ABSTRACT

Scholarly Paper: **SYSTEMS ENGINEERING DESIGN AND TRADEOFF ANALYSIS WITH WEB TECHNOLOGY**

Peter Joseph Linnehan, Master of Science, 2014

Directed by: Associate Professor Mark Austin
Department of Civil and Environmental Engineering
and ISR

Abstract. This project proposes a novel approach to the Home Theater Design Problem previously discussed at the University of Maryland College Park. The approach applies a web-based framework to handle the system design requirements, data storage, component-selection, and system configuration.

Last Modified: April 30, 2014

SYSTEMS ENGINEERING DESIGN AND TRADEOFF ANALYSIS WITH WEB TECHNOLOGY

by

Peter Joseph Linnehan

Scholarly Paper
Master of Science in Systems Engineering
2014

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

The importance of systems engineering lies in its ability to take in the functionality and design of the entire system, rather than focus on a specific domain within that system. Systems engineers bridge the gap between technical and business aspects of a project by taking both side's concerns into account. Some typical design side concerns are how well the system should perform, the cost of the system, and how performance of the system will be verified and validated. Management is often concerned with the process necessary to manage the development and how long it will take for the design to reach the market [2]. A good systems engineering design is one that puts both the engineers and the business as whole in a position to succeed.

As the role of software and the reliance on it increases, good software systems engineering designs will become more and more important [3]. This project researches a web-based approach to software system design and the associated technologies. A sample problem is introduced and a demonstration of this approach is implemented.

1.2 Current Practices and Common Issues

When systems engineering is used to implement a software solution, it is equally as important to focus on the back-end system architecture, as it is to focus on the front-end user experience. A well-designed back-end system aids in system scaling, lowering maintenance costs, and making the software more flexible to change with customer requirements, company direction, additional functionality, etc [4].

Unfortunately, some software solutions give preference to a fast time to market and do not focus on a well-designed back-end system. This may work in the short run, but often leads to the integration of heterogeneous technologies that increase system complexity such as the use of multiple programming languages or software packages that were not intended to work concurrently. The results are custom solutions and other work-a-rounds that take the company additional time and effort to develop and maintain. These features of system design represent unnecessary churn and make system maintainability more difficult in the long run. A more effective system design would leverage homogenous technologies and a common structure wherever possible [4]. This is the basis of the software system design researched in this project.

1.3 Project Problem

To demonstrate the advantages to the web-based system design, it will be applied to solve the Home Theater Design Problem. This problem was originally posed by NASA Goddards David Everett and co-workers as an exercise in understanding how requirements should be written and organized for the team-based development of engineering systems. In this problem a customer is looking to buy a home theater system consisting of three separate components: an amplifier, speakers, and a television.

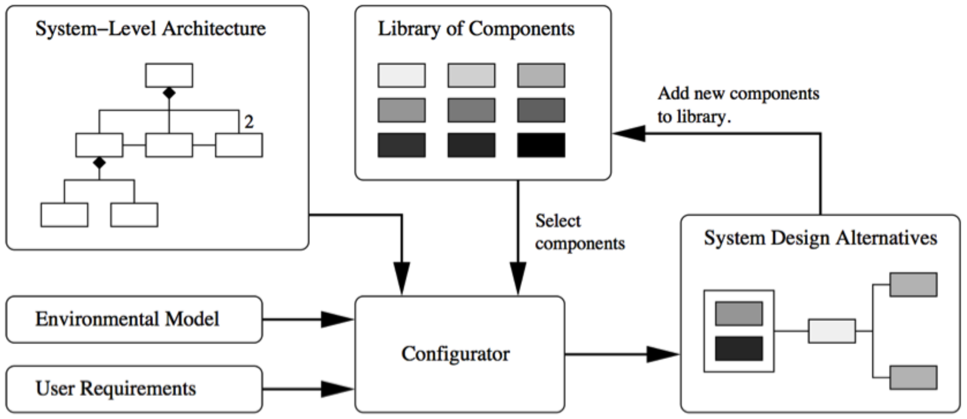


Figure 1.1: Schematic of the component selection design problem [1].

The primary objective in this problem is to create a library of components that can be configured into different system design alternatives. The system design alternatives are based off of a predefined system-level architecture and additional user requirements such as a maximum total cost. For the purposes of this problem, the home theater system architecture is an amplifier, speaker set, and a television.

1.3.1 Previous Work at UMCP

The first major pieces of work done on the Home Theater Design Problem were a series of papers written by Mark Austin, Vimal Mayank, and Natalya Shmunis [7, 6, 5]. These papers were preliminary steps toward ontology-based computing for the verification and validation purposes rather than a direct solution to the posed problem. Its relevance to this project is primarily in providing background and constraints.

A second paper was the Masters Thesis System Engineering Design and Tradeoff Analysis with RDF [24] Graph Models by Nefretiti Nassar under the direction of Mark Austin [22, 23]. Although the author's implementation was successful, the solution required multiple programming languages and a series of data conversions. Consequently a full-scale implementation of the proposed solution would be difficult and impractical.

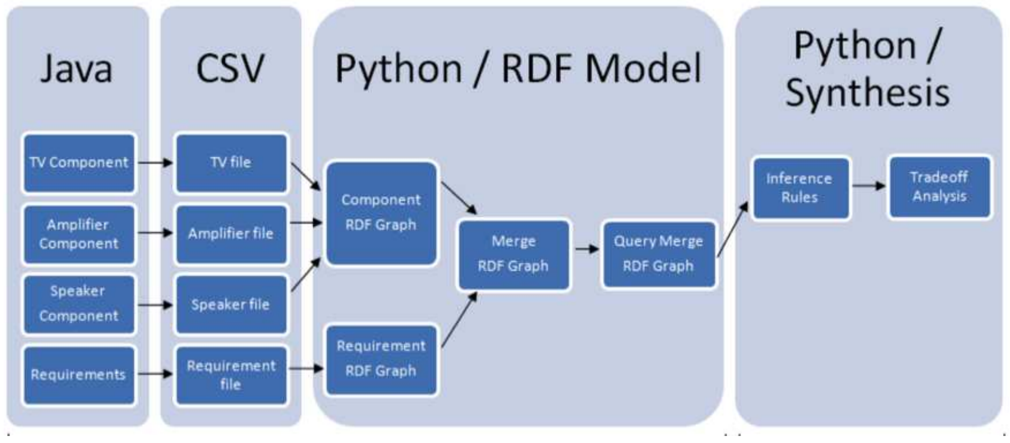


Figure 1.2: Nassar's software pipeline for the Home Theater Design Problem [22].

1.4 Project Scope

This project begins with a technology review of the selected components for the web-based system design, and some of the advantages of the proposed solution. Next, the system design is applied to a manufactured data set of system components to demonstrate the capabilities of the system design in practice. The actual implementation is meant to be more of a proof-of-concept than a full-scale solution, but the advantages of the system design from a systems engineering and functionality standpoint should be clear nonetheless.

Chapter 2

Technology Review and Proposed Architecture

This section discusses the selected technologies that will make up the web-based system architecture, but is not, nor is it intended to be, a definitive claim that the selected technology is superior to all others in all cases. Additionally, the information presented is not exhaustive. The intention is to provide a brief overview of the key features and some of their advantages over other approaches.

2.1 Data

Data are the building blocks of this project, therefore it is critical to choose a data format that is flexible, easy to understand, and easy to work with. Given the web-based nature of the system architecture, the two data models considered were the eXtensible Markup Language (XML) [26] and the JavaScript Object Notation (JSON) [12, 16].

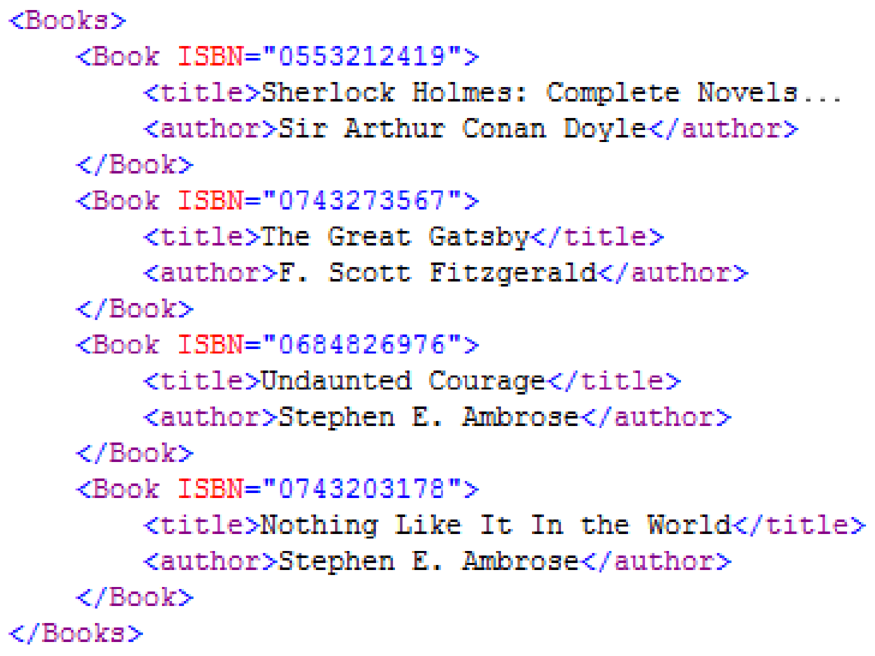
2.1.1 eXtensible Markup Language

The eXtensible Markup Language (XML) is human-readable and machine-readable way to encode data. These data can be extracted by parsing the XML with a computer, which makes XML a useful data-interchange. Although XML is a well-known and widely used format, there are some disadvantages that make it impractical to use for the web-based home theater system architecture.

The first disadvantage associated with XML is its verbosity. Similar to the Hypertext Markup Language (HTML), XML requires opening and closing tags when defining an encoding as seen in Figure 2.1. These tags can quickly increase the size of the document making it more difficult to read and harder to maintain. A second disadvantage is XMLs lack of a well-defined data structure. XML provides an open-ended tree structure that can represent data in a variety of ways. At times, the open-ended data structure can be advantageous or even necessary; however it can also lead to mismatched data structures between parties when they choose to represent the data differently. Lastly, although XML is easy to parse, XML does not map directly to most programming language variables. This further complicates the process of using the data, and for these reasons, XML was not chosen for the data encoding in this project.

2.1.2 JavaScript Object Notation

JSON is described as a lightweight data-interchange format that is easy for humans to read and write as well as being easy for computers to parse and generate. JSON is language independent much like XML, however the conventions to store and organize the data model are familiar to programmers of the C-family of languages. JSON is built on two structures that share a number of similarities with the data structures used in dynamic programming languages. The first structure is

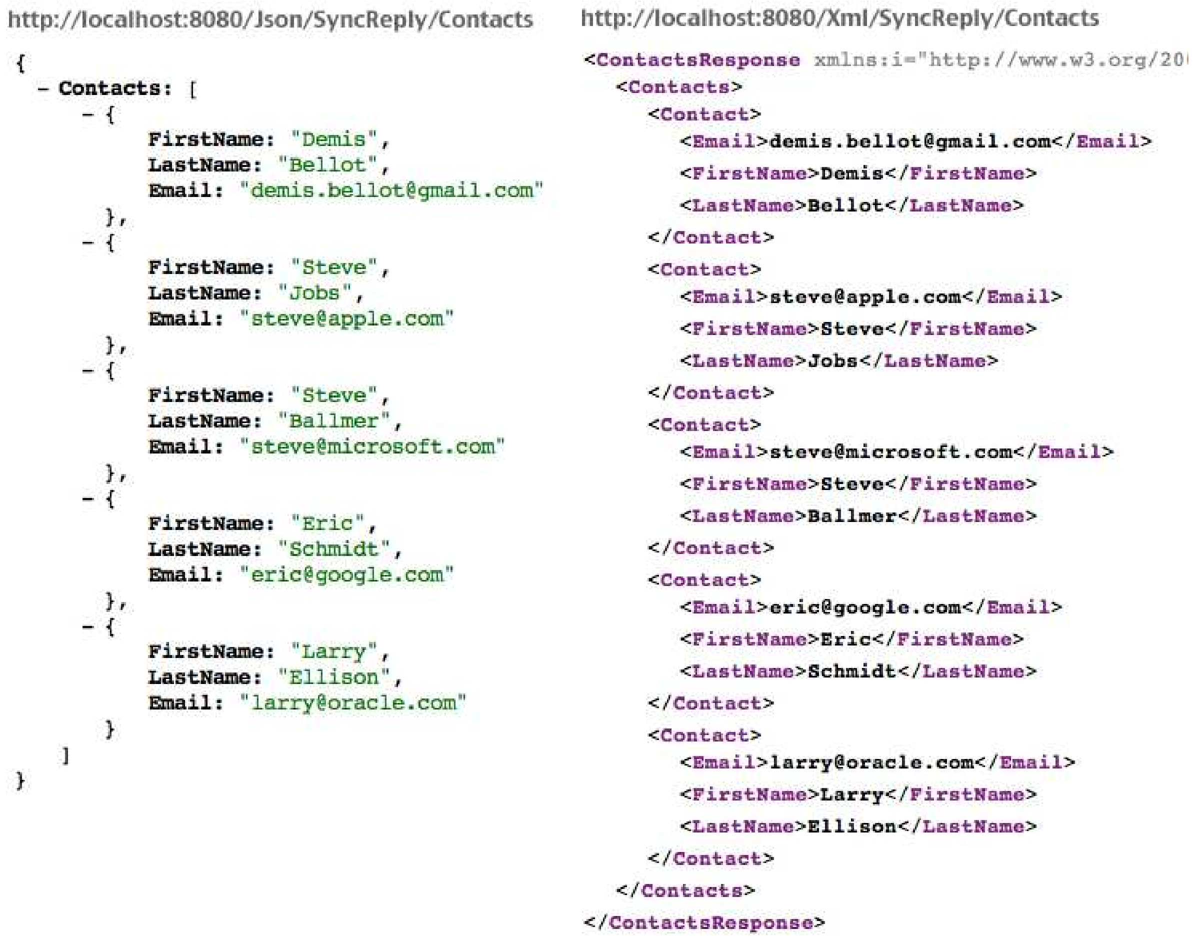


```
<Books>
    <Book ISBN="0553212419">
        <title>Sherlock Holmes: Complete Novels...
        <author>Sir Arthur Conan Doyle</author>
    </Book>
    <Book ISBN="0743273567">
        <title>The Great Gatsby</title>
        <author>F. Scott Fitzgerald</author>
    </Book>
    <Book ISBN="0684826976">
        <title>Undaunted Courage</title>
        <author>Stephen E. Ambrose</author>
    </Book>
    <Book ISBN="0743203178">
        <title>Nothing Like It In the World</title>
        <author>Stephen E. Ambrose</author>
    </Book>
</Books>
```

Figure 2.1: Example XML syntax to encode a group of books [27].

a collection of name/value pairs, which can equivalently be compared to an object, record, struct, dictionary, hash table, keyed list, or associative. The second structure is an ordered list of values, which can be equivalently compared to an array, vector, list or sequence [16].

These two structures provide two distinct benefits when defining a data model. First, JSON provides a more natural data model for programmers and engineers to map new objects to because the data model is one they are already familiar with. Similarly, this structure also helps programmers and engineers work with the data directly with minimal processing. In fact, the close similarities between JSON representations and JavaScript objects enable a near seamless transition between the two technologies. This ease of use is one of the primary drivers for choosing JSON in this system design.


http://localhost:8080/Json/SyncReply/Contacts

```
{
  - Contacts: [
      - {
            FirstName: "Demis",
            LastName: "Bellot",
            Email: "demis.bellot@gmail.com"
        },
      - {
            FirstName: "Steve",
            LastName: "Jobs",
            Email: "steve@apple.com"
        },
      - {
            FirstName: "Steve",
            LastName: "Ballmer",
            Email: "steve@microsoft.com"
        },
      - {
            FirstName: "Eric",
            LastName: "Schmidt",
            Email: "eric@google.com"
        },
      - {
            FirstName: "Larry",
            LastName: "Ellison",
            Email: "larry@oracle.com"
        }
    ]
}
```

http://localhost:8080/Xml/SyncReply/Contacts

```
<ContactsResponse xmlns:i="http://www.w3.org/20
  <Contacts>
    <Contact>
      <Email>demis.bellot@gmail.com</Email>
      <FirstName>Demis</FirstName>
      <LastName>Bellot</LastName>
    </Contact>
    <Contact>
      <Email>steve@apple.com</Email>
      <FirstName>Steve</FirstName>
      <LastName>Jobs</LastName>
    </Contact>
    <Contact>
      <Email>steve@microsoft.com</Email>
      <FirstName>Steve</FirstName>
      <LastName>Ballmer</LastName>
    </Contact>
    <Contact>
      <Email>eric@google.com</Email>
      <FirstName>Eric</FirstName>
      <LastName>Schmidt</LastName>
    </Contact>
    <Contact>
      <Email>larry@oracle.com</Email>
      <FirstName>Larry</FirstName>
      <LastName>Ellison</LastName>
    </Contact>
  </Contacts>
</ContactsResponse>
```

Figure 2.2: JSON representation vs. XML representation [17].

2.2 Back-End Data Storage

The Home Theater system architecture requires a back-end storage system to house the different types of speakers, amplifiers, and televisions. This back-end data store corresponds to the library of components that was indicated in Figure 1.1.

The conventional choice for a back-end database for a web-based system design would be a relational database such as Oracle's MySQL database [21]. Some of the benefits of using MySQL are the extensive user base, the open-source code, and the high performance and stability it offers. MySQL would be a reasonable choice as the back-end solution. However given the choice of JSON as the data structure, an alternate database was chosen.

2.2.1 MongoDB

The back-end database for this system design is the document database MongoDB [19]. MongoDB is what is currently defined as a NoSQL database [18, 20]. At a basic level, NoSQL databases are non-relational databases that are distributed on a cluster of machines rather than on a single machine and work well with large-scale data processing. MongoDB offers a dynamic data structure, as opposed to the more rigid data structure of a relational database, as well as powerful querying and other useful features such as direct mapping to the JSON data format.

MongoDB uses a JSON style data structure named Binary JSON (BSON), which works in much the same way as JSON. The ability to define home theater components in the same format that they will be stored in is a major advantage for this system design. First, the similar structures help simplify the process of adding new components into the existing library of components. Loading JSON representations of the components into MongoDB requires almost no preprocessing of the raw data. Additionally, pulling data out of MongoDB also requires minimal processing as BSON data structure works well with the JavaScript scripting language and modern Internet web browsers that are used for implementing the systems business logic and user-interface respectively.

MongoDBs dynamic, document-based, data storage also plays a key role in the system design as it allows for the back-end data structure to change with minimal refactoring. Home theater components are stored in a MongoDB collection that allows the user to define whatever attributes per component they require. This is important because as technology advances and new components become available, components will have different attributes that arent captured in the current data model or older attributes that are no longer captured. This flexibility helps lower maintenance costs and allows the system design to scale more easily.

2.3 User-Interface

2.3.1 Modern Web Browser

The system design for the home theater solution requires a user interface for the user to set system requirements and to render resulting designs. A modern web browser is a software application for retrieving, presenting, and traversing information resource on the World Wide Web [25]. The web browser has become ubiquitous in our everyday lives so it offers a great setting for both the front-end user and the back-end developer. For the purposes of this project, a modern web browser would be Mozilla's Firefox, Google's Chromium, Apple's Safari, Opera, and Microsoft's Internet Explorer 9.

2.3.2 HyperText Markup Language (HTML)

In addition to the web-browser, the system design requires a web page which the user will see and interact with. The web page that the web browser renders is defined using the HyperText Markup Language (HTML). HTML is the core language of nearly all web content. Most of what is shown on the screen in a browser is described, fundamentally, using HTML. More precisely, HTML is the language that describes the structure and the semantic content of a web document. Content within a Web page is tagged with HTML elements [13].

2.3.3 Cascading Style Sheet (CSS)

CSS is a style sheet language used to describe the presentation of a document written in HTML or XML. CSS describes how the structure element must be rendered on screen. CSS is what is used to provide style (fonts, color, backgrounds, etc.) to a web application such as the one in this project [10]. Although CSS does not provide any functionality to the system design, it is important nonetheless to make the solution user friendly and aesthetically pleasing.

2.3.4 Twitter Bootstrap [8]

Bootstrap is a free collection of tools for creating websites and web applications. It contains HTML and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions [9]. Bootstrap was leveraged in this project to help design the layout of the user-interface.

2.4 JavaScript

JavaScript is a dynamic computer programming language and is most commonly used as part of web browsers. JavaScript can be used to interact with the user, control the browser, communicate asynchronously, alter the document, and function as a server-side programming language [15]. As of 2012 all modern browsers fully support ECMAScript (the scripting language that forms the basis of JavaScript)

[14].

In addition to working well with the web browser, JavaScript has a natural mapping to the JavaScript Object Notation data structure and the Binary JavaScript Object Notation of MongoDB. The ability of JavaScript to work well with the front-end user interface, the back-end database, and the home theater component data is a major advantage. Leveraging JavaScript as the primary programming language for each step in our system design process streamlines the software pipeline and helps abstract out a lot of potential complexity.

2.4.1 jQuery

jQuery is an open source JavaScript library that simplifies interacting with the web-browser. It is used to expand the functionality of the webpage and Twitter Bootstrap requires jQuery to be loaded.

2.5 Document Driven Design

The final piece of technology in the system design is a way to graphically compare different designs on the web page. There are many options for web visualizations such as Google Visualizations, Highcharts JS, Chart.js, and Document-Driven Design (D3). For the purposes of this project, the system designs will be rendered using D3, which is a JavaScript library for manipulating documents based on data that was written by Mike Bostock.

D3 helps bring data to life using HTML, Scalable Vector Graphics (SVG), and CSS. D3s emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combing powerful visualization components and data-driven approach to Document Object Model (DOM) manipulation [11]. Additionally, D3 works well with JSON formatted data and the SVGs build visualizations that automatically grow with the data. This is an important as it enables the Home Theater application to grow as more components are added without having to refactor the initial design. Minimizing the amount of maintenance is a key concern in establishing the system design.

Chapter 3

Home Theater Solution Implementation

In order to prototype the proposed system design, the technologies discussed in chapter two were applied to solve the Home Theater Design Problem. The overall purpose of this problem was to use a library of components and a set of home theater system requirements to configure possible designs.

3.1 Solution Overview

The proposed solution is broken into two subsystems: back-end and front-end. The back-end subsystem handles the data structure and data storage, where as the front-end subsystem handles the home theater system configurations and data rendering. The complete process pipeline can be seen in Figure 3.1.

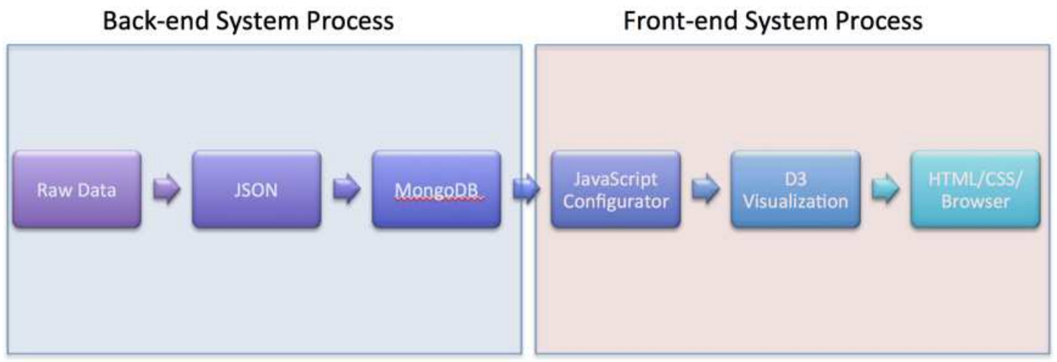


Figure 3.1: Front- and Back-end software subsystems for Home Theater Design Problem implementation.

3.2 Back-end Sytem Process

This section focuses on the three main processes of the back-end subsystem, and illustrated on the left-hand side of Figure 3.1.

3.2.1 Raw Data

The first step in building out the solution was to collect raw data about the system components. For this project, the component data came from Nassar's Masters Thesis [22]. Data for the television, amplifier and speaker components is shown in Tables 3.1 through 3.3, respectively.

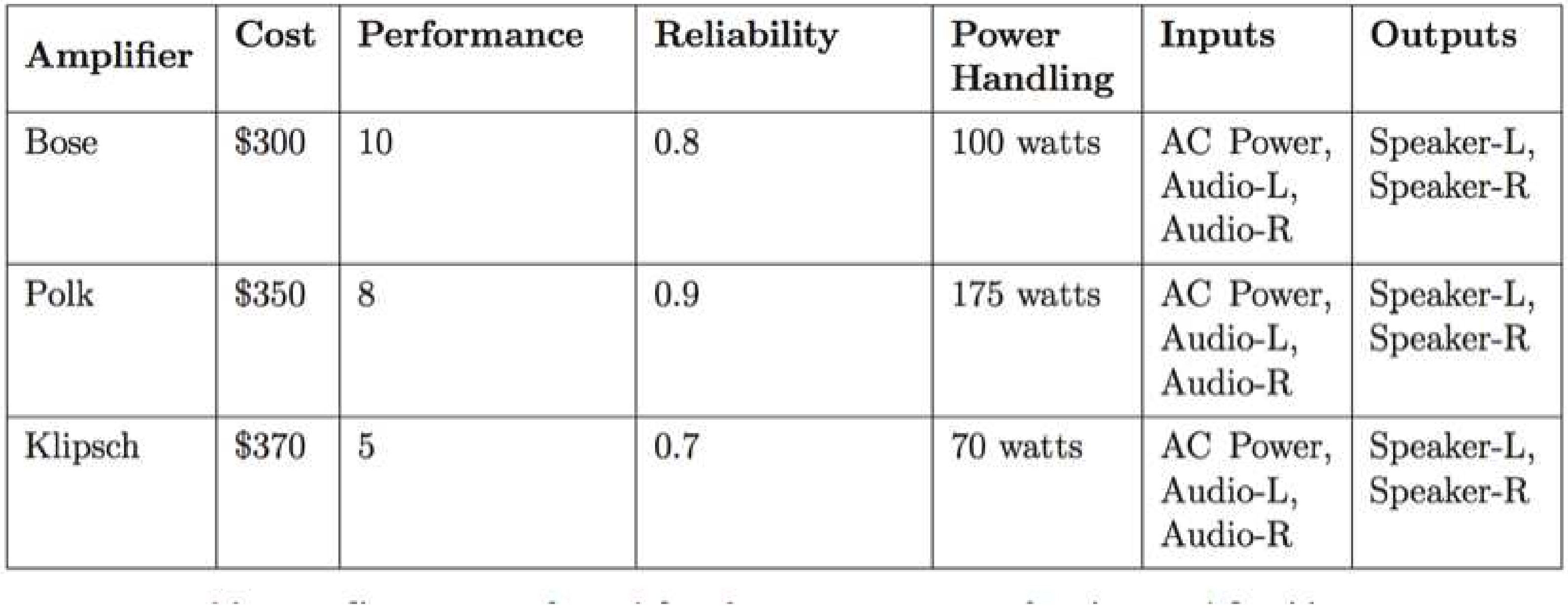
There are three of each type of component for a total of nine components. Each component is associated with a price and other various attributes. This demonstration focuses on cost, performance, and reliability.

3.2.2 JSON Representation

The next step in our system solution was to convert the raw component data into a JSON representation. The JSON representation chosen for this demonstration was one of many possible representations. It may not be the optimal representation for every circumstance, however it worked well for the purposes of the demonstration.

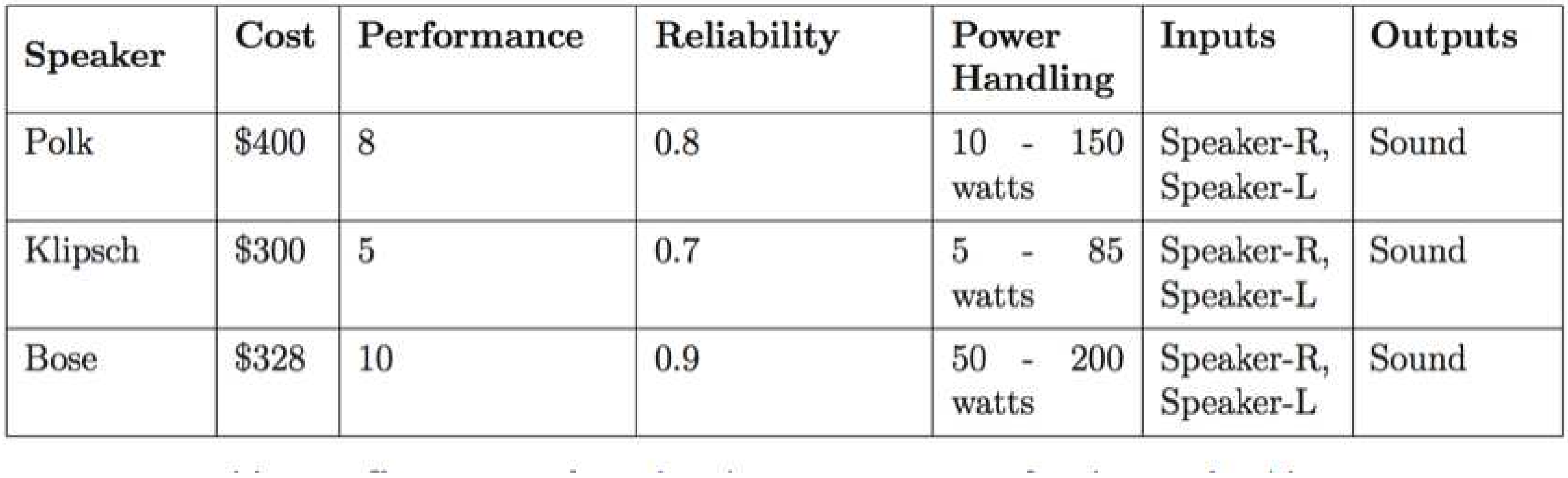
TV	Cost	P	R	Height	Width	Thickness	Weight	Inputs	Outputs
LG	$1300	5	0.7	30.8 in	50.6 in	1.2 in	48.7 lbs	AC Power, HDMI, Video, Audio-L, Audio-R, Antenna/Cable, LAN, USB	Audio-L, Audio-R, Headphones
Samsung	$1650	8	0.8	29.0 in	49.3 in	1.2 in	35.7 lbs	AC Power, HDMI, Video, Audio-L, Audio-R, Antenna/Cable, LAN, USB, Ex-Link	Audio-L, Audio-R, Headphones
Sony	$1200	10	0.9	30.4 in	50.0 in	1.6 in	44.5 lbs	AC Power, HDMI, Video, Audio-L, Audio-R, Antenna/Cable, LAN, USB	Audio-L, Audio-R, Headphones

Table 3.1: Generation of television design components for the television library. Legend: P = performance, R = reliability.



Amplifier	Cost	Performance	Reliability	Power Handling	Inputs	Outputs
Bose	$300	10	0.8	100 watts	AC Power, Audio-L, Audio-R	Speaker-L, Speaker-R
Polk	$350	8	0.9	175 watts	AC Power, Audio-L, Audio-R	Speaker-L, Speaker-R
Klipsch	$370	5	0.7	70 watts	AC Power, Audio-L, Audio-R	Speaker-L, Speaker-R

Table 3.2: Generation of amplifier design components for the amplifier library.



Speaker	Cost	Performance	Reliability	Power Handling	Inputs	Outputs
Polk	$400	8	0.8	10 - 150 watts	Speaker-R, Speaker-L	Sound
Klipsch	$300	5	0.7	5 - 85 watts	Speaker-R, Speaker-L	Sound
Bose	$328	10	0.9	50 - 200 watts	Speaker-R, Speaker-L	Sound

Table 3.3: Generation of speaker design components for the speaker library.

3.2.2.1 Television Component Data Structure

The data structure for the television component is shown in Tables 3.4 and 3.5. Table 3.4 is a conceptual representation of the television component broken into its subsequent key : value pairs. The 'Value' columns represent the data type associated with each 'Key' column, or, in the case of the television inputs and outputs, represent the special situation of embedded data. The television inputs and outputs themselves can be represented uniquely as objects; therefore to capture all the different types, the television inputs value and outputs value is actually a container array of their respective objects.

Table 3.5 is a direct mapping of the raw LG brand television data to JSON. This table represents a single JSON object with two embedded arrays of objects for the television's inputs and outputs.

Key	**Value**	**Key**	**Value**
Component	String		
Brand	String		
Cost	Integer		
Performance	Integer		
Reliability	Integer Fraction		
Height	Integer Fraction		
Width	Integer Fraction		
Thickness	Integer Fraction		
Weight	Integer Fraction		
Inputs	Array:		
		Type	String
		Quantitity	Integer
Outputs	Array:		
		Type	String
		Quantitity	Integer

Table 3.4: Key-Value pairs of the JSON television representation

```
{
     "component" : "television",
     "brand" : "lg",
     "cost" : 1300,
     "performance" : 5,
     "reliability" : 0.7,
     "height" : 30.8,
     "width" : 50.6,
     "thickness" : 1.2,
     "weight" : 48.7,
     "inputs" : [
          {
               "type" : "ac power",
               "quantity" : 1
          },
          {
               "type" : "hdmi",
               "quantity" : 1
          },
          {
               "type" : "video",
               "quantity" : 1
          }
     ],
     "outputs" : [
          {
               "type" : "audio-l",
               "quantity" : 1
          },
          {
               "type" : "audio-r",
               "quantity" : 1
          },
          {
               "type" : "headphones",
               "quantity" : 1
          }
     ]
}
```

Table 3.5: Sample television component in JSON format

3.2.2.2 Amplifier Component Data Structure

The data structure for the amplifier component is shown in Tables 3.6 and 3.7. Like the television component data tables, Table 3.6 is a conceptual representation of the amplifier component broken into its subsequent key : value pairs. The 'Value' columns represent the data type associated with each 'Key' column, or, in the case of the amplifier inputs and outputs, represent the special situation of embedded data. The amplifier inputs and outputs themselves can be represented uniquely as objects; therefore to capture all the different types, the amplifier inputs value and outputs value is actually a container array of their respective objects.

Table 3.7 is a direct mapping of the raw Bose brand amplifier data to JSON. This table represents a single JSON object with two embedded arrays of objects for the amplifier's inputs and outputs.

Key	**Value**	**Key**	**Value**
Component	String		
Brand	String		
Cost	Integer		
Performance	Integer		
Reliability	Integer Fraction		
Power Handling	Integer		
Inputs	Array:		
		Type	String
		Quantitity	Integer
Outputs	Array:		
		Type	String
		Quantitity	Integer

Table 3.6: Key-Value pairs of the JSON amplifier representation

3.2.2.3 Speaker Component Data Structure

The data structure for the speaker component is shown in Tables 3.8 and 3.9. Like the other component data tables, Table 3.8 is a conceptual representation of the speaker component broken into its subsequent key : value pairs. The 'Value' columns represent the data type associated with each 'Key' column, or, in the case of the speaker Inputs, represent the special situation of embedded data. The speaker inputs themselves can be represented uniquely as objects; therefore to capture all the different types, the speaker inputs value is actually a container array of its respective objects.

Table 3.9 is a direct mapping of the raw Klipsch brand speaker data to JSON. This table represents a single JSON object with one embedded array of objects for

```
{
     "component" : "amplifier",
     "brand" : "bose",
     "cost" : 300,
     "performance" : 10,
     "reliability" : 0.8,
     "power_handling" : 100,
     "inputs" : [
          {
               "type" : "ac power",
               "quantity" : 1
          },
          {
               "type" : "audio-l",
               "quantity" : 1
          },
          {
          "type" : "audio-r",
          "quantity" : 1
          }
     ],
     "outputs" : [
          {
               "type" : "speaker-l",
               "quantity" : 1
          },
          {
               "type" : "speaker-r",
               "quantity" : 1
          }
     ]
}
```

Table 3.7: Sample amplifier component in JSON format

the speaker's inputs.

Key	**Value**	**Key**	**Value**
Component	String		
Brand	String		
Cost	Integer		
Performance	Integer		
Reliability	Integer Fraction		
Power Handling Min	Integer		
Power Handling Max	Integer		
Inputs	Array:		
		Type	String
		Quantitity	Integer

Table 3.8: Key-Value pairs of the JSON speaker representation

```
{
     "component" : "speaker",
     "brand" : "klipsch",
     "cost" : 300,
     "performance" : 5,
     "reliability" : 0.7,
     "power_handling_min" : 5,
     "power_handling_max" : 85,
     "inputs" : [
          {
               "type" : "speaker-r",
               "quantity" : 1
          },
          {
               "type" : "speaker-l",
               "quantity" : 1
          }
     ],
     "outputs" : [ ]
}
```

Table 3.9: Sample speaker component in JSON format

3.2.3 MongoDB

3.2.3.1 Schema Definition

With the component data defined as JSON, the next step was to create a BSON schema representation in MongoDB. Once again, the schema chosen for this demonstration was just one of many possible representations.

Defining a schema in MongoDB is a straightforward process once the data have been defined in JSON. The one significant change made to the component JSON representations was that each component was stored in a single array. Thus, instead of having nine separate components, there was one array with nine indices, each of which is assigned to a component. This was necessary because the component array now represents a MongoDB collection. A MongoDB collection is synonymous with a table in a relational database. The collection is what stores each instance of a particular object. The complete MongoDB schema for the system is:

- **Database Name**: homeTheater
    - **Collection Name**: components
        - **Objects**: Each of the nine components

3.2.3.2 Data Ingest

Once the database and collection are defined, the next step was to import the JSON component representations directly into MongoDB. MongoDB does not require the user to define a schema prior to import because the schema will be inferred from how the components are represented in JSON. This makes the importing process extremely easy requiring only one command from the command line:

```
$ mongoimport \
     --db homeTheaterComponents \
     --collection components \
     --type json \
     --jsonArray \
     --file homeTheaterSystemComponents.json
```

Table 3.10: Command line tool for importing data to MongoDB

MongoImport is a tool provided with the MongoDB installation. The db option selects the database to import to, collection indicates the collection in that database, type is the type of data thats being imported, jsonArray is an option indicating that the data are in an array, and the file option points to the raw JSON representation. After running this command a successful importation will look like the following:

```
connected to: 127.0.0.1
Sun Apr 20 15:52:56.983 imported 9 objects
```

Table 3.11: Ouput after successfully importing JSON data

3.2.3.3 Data Export

In order for the front-end to work with the data, it needs to have access to the data from the database. For the purposes of this demonstration, the data was exported into a JSON flat file, which was used for the Representational State Transfer (REST) calls from the front-end.

Exporting the data is also a straightforward process with MongoDB:

```
$ mongo localhost/homeTheaterComponents mongoScript.js >> test.json
```

Table 3.12: Command line statement to export a JSON flat file

This command runs a JavaScript script file, mongoScript (Appendix A), on the database homeTheaterComponents that is running locally on localhost. MongoScript queries the database collection components and returns each component in the collection as a JSON array.

3.3 Front-end System Process

As illustrated on the right-hand side of Figure 3.1, the front-end system design corresponds to a three-stage process: (1) JavaScript Configuration, (2) D3 Visualization and (3) HTML/CSV Browser. For this demonstration, the home theater system design was assumed to be one of each of the three types of components. There were no additional user parameters such as max cost or additional design parameter such as compatibility checks between components.

3.3.1 JavaScript Logic

The first step of the front-end process was to access the available components from the component library and configure the possible designs. JavaScript was used to implement all the business logic associated with grabbing the component library, parsing through each component, and configuring the possible designs.

JavaScript and jQuery were used to access the exported JSON file from the MongoDB library of components. Three container arrays were defined to hold each type of component: televisions, amplifiers, and speakers. JavaScript looped through each JSON object in the components array assigning each component to its associated container array. The code in Table 3.13 performed this process and created JavaScript objects for each component with associated properties taken from the component library.

Once JavaScript has completed parsing the component library, it passes the three component container arrays to a function that configures the possible designs. The system configuration was implemented using a series of for-loops that go through each container array adding a component, calculating the associated properties, and pushing the system design object into a system design container array. Table 3.14 is the excerpt of code that performs this process.

```
/*
     JQuery function that goes to the URL in the first argument and
     returns the json data
*/
$.getJSON("data/homeTheaterSystemComponents.json", function(data) {
     var televisions =     []; // Holds information on each television
     var speakers =        []; // Holds information on each speaker
     var amplifiers =       []; // Holds information on each amplifier

     /*
          cycles though each json object
     */
     $.each(data, function(key, val) {
     /*
          "if" statements assigns each component’s information
          into its respective container array
     */
     if (val.component == "television") { televisions.push(
               {
                    "brand" : val.brand,
                    "cost" : val.cost,
                    "performance" : val.performance,
                    "reliability" : val.reliability
               }
          );
     };
     if (val.component == "speaker") { speakers.push(
               {
                    "brand" : val.brand,
                    "cost" : val.cost,
                    "performance" : val.performance,
                    "reliability" : val.reliability
               }
          );
     };
     if (val.component == "amplifier") {amplifiers.push(
               {
                    "brand" : val.brand,
                    "cost" : val.cost,
                    "performance" : val.performance,
                    "reliability" : val.reliability
               }
          );
     };
});
```

Table 3.13: JavaScript logic for parsing throught the exported JSON flat file.

```
function buildSystems(televisions, amplifiers, speakers) {
     var sum = 0;   // Total system design cost
     var performance = 0; // Total system performance
     var reliability = 0;  // Total system reliability
     var counter = 1; // Used to label design names

     for (var i = televisions.length - 1; i >= 0; i--) { // Loop through television container array
          for (var j = amplifiers.length - 1; j >= 0; j--) { // Loop through amplifier container array
               for (var k = speakers.length - 1; k >= 0; k--) { // Loop through speaker container array
                    sum = televisions[i].cost + amplifiers[j].cost + speakers[k].cost;
                    performance = (televisions[i].performance + amplifiers[j].performance + speakers[k].performance) / 3;
                    reliability = (televisions[i].reliability + amplifiers[j].reliability + speakers[k].reliability) / 3;
                    systemDesigns.push( // Add system design object to system design container array
                         {
                              "name" : "design " + counter,
                              "cost" : sum,
                              "performance" : performance,
                              "reliability" : reliability,
                              "television" : televisions[i].brand,
                              "amplifier" : amplifiers[j].brand,
                              "speaker" : speakers[k].brand
                         }
                    );
                    counter++;
               };
          };
     };
};
```

Table 3.14: JavaScript logic for configuring system designs.

With this function complete, the array of system design objects is now available to be passed to the web page and rendered to the user through the browser.

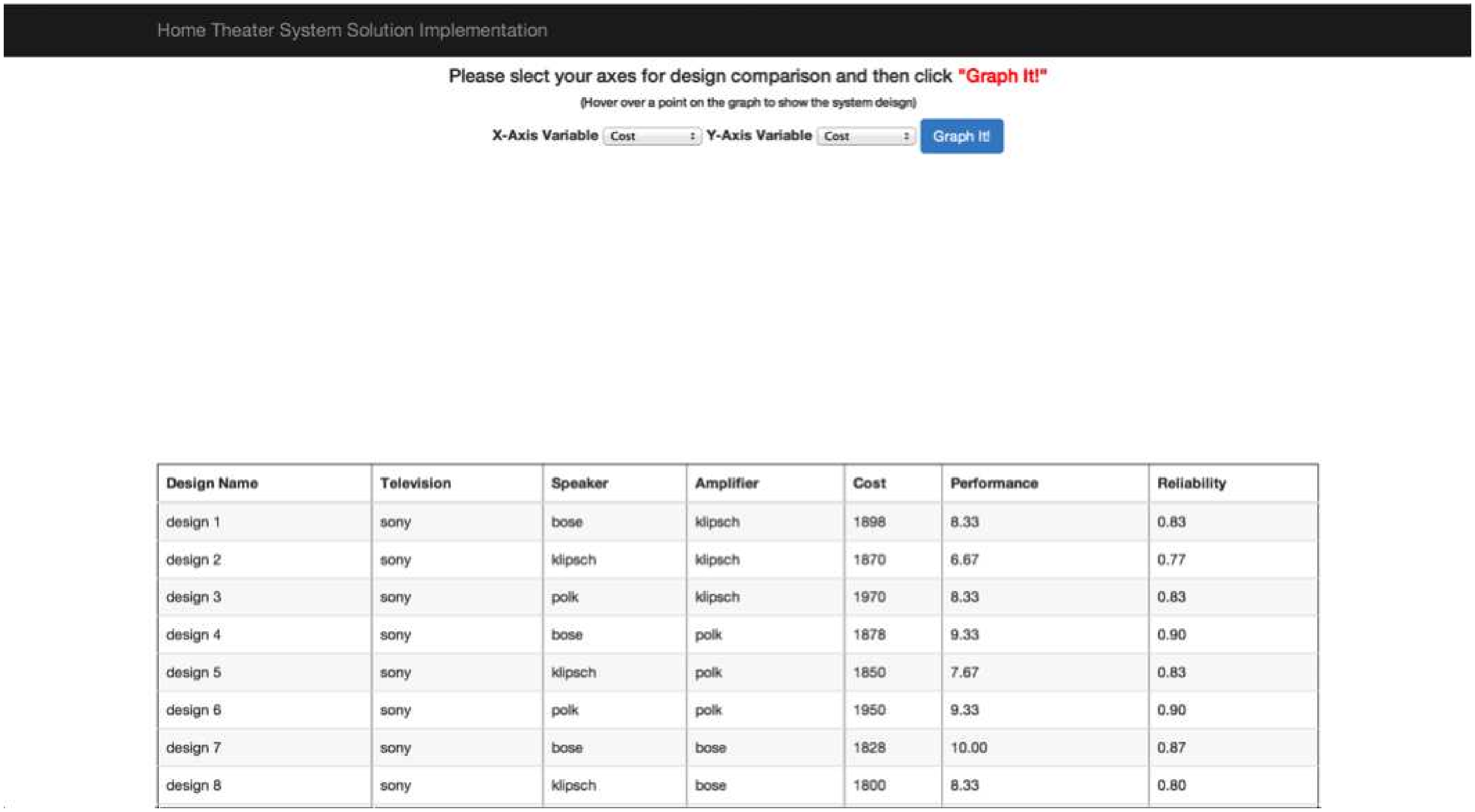
3.3.2 Web-page Layout

A sample of the web-page layout for this demonstration can be seen in Figure 3.2. The layout follows a simple design and provides two primary functions. The first is to display a table of all the possible design alternatives including the associated television, speaker, amplifier, cost, performance, and reliability. A sample of this table is shown in Figure 3.3. The second function of the web is to display graphical representations of the design alternatives.

The graphical representations were rendered using D3. D3 was passed the array of system design objects created by the JavaScript logic. When the Graph It! button is clicked, D3 creates a scatter plot based on the x and y variables the user selects. The user can select between the following choices (or their inverses):

- **Cost Vs. Performance** - See Figure 3.4.
- **Cost Vs. Reliability** - See Figure 3.5.
- **Reliability Vs. Performance** - See Figure 3.6.

The user has the freedom to adjust the x and y parameters freely to see if any designs are superior to all others. To determine which design belongs to which point, the user must only hover over that point and the associated design will appear.



Home Theater System Solution Implementation

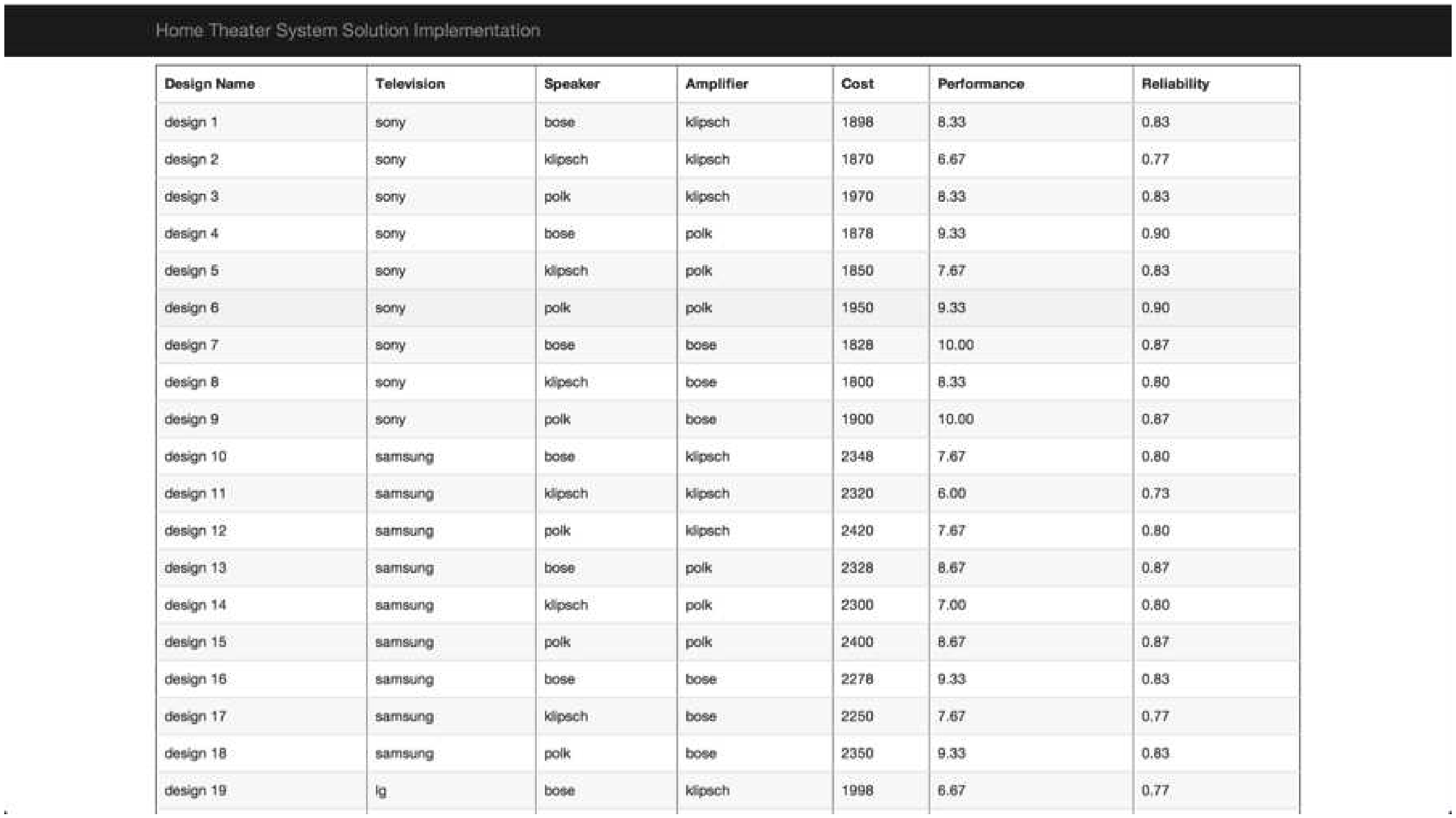
Please slect your axes for design comparison and then click "Graph It!"

(Hover over a point on the graph to show the system deisgn)

X-Axis Variable: Cost — Y-Axis Variable: Cost — Graph It!

Design Name	Television	Speaker	Amplifier	Cost	Performance	Reliability
design 1	sony	bose	klipsch	1898	8.33	0.83
design 2	sony	klipsch	klipsch	1870	6.67	0.77
design 3	sony	polk	klipsch	1970	8.33	0.83
design 4	sony	bose	polk	1878	9.33	0.90
design 5	sony	klipsch	polk	1850	7.67	0.83
design 6	sony	polk	polk	1950	9.33	0.90
design 7	sony	bose	bose	1828	10.00	0.87
design 8	sony	klipsch	bose	1800	8.33	0.80

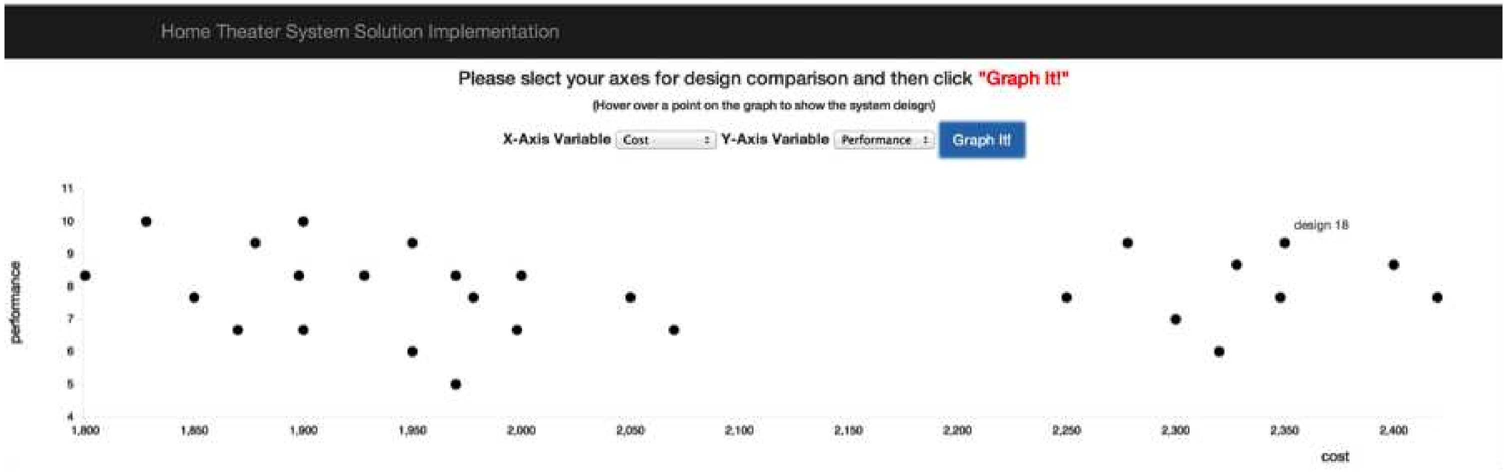
Figure 3.2: Partial view of the user-interface.



Home Theater System Solution Implementation

Design Name	Television	Speaker	Amplifier	Cost	Performance	Reliability
design 1	sony	bose	klipsch	1898	8.33	0.83
design 2	sony	klipsch	klipsch	1870	6.67	0.77
design 3	sony	polk	klipsch	1970	8.33	0.83
design 4	sony	bose	polk	1878	9.33	0.90
design 5	sony	klipsch	polk	1850	7.67	0.83
design 6	sony	polk	polk	1950	9.33	0.90
design 7	sony	bose	bose	1828	10.00	0.87
design 8	sony	klipsch	bose	1800	8.33	0.80
design 9	sony	polk	bose	1900	10.00	0.87
design 10	samsung	bose	klipsch	2348	7.67	0.80
design 11	samsung	klipsch	klipsch	2320	6.00	0.73
design 12	samsung	polk	klipsch	2420	7.67	0.80
design 13	samsung	bose	polk	2328	8.67	0.87
design 14	samsung	klipsch	polk	2300	7.00	0.80
design 15	samsung	polk	polk	2400	8.67	0.87
design 16	samsung	bose	bose	2278	9.33	0.83
design 17	samsung	klipsch	bose	2250	7.67	0.77
design 18	samsung	polk	bose	2350	9.33	0.83
design 19	lg	bose	klipsch	1998	6.67	0.77

Figure 3.3: Partial view of the user-interface table of designs.

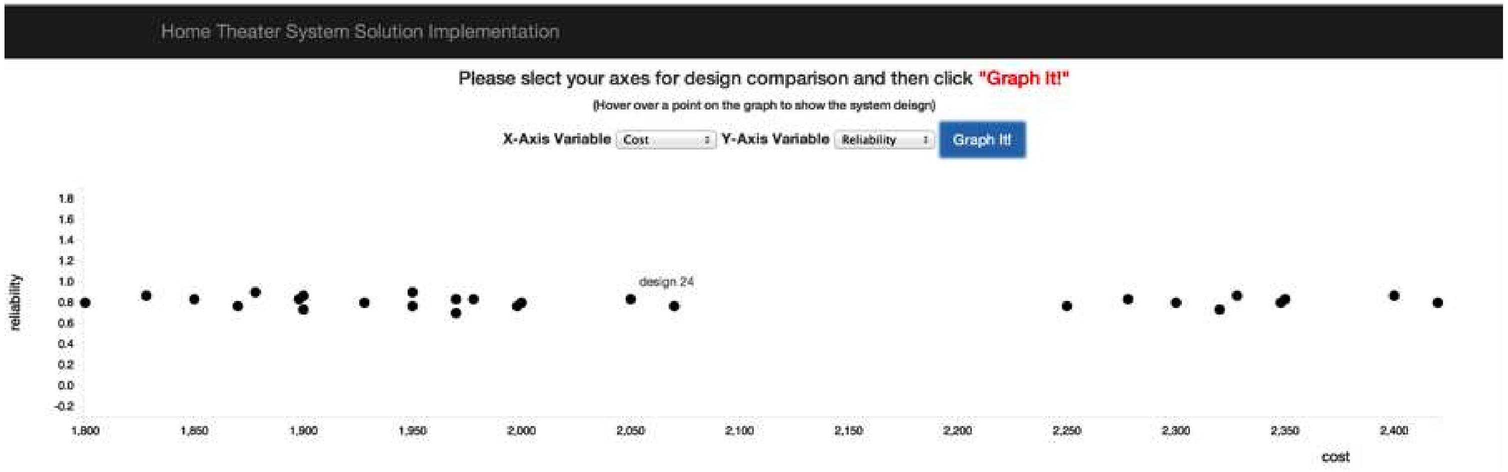


Please slect your axes for design comparison and then click "Graph It!"

(Hover over a point on the graph to show the system deisgn)

X-Axis Variable Cost Y-Axis Variable Performance

Figure 3.4: Cost vs. Performance visualization

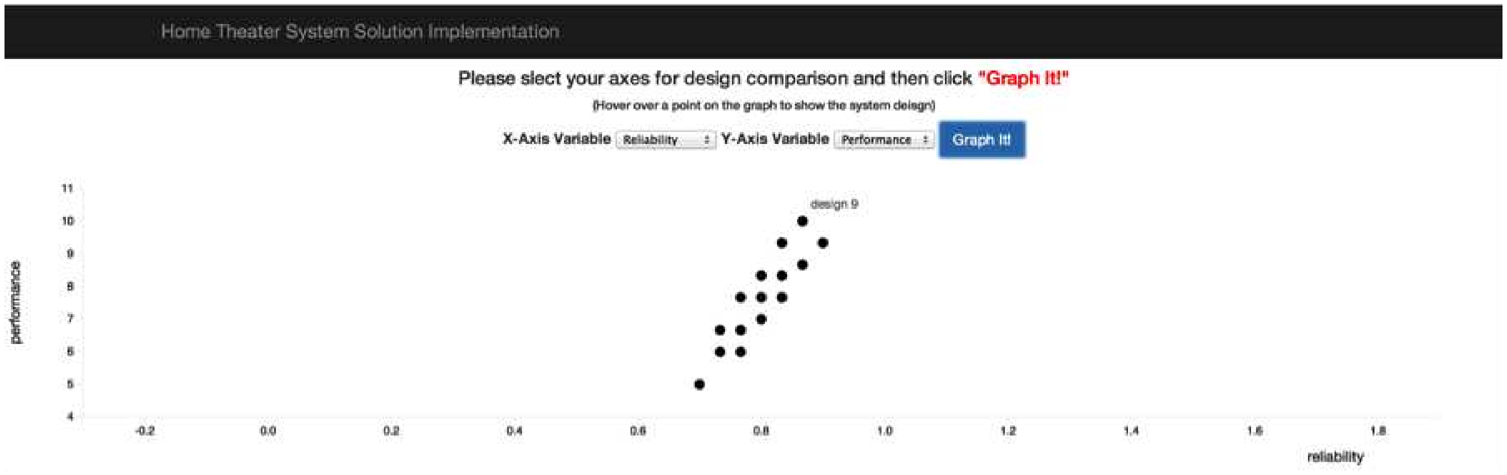

Home Theater System Solution Implementation

Please slect your axes for design comparison and then click "Graph It!"

(Hover over a point on the graph to show the system deisgn)

X-Axis Variable: Cost Y-Axis Variable: Reliability

Figure 3.5: Cost vs. Reliability visualization



Home Theater System Solution Implementation

Please slect your axes for design comparison and then click "Graph It!"

(Hover over a point on the graph to show the system deisgn)

X-Axis Variable Reliability Y-Axis Variable Performance Graph It!

Figure 3.6: Reliability vs. Performance visualization

Chapter 4

Conclusions and Future Work

4.1 Conclusions

Although this solution is not yet a full-scale solution, the proposed system design is a major step forward for the Home Theater Design and other similiar problems. The system design proposed lays the ground-work for creating a software solution capable of dealing with organizing and executing system requirements. Some of the major benefits of this system design are as follows:

- The back-end data and data storage provide the front-end developer with a familiar data structure that integrates well with the front-end user interface
- The JavaScript language and system design object representations make it easy to apply additional logic to meet expanding customer requirements
- The front-end provides a familiar user-interface and easy customization depending on user requirements. Additionally, the browser effectively hides all the computation and complexity from the end-user making for a better experience
- The system design is scalable. Adding additional components only requires a JSON representation; then those components can be added directly to the database.

- The system design is done primarily in JavaScript or a variant of JavaScript. Thus simplifying development and maintenance

4.2 Limitations and Future Work

The proposed system design and implementation provides an effective proof-of-concept towards creating a production quality solution for the Home Theater System Design Problem. The following list are some of limitations associated with the current solution and potential for future improvements:

- Exporting a JSON file of components from MongoDB is not a practical or realistic solution.
  - Instead, another system deisgn layer should be implemented in between the front-end and back-end subsystems. This layer should allow the front-end to query and retrieve data from the back-end database directly.
- The raw data does not accurately represent available data about actual home theater components.
  - To make the raw data more realistic, the raw data should be adjusted to model actual speaker, television and amplifier models. Also, a more defined process to determine attribute scores should also be a focus for improving the raw data.
- The solution and user-interface would be more realistic if it allowed for user to set parameters based on their own preferences. Additionally, compatability

checks between components also needs to be a point of focus.

  - This could be achieved by adding more user controls to the solution web-page. These controls could set additional parameters on viable system designs. The additional parameters could correspond to system design attributes that could be evaluated using JavaScript.

- The table and graph visualizations are useful, but not ideal for helping the user determine the best solution.

  - Introducing a sortable and searchable table of designs would be an easy way to create a better experience for the user. In addition to the graphs, JavaScript could also be used to automatically list out the best design(s) for the user.

- Using a local version of this solution also is not optimal given its compatability with the Internet.

  - Hosting the solution online would be a major step towards a full-scale solution.

- The brute force logic for configuring the system designs is not a scaleable solution.

  - Some research could be done into more efficient algorithms or ways of making the process more efficient, such as cacheing system designs rather than having them repeatedly calculated every time web-page is refreshed.

Bibliography

[1] Austin M.A. *Component Selection Design Process with Tradeoff Analysis*. Reading from Lecture Notes for ENSE622/ENPM642 Information-Centric Systems Engineering, Institute for Systems Research, University of Maryland, College Park, MD 20742, 2012.

[2] Austin M.A. *Our Definition of Systems Engineering*. Reading from Lecture Notes for ENSE621/ENPM641 Systems Concepts, Issues and Processes, Institute for Systems Research, University of Maryland, College Park, MD 20742, 2012.

[3] Austin M.A. *Large Computer Programs*. Reading from Lecture Notes for ENCE 688R: Engineering Software Development in Java, Department of Civil and Environmental Engineering, University of Maryland, College Park, MD 20742, 2013.

[4] Austin M.A. *Orchestration of Good Design Solutions*. Reading from Lecture Notes for ENCE 688R: Engineering Software Development in Java, Department of Civil and Environmental Engineering, University of Maryland, College Park, MD 20742, 2013.

[5] Austin M.A., Mayank V., and Shmunis N. Paladin Software Toolset. Institute for Systems Research, 2003. For more information, see http://www.isr.umd.edu/paladin/.

[6] Austin M.A., Mayank V., and Shmunis N. Ontology-Based Validation of Connectivity Relationships in a Home Theater System. *International Journal of Intelligent Systems*, 21(10):1111–1125, October 2006.

[7] Austin M.A., Mayank V., and Shmunis N. PaladinRM: Graph-Based Visualization of Requirements Organized for Team-Based Design. *Systems Engineering: The Journal of the International Council on Systems Engineering*, 9(2):129–145, May 2006.

[8] Twitter Bootstrap – Twitter Inc., See http://getbootstrap.com/, (Accessed April 24, 2014).

[9] About Twitter Bootstrap – Wikipedia Foundation, See http://en.wikipedia.org/wiki/Bootstrap(frontendframework), (Accessed April 24, 2014).

[10] Cascading Style Sheet – Mozilla Developer Network, See https://developer.mozilla.org/en-US/docs/Web/CSS, (Accessed April 24, 2014).

[11] Document Driven Design – Mike Bostock, See hhttp://d3js.org/, (Accessed April 24, 2014).

[12] ECMA International. *Standard ECMA-404 The JSON Data Interchange Format*. ECMA International, Rue du Rhone 114 CH-1204 Geneva, 2013.

[13] HyperText Markup Language – Mozilla Developer Network, See https://developer.mozilla.org/en-US/docs/Web/HTML, (Accessed April 24, 2014).

[14] JavaScript – Mozilla Developer Network, See https://developer.mozilla.org/en-US/docs/Web/JavaScript, (Accessed April 24, 2014).

[15] About JavaScript – Wikipedia Foundation, See http://en.wikipedia.org/wiki/JavaScript, (Accessed April 24, 2014).

[16] Introducing JSON. See http://www.json.org, (Accessed April 24, 2014).

[17] JSON syntax vs. XML syntax. See http://www.auroraedialliance.com/Portals/126065/images/JsonXmlResults.png, (Accessed April 24, 2014).

[18] NoSQL Databases Explained – MongoDB, Inc. See http://www.mongodb.com/nosql-explained, (Accessed April 24, 2014).

[19] Introduction to MongoDB – MongoDB, Inc., See http://www.mongodb.org/about/introduction/, (Accessed April 24, 2014).

[20] MongoDB, Inc. *Top 5 Considerations When Evaluating NoSQL Databases*. MongoDB, Inc., 2013.

[21] MySQL Relational Databse – Oracle. See http://www.mysql.com/why-mysql/, (Accessed April 24, 2014).

[22] Nassar N. *Systems Engineering Design and Tradeoff Analysis with RDF Graph Models*. Masters Thesis in Systems Engineering, Institute for Systems Research, University of Maryland, College Park, MD 20742, 2012.

[23] Nassar N. and Austin M.A. Model-Based Systems Engineering Design and Trade-Off Analysis with RDF Graphs. In *Conference on Systems Engineering Research (CSER '13')*, pages 216–225. Procedia Computer Science, 2013.

[24] An Introduction to RDF and the Jena RDF API, Apache Jena – The Apache Software Foundation, (Accessed April 3, 2014).

[25] Internet Web Browser – Wikipedia Foundation, See http://en.wikipedia.org/wiki/WebBrowser, (Accessed April 24, 2014).

[26] eXtensible Markup Language (XML). See http://www.w3.org/XML. 2004.

[27] Example XML code. See http://www.kirupa.com/, (Accessed April 24, 2014).

Appendix A: Mongo Export to Flat File Script

This appendix contains the JavaScript code for exporting Mongo data to a JSON flat file.

```
/*
     mongoScript.js
*/
     var x = db.components.find(); // Returns an iterator of components
     print('['); // Start JSON array
     while (x.hasNext()) { // Loop through the results
          x.forEach(function(doc) { // For each result:
          doc._id=doc._id.valueOf(); // Convert the BSON ID to a String
          print(tojson(doc)); // Print as JSON
          print(",") // Appnd with a comma
     });
}
print(']'); // Close the JSON array
```