter in the hearing aid's tuning library. The main purpose of the GUI is to make it easy for the user to adjust parameters via their smartphone's browser. It does this by having a non-cluttered interface with a tab for each parameter, a concept design can be found in Figure 3. Simple touch inputs and button presses are converted to post requests that the server can interpret. These Ajax requests contain data objects formatted according to the JSON standard.

2) The server: is started by the demo. It passes on the parameter library to the GUI by compiling a webpage. It is responsible for providing the GUI at the default endpoint and all other necessary files to the client when an HTTP request arrives. Lastly, there is a command endpoint where all the post inputs are received. This access point contains operations to parse the received JSON data and to place the parameter adjustment or playback control in the queue.

3) The demo: is the main controller of the hearing aid algorithm. The demo is a concept of how a hearing aid should run. It constructs the aid object, which is the audio algorithm. After retrieving the tuning parameter library from the aid it creates a server, which uses the tuning parameters to construct a browser application. Following the initiation phase, the main operations of the demo begins, which is to regulate the running of the audio algorithm and checking the queue in the server. Currently the demo checks the queue at fixed time intervals, however the demo can also check the queue only in favorable conditions. This can be done if the hearing aid provides feedback and will not affect the Java server since the queue will just fill up if the hearing aid is busy for longer periods.

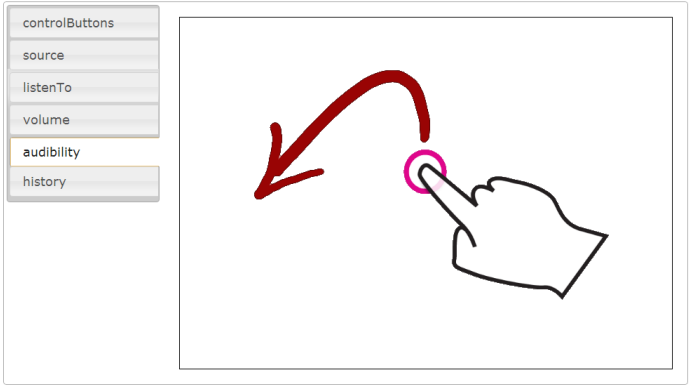

Fig. 3. The tabular layout of the client side application.

4) The aid: in this sequence diagram is the algorithm part of the hearing aid. It has the job to provide tuning parameters and to run its algorithm. This algorithm includes reading input audio, doing filtering operations and outputting audio. However, the operations of the algorithm are irrelevant to the system. The system would just as well work if the aid was a video enhancement algorithm or a TV. Lastly, the aid must be able to accept parameter adjustments in order to adjust to the patient.

5) The patient: is the only actor in the system. The system loops until the patient is happy. This is because the hearing aid is adjusting to the patients needs. In fact, a better way to write the loop condition is "while the patient will be forever satisfied." In real-life this would be never and ultimately a process like a hearing aid will never stop fitting itself to its owner's preferences. The patient simply has to provide touch input to the browser according to his or her wishes. In other instances, the actor name "patient" may not be appropriate, but in the case of hearing aids it is. The ultimate vision is that the system provides a quick and simple way of continuously satisfying the user's needs.

For further implementation details of the system one should refer to Appendix A, where each system element is described in detail and with code snippets.

IV. DISCUSSION

The discussion first validates the versatility requirement of the given architecture and afterwards it discusses ways that the architecture could be designed differently.

A. Versatility

One of the main goals set while discussing the problem is to ensure that the designed architectures is versatile. This

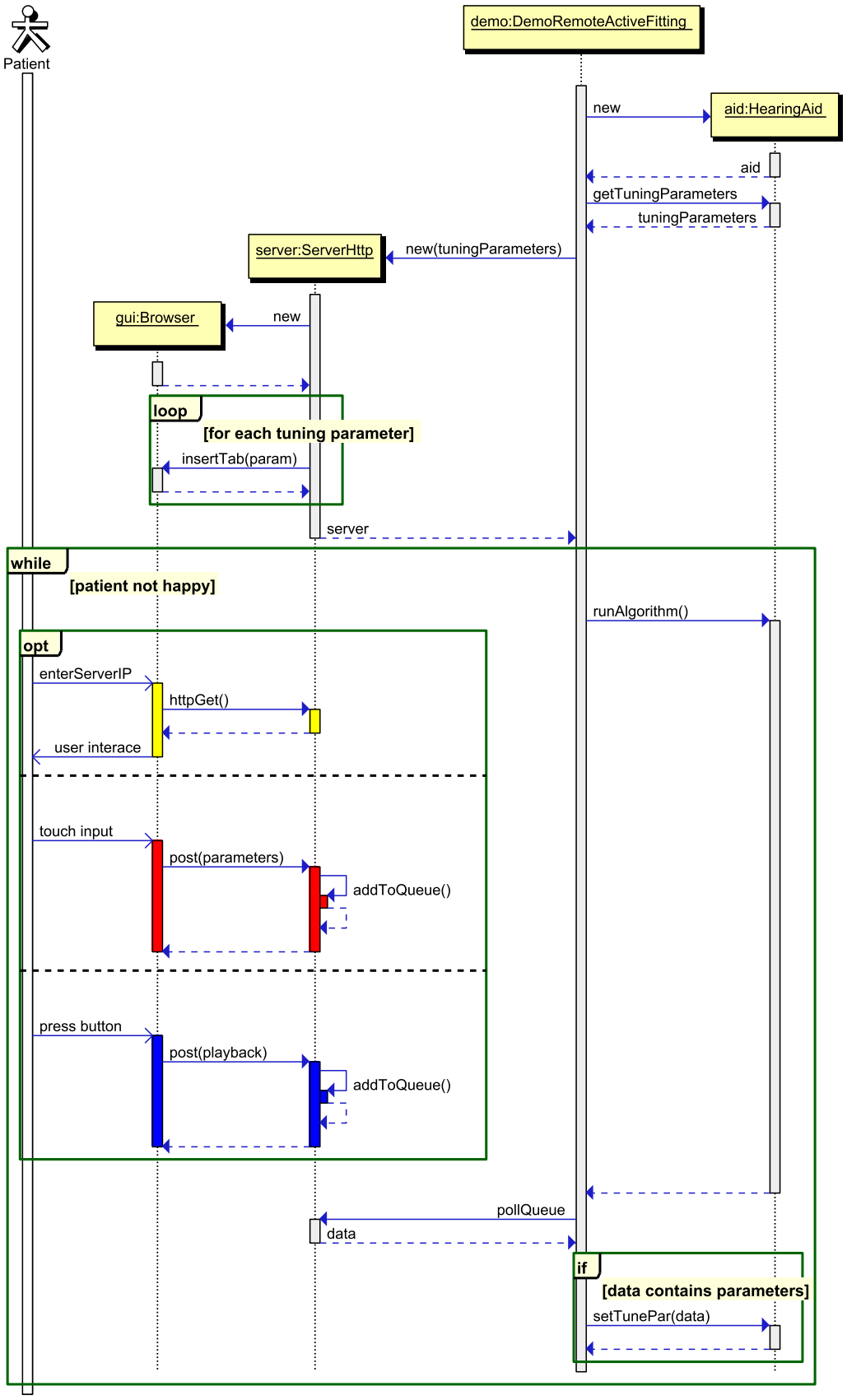

Fig. 2. Sequence Diagram showing the Objects interacting within the Framework.

is achieved by the system not assuming anything about the algorithm that needs to be communicated with. In Figure 2 there is an object called "aid" representing a hearing aid algorithm. However, the only thing it has to do is provide a tuning parameter library, run its algorithm and accept new parameter values. This level of abstraction allows for a high-level of versatility in the sense that any algorithm that fulfills those three requirements can work with the system. The aforementioned is because the webpage is compiled according to the parameter library that the hearing aid supplies. Only when a new type of input is required, additional handling functionality needs to be added in the application, the server and the handler of the algorithm (in the sequence diagram, Figure 2, this handler is called demo). Another point where the architecture is versatile is that changes in the operations of the 'demo' in the loop do not affect the server, since the queue can be called upon whenever the demo sees fit. The asynchronous nature of the communication between the algorithm thread and the server thread makes variable running time of an algorithm have no effect on the server's operations. Knowing that the system is independent of the tuning parameter library, which includes the source of this library, and knowing that the server algorithm communication is asynchronous, the versatility goal is achieved.

B. Future Research

In the introduction the importance of remote-control applications is discussed. It shows that applications like the one described in section III are important for the future of parameter-adjustment in, for example, hearing aid adjustment. It makes it possible for machine learning techniques to be implemented in order to customize hearing aids to the patients needs.

In section II-A, it is stated that the use of the Bluetooth protocol is not used in the current application. However, it may be necessary to use Bluetooth instead of Wifi in a commercial hearing aid due to power consumption. Bluetooth is also supported by most smartphones, but will probably require a native application for each application environment. Future work can write an application using Bluetooth, but should be based on the architecture described in section III, which is still valid. For testing purposes of hearing aid algorithms running in MATLAB the current application fulfills all requirements.

The versatility of the current architecture and application, as described in section IV-A, should always be taken into account during future research. It is paramount that the applications can be easily fitted to every hearing aid, so that each patient can eventually control their hearing aids via their smartphone or similar device. Even devices that are not hearing aids can be controlled using this architecture.

Improvements to the current system can be made as follows. Communication from the algorithm to the user should be implemented. The server could be implemented on an actual hearing aid or other such device (like a TV) in order to test the system in an environment that is more realistic than communicating with a PC. And lastly, improvements to the current User Interface can make the application more attractive to users. This is especially important for marketing if the architecture is coupled with a commercialized product.

V. Conclusion

The research problem is based on the difficulties of creating an architecture to pass parameters from the user to a real-time algorithm. An implementation of such an architecture is given, which realizes communication between a MATLAB-based, hearing-aid algorithm and the browser of a user's smartphone. The architecture fulfills all the requirements set by the sub-problems: it creates a new Java thread for the server so that the algorithm can run uninterrupted; it uses a concurrent queue of Java objects to realize asynchronous communication between the server and the algorithm; it employs Ajax post requests between the client and the server to communicate JSON objects; its communication is in real-time; and the architecture is independent of the type of algorithm. One of the practical applications of this architecture is with hearing aid algorithms adjusting to the patient's preferences. This architecture could make visits to the audiologist less frequent or even obsolete. Ultimately, any algorithm can make use of this type of parameter-adjustment architecture; any user can then adjust parameters of equipment all around them from mobile devices that they already use for other purposes.

Acknowledgment

M. van der Maas thanks Bert de Vries, Joris Kraak, Erik van der Werf and the department of Electrical Engineering at Eindhoven University of Technology for their contributions in the project.

References

[1] A. Flaherty. *A Modern Approach to Server Push*. [Online]. Available: http://answers.oreilly.com/topic/1682-a-modern-approach-to-server-push/
[2] *A hearing aid is customized in several phases*. [Online]. Available: http://www.phonak.com/com/b2c/en/hearing/ways_to_better_hearing/3_steps_to_a_customized_hearing_aid.html
[3] *About HTML5WebSockets*. [Online]. Available: http://www.websocket.org/aboutwebsocket.html
[4] *Addressing patient complaints when fine-tuning a hearing aid*. [Online]. Available: http://blog.starkeypro.com/addressing-patient-complaints-when-fine-tuning-a-hearing-aid/
[5] *Bll REPORT: The Winner Of The Apps vs Browsers War Is*. [Online]. Available: http://www.businessinsider.com/bii-report-the-winner-of-the-apps-vs-browsers-war-is-2012-9
[6] *Browser Support*. [Online]. Available: http://jquery.com/browser-support/
[7] E. Vogier. *Bluetooth vs Wi-Fi Power Consumption*. [Online]. Available: http://science.opposingviews.com/bluetooth-vs-wifi-power-consumption-17630.html
[8] *Hearing Aid Remote Controls*. [Online]. Available: http://www.miracle-ear.com/hearing-aid-remote-controls
[9] *Introducing JSON*. [Online]. Available: http://www.json.org/
[10] J. R. Black. *Server Push - Client Pull*. [Online]. Available: http://www.jrb.com/faq9.html
[11] *JavaScript Libraries*. [Online]. Available: http://www.w3schools.com/js/js_libraries.asp
[12] *jQuery: Advantages and Disadvantages*. [Online]. Available: http://www.jscripters.com/jquery-disadvantages-and-advantages/
[13] *jQuery file size*. [Online]. Available: http://mathiasbynens.be/demo/jquery-size
[14] *JSON: The Fat-Free Alternative to XML*. [Online]. Available: http://www.json.org/xml.html
[15] P. Bengtsson. *Are WebSockets faster than AJAX? with latency in mind?*. [Online]. Available: http://www.peterbe.com/plog/are-websockets-faster-than-ajax

[16] P. Lubbers and F. Greco. *HTML5 Web Sockets: A Quantum Leap in Scalability for the Web*. [Online]. Available: http://www.websocket.org/quantum.html
[17] S. Traugott and J. Huddleston. *Bootstrapping an Infrastructure*. [Online]. Available: http://www.infrastructures.org/papers/bootstrap/bootstrap.html
[18] *Software Tools: Server Push and Client Pull*. [Online]. Available: www.cse.ust.hk/~huamin/111/notes/serverPush.ppt
[19] *The HTML5 Vs Native Apps Battle Broken Down*. [Online]. Available: http://www.businessinsider.com/battle-between-html5-vs-native-apps-2013-5

APPENDIX A
IMPLEMENTATION DESCRIPTION

In this Appendix the detailed workings of the implementation is discussed.

A. MATLAB Description

The following section describes the main functionality that is implemented in MATLAB files, excluding the total system functionality (contained in a demo file in MATLAB) which is already described in section III-B

1) Starting the Server: Initializing the server in MATLAB requires the java path to be augmented. The folder in which all the classes can be found must be added in conjunction with the .jar files which the JSON-lib library requires to run. Finally, when initializing the server, each tuning parameter given by the hearing aid is individually added to the server.

2) Stopping the Server: Stopping the server involves calling the stop function of the server and clearing the server variable from the MATLAB stack. This needs to be called correctly, otherwise the binding of the server to the listening port will not be undone and restarting the server will not be possible unless MATLAB is completely restarted.

3) Processing the queue: Processing the queue involves checking whether the queue of the server is empty or not. If it is not empty, the oldest element is taken off the queue and processed. How the parameter data object is structured can be seen in section A-B. If the object contains parameter adjustments, the modification is communicated with the hearing aid; otherwise if the object contains a playback control, the current playback parameters are adjusted accordingly.

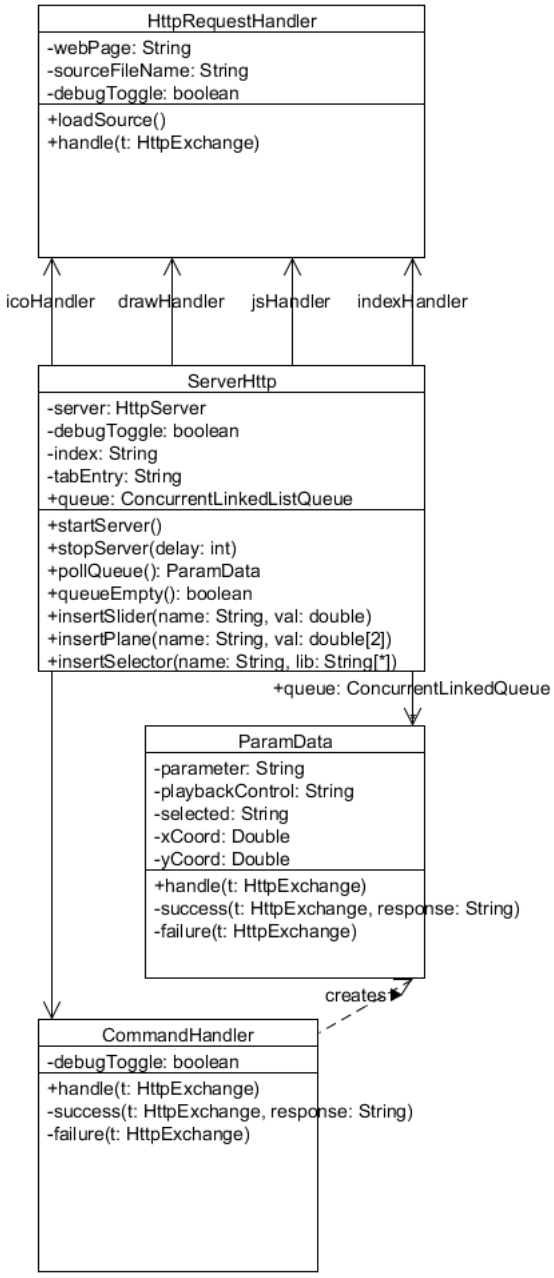

Fig. 4. Class Diagram of Java Server

B. Server Description

This section describes each class in Figure 4 in more detail and , in the process, explains the workings of the server.

1) The HTTP Server: The ServerHttp class is the central point of the Java operations. It itself contains an HttpServer from the `com.sun.net.httpserver` package. Primarily it creates the endpoints to which the HTTP and command handlers are assigned. Secondly, the default endpoint of the webpage is compiled in the server by calling an insert function for each parameter, which entails adding a User Interface item for that parameter. Lastly, a public ConcurrentLinkedQueue is available which can be accessed by the command handler and MATLAB.

2) Handling HTTP Requests: For each endpoint or file that is necessary for the client, an HttpRequestHandler is created, which reads out its corresponding file when it gets a request from the client. The HttpExchange object contains everything necessary for receiving and sending the data, including an input stream, an output stream and ways to adjust the headers. The MIME-type of each request should be in accordance with the content that is being sent.

3) Parameter Data: Instances of the ParamData class contain the data that is sent by the client. Each Ajax request that is received is parsed and the fields of the object are filled. The queue that is in the ServerHttp class contains only ParamData objects and this is the medium of

communication between the Java server on one thread and the MATLAB algorithm on another thread. The details of how the ParamData class is structured can be found in Listing 2.

```
public class ParamData {
  private String parameter, playbackControl,
      selected;
  private Double xCoord, yCoord, value;

  public ParamData(String param, Integer x, Integer
       y, Integer width, Integer height) {
      xCoord = new Double(x.doubleValue() / width.
          doubleValue());
      yCoord = new Double((height.doubleValue() - y
          .doubleValue()) / height.doubleValue());
          // Invert y-axis so that the origin is in
           the bottom left corner.
      parameter = param;
  }

  public ParamData(String param, Double val) {
      value = val;
      parameter = param;
  }

  public ParamData(String control) {
      playbackControl = control;
  }

  public ParamData(String param, String sel) {
      parameter = param;
      selected = sel;
  }

  public String getName() {
      return parameter;
  }

  public double getX() { //Returns -1 if x
      coordinate is not in the data
      if(xCoord != null) {
          return xCoord.doubleValue();
      } else {
          return -1.0;
      }
  }

  public double getY() { //Returns -1 if y
      coordinate is not in the data
      if(yCoord != null) {
          return yCoord.doubleValue();
      } else {
          return -1.0;
      }
  }

  public double getValue() {
      if(value != null) {
          return value.doubleValue();
      } else {
          return -1.0;
      }
  }

  public String getPlayback() {
      return playbackControl;
  }

  public String getSelected() {
      return selected;
  }
}
```

Listing 2. The ParamData class, of which instances form the communication between the server thread and the algorithm thread.

4) Handling Commands: There is a special endpoint in the HTTP server, which is the extension ”/command”. This is where all the Ajax requests arrive. The CommandHandler class first waits for a complete HTTP package to arrive and then offers functionality to parse these posts and package them into ParamData objects.

C. Client Description

The following section describes the client, which solely runs in a browser and is composed of HTML and JavaScript constituents.

1) The HTML Template: This is the skeleton of the index.html file in which all the parameter inputs are inserted. It is based on the tabs of jQuery UI, which can be seen in Figure 3. It contains comments like `<!--INSERTTABSHERE-->` for adding tab entries and `<!--INSERTHERE-->` for adding extra parameter adjustment sections. Additionally, it includes all necessary JavaScript libraries that are needed for the functionality of the app.

2) The Different Types of Input: Each of the inputs use the JavaScript function postAjax, which takes the URL of the page and uses jQuery to post a JavaScript object to the ”/command” endpoint of the server. The source code of the function can be found in Listing 3. The one-dimensional input is a slider is provided by jQuery UI, which uses a function in Listing 4 to draw it. The two-dimensional input is an HTML5 div box and includes an image of a dot that can be moved. The dot is made draggable by including the code in Listing 5, which is enabled by jQuery UI and touch-punch extensions. Lastly there is a discrete input, which uses the HTML selector tag and uses the comment `<!--INSERTOPTIONSHERE-->` to variably insert options according to the hearing aid’s tuning parameter library. Table I summarizes which data fields are sent with the Ajax request with the different input types.

```
function postAjax(o) {
    var sendData = JSON.stringify(o);
    var url = "http://"+document.domain+":8000/
        command";
    jQuery.ajax(url,
        {
            method: "POST",
            data: sendData,
            success: postSuccess,
            error: postError
        });
}
```

Listing 3. The function postAjax(o) in connectionHandler.js

```
function drawSlider(name, val) {
  var sldr = jQuery('#'+name+'Slider');
  sldr.slider();
  sldr.slider("option", "max", 1);
  sldr.slider("option", "min", 0);
  sldr.slider("option", "step", 0.01);
  sldr.slider("option", "value", val);
  sldr.slider({
        change: function(event, ui) {postVal(name);}
      });
}
```

Listing 4. The JavaScript file necessary for configuring and drawing the canvas and the slider.

```
<div id="NAME">
  <div id="NAMEContainer" style="height:500px;border
      :1px solid black;">
    <img id="NAMEDragger" src="http://i1.wp.com/
        jimhull.files.wordpress.com/2011/10/red-dot21
        .png?resize=40%2C40" />
  </div>
  <script>
    jQuery("#NAMEDragger").draggable({
      containment: "#NAMEContainer",
      stop: function() {
        postXY('NAME');
      },
      scroll: false
    });

  </script>
</div>
```

Listing 5. HTML code that is inserted for the 2d input to work, consult the script tag for the JavaScript necessary for making the image draggable.

D. Versatility

This compiling is made possible, because the Ajax request is independent of the name of the HTML element, which is passed on to JavaScript through an argument. An example function in Javascript can be found in Listing 6. Adding new parameters with already implemented types is not a problem only when an additional type is introduced then an additional JavaScript function is needed, the command handler in the server should be extended to accept the input, the ParamData object should be extended with the appropriate fields, and the MATLAB queue handler will need additional functionality. Lastly, the MATLAB starting server file is includes versatility in the path. It initiates the server in the correct path so that the realized system can be put in any folder as part of a bigger system.

```
function postXY(name) {
    var container = jQuery('#'+name+'Container'),
        dragger = jQuery('#'+name+'Dragger'),
        x = dragger.position().left - container.
            position().left - 1,
        y = dragger.position().top - container.
            position().top - 1,
    w = container.width() - dragger.width(),
    h = container.height() - dragger.height();

    var o = {
    'parameter': name,
    'x-coord': x,
    'y-coord': y,
    'width': w,
    'height': h
  }
    postAjax(o);
}
```

Listing 6. One of the Ajax request functions that receives a name as argument.