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

Advisory Committee:
Associate Professor Mark Austin, Chair/Advisor
Second Reader: Professor Ray Adomaitis
# Table of Contents

List of Figures iv

1 Introduction 1
   1.1 Problem Statement 1
   1.2 Current Practices and Common Issues 2
   1.3 Project Problem 2
      1.3.1 Previous Work at UMCP 3
   1.4 Project Scope 4

2 Technology Review and Proposed Architecture 6
   2.1 Data 6
      2.1.1 eXtensible Markup Language 6
      2.1.2 JavaScript Object Notation 7
   2.2 Back-End Data Storage 10
      2.2.1 MongoDB 10
   2.3 User-Interface 12
      2.3.1 Modern Web Browser 12
      2.3.2 HyperText Markup Language (HTML) 12
      2.3.3 Cascading Style Sheet (CSS) 13
      2.3.4 Twitter Bootstrap [8] 13
   2.4 JavaScript 13
      2.4.1 jQuery 14
   2.5 Document Driven Design 14

3 Home Theater Solution Implementation 16
   3.1 Solution Overview 16
   3.2 Back-end System Process 17
      3.2.1 Raw Data 17
      3.2.2 JSON Representation 17
         3.2.2.1 Television Component Data Structure 21
         3.2.2.2 Amplifier Component Data Structure 24
         3.2.2.3 Speaker Component Data Structure 25
      3.2.3 MongoDB 28
         3.2.3.1 Schema Definition 28
         3.2.3.2 Data Ingest 29
         3.2.3.3 Data Export 30
   3.3 Front-end System Process 31
      3.3.1 JavaScript Logic 31
      3.3.2 Web-page Layout 35

4 Conclusions and Future Work 41
   4.1 Conclusions 41
   4.2 Limitations and Future Work 42
Bibliography  

Appendix A: Mongo Export to Flat File Script
List of Figures

1.1 Schematic of the component selection design problem [1] . . . . . . . . 3
1.2 Nassar’s software pipeline for the Home Theater Design Problem [22]. 4

2.1 Example XML syntax to encode a group of books [27]. . . . . . . . . 8
2.2 JSON representation vs. XML representation [17]. . . . . . . . . . . . 9

3.1 Front- and Back-end software subsystems for Home Theater Design
   Problem implementation. . . . . . . . . . . . . . . . . . . . . . . 16
3.2 Partial view of the user-interface. . . . . . . . . . . . . . . . . . . . . 36
3.3 Partial view of the user-interface table of designs. . . . . . . . . . . 37
3.4 Cost vs. Performance visualization . . . . . . . . . . . . . . . . . . . 38
3.5 Cost vs. Reliability visualization . . . . . . . . . . . . . . . . . . . . 39
3.6 Reliability vs. Performance visualization . . . . . . . . . . . . . . . 40
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.

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].](image)

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 XML’s 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
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.
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 are not 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)
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. D3’s 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, whereas 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.](image-url)
3.2 Back-end System 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.
Table 3.1: Generation of television design components for the television library. Legend: P = performance, R = reliability.

<table>
<thead>
<tr>
<th>TV</th>
<th>Cost</th>
<th>P</th>
<th>R</th>
<th>Height</th>
<th>Width</th>
<th>Thickness</th>
<th>Weight</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>LG</td>
<td>$1300</td>
<td>5</td>
<td>0.7</td>
<td>30.8 in</td>
<td>50.6 in</td>
<td>1.2 in</td>
<td>48.7 lbs</td>
<td>AC Power, HDMI, Video, Audio-L, Audio-R, Antenna/Cable, LAN, USB</td>
<td>Audio-L, Audio-R, Headphones</td>
</tr>
<tr>
<td>Samsung</td>
<td>$1650</td>
<td>8</td>
<td>0.8</td>
<td>29.0 in</td>
<td>49.3 in</td>
<td>1.2 in</td>
<td>35.7 lbs</td>
<td>AC Power, HDMI, Video, Audio-L, Audio-R, Antenna/Cable, LAN, USB, Ex-Link</td>
<td>Audio-L, Audio-R, Headphones</td>
</tr>
<tr>
<td>Sony</td>
<td>$1200</td>
<td>10</td>
<td>0.9</td>
<td>30.4 in</td>
<td>50.0 in</td>
<td>1.6 in</td>
<td>44.5 lbs</td>
<td>AC Power, HDMI, Video, Audio-L, Audio-R, Antenna/Cable, LAN, USB</td>
<td>Audio-L, Audio-R, Headphones</td>
</tr>
</tbody>
</table>
Table 3.2: Generation of amplifier design components for the amplifier library.

<table>
<thead>
<tr>
<th>Amplifier</th>
<th>Cost</th>
<th>Performance</th>
<th>Reliability</th>
<th>Power Handling</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bose</td>
<td>$300</td>
<td>10</td>
<td>0.8</td>
<td>100 watts</td>
<td>AC Power, Audio-L, Audio-R</td>
<td>Speaker-L, Speaker-R</td>
</tr>
<tr>
<td>Polk</td>
<td>$350</td>
<td>8</td>
<td>0.9</td>
<td>175 watts</td>
<td>AC Power, Audio-L, Audio-R</td>
<td>Speaker-L, Speaker-R</td>
</tr>
<tr>
<td>Klipsch</td>
<td>$370</td>
<td>5</td>
<td>0.7</td>
<td>70 watts</td>
<td>AC Power, Audio-L, Audio-R</td>
<td>Speaker-L, Speaker-R</td>
</tr>
<tr>
<td>Speaker</td>
<td>Cost</td>
<td>Performance</td>
<td>Reliability</td>
<td>Power Handling</td>
<td>Inputs</td>
<td>Outputs</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>-------------</td>
<td>-------------</td>
<td>----------------</td>
<td>-----------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Polk</td>
<td>$400</td>
<td>8</td>
<td>0.8</td>
<td>10 - 150 watts</td>
<td>Speaker-R, Speaker-L</td>
<td>Sound</td>
</tr>
<tr>
<td>Klipsch</td>
<td>$300</td>
<td>5</td>
<td>0.7</td>
<td>5 - 85 watts</td>
<td>Speaker-R, Speaker-L</td>
<td>Sound</td>
</tr>
<tr>
<td>Bose</td>
<td>$328</td>
<td>10</td>
<td>0.9</td>
<td>50 - 200 watts</td>
<td>Speaker-R, Speaker-L</td>
<td>Sound</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>String</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brand</td>
<td>String</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cost</td>
<td>Integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td>Integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>Integer Fraction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Height</td>
<td>Integer Fraction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Width</td>
<td>Integer Fraction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thickness</td>
<td>Integer Fraction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td>Integer Fraction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inputs</td>
<td>Array:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Type String</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Quantity Integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td>Array:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Type String</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Quantity Integer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.4: Key-Value pairs of the JSON television representation
Table 3.5: Sample television component in JSON format

```json
{
    "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
        }
    ]
}
```
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.
### Table 3.6: Key-Value pairs of the JSON amplifier representation

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>String</td>
<td>Cost</td>
<td>Integer</td>
</tr>
<tr>
<td>Brand</td>
<td>String</td>
<td>Performance</td>
<td>Integer</td>
</tr>
<tr>
<td>Reliability</td>
<td>Integer Fraction</td>
<td>Power Handling</td>
<td>Integer</td>
</tr>
<tr>
<td>Inputs</td>
<td>Array:</td>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Outputs</td>
<td>Array:</td>
<td>Quantity</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**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
Table 3.7: Sample amplifier component in JSON format
the speaker’s inputs.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>String</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brand</td>
<td>String</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cost</td>
<td>Integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td>Integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>Integer</td>
<td>Fraction</td>
<td></td>
</tr>
<tr>
<td>Power Handling Min</td>
<td>Integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Handling Max</td>
<td>Integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inputs</td>
<td>Array:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Quantity</td>
<td>Integer</td>
</tr>
</tbody>
</table>

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

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

Table 3.11: Output 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:

```plaintext
$ 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. Mongo-
Script 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 thea-
ter 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 through 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.
Please select your axes for design comparison and then click "Graph It!"
(Hover over a point on the graph to show the system design)

<table>
<thead>
<tr>
<th>Design Name</th>
<th>Television</th>
<th>Speaker</th>
<th>Amplifier</th>
<th>Cost</th>
<th>Performance</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>design 1</td>
<td>sony</td>
<td>bose</td>
<td>klipsch</td>
<td>1898</td>
<td>8.33</td>
<td>0.63</td>
</tr>
<tr>
<td>design 2</td>
<td>sony</td>
<td>klipsch</td>
<td>klipsch</td>
<td>1870</td>
<td>6.67</td>
<td>0.77</td>
</tr>
<tr>
<td>design 3</td>
<td>sony</td>
<td>polk</td>
<td>klipsch</td>
<td>1970</td>
<td>8.33</td>
<td>0.63</td>
</tr>
<tr>
<td>design 4</td>
<td>sony</td>
<td>bose</td>
<td>polk</td>
<td>1678</td>
<td>9.33</td>
<td>0.50</td>
</tr>
<tr>
<td>design 5</td>
<td>sony</td>
<td>klipsch</td>
<td>polk</td>
<td>1850</td>
<td>7.67</td>
<td>0.63</td>
</tr>
<tr>
<td>design 6</td>
<td>sony</td>
<td>polk</td>
<td>polk</td>
<td>1950</td>
<td>9.33</td>
<td>0.90</td>
</tr>
<tr>
<td>design 7</td>
<td>sony</td>
<td>bose</td>
<td>bose</td>
<td>1828</td>
<td>10.00</td>
<td>0.67</td>
</tr>
<tr>
<td>design 8</td>
<td>klipsch</td>
<td>bose</td>
<td>bose</td>
<td>1800</td>
<td>8.33</td>
<td>0.80</td>
</tr>
</tbody>
</table>

Figure 3.2: Partial view of the user-interface.
<table>
<thead>
<tr>
<th>Design Name</th>
<th>Television</th>
<th>Speaker</th>
<th>Amplifier</th>
<th>Cost</th>
<th>Performance</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>design 1</td>
<td>sony</td>
<td>bose</td>
<td>klipsch</td>
<td>1890</td>
<td>8.33</td>
<td>0.93</td>
</tr>
<tr>
<td>design 2</td>
<td>sony</td>
<td>klipsch</td>
<td>klipsch</td>
<td>1870</td>
<td>6.67</td>
<td>0.77</td>
</tr>
<tr>
<td>design 3</td>
<td>sony</td>
<td>polk</td>
<td>klipsch</td>
<td>1970</td>
<td>8.23</td>
<td>0.83</td>
</tr>
<tr>
<td>design 4</td>
<td>sony</td>
<td>bose</td>
<td>polk</td>
<td>1878</td>
<td>9.33</td>
<td>0.90</td>
</tr>
<tr>
<td>design 5</td>
<td>sony</td>
<td>klipsch</td>
<td>polk</td>
<td>1850</td>
<td>7.67</td>
<td>0.83</td>
</tr>
<tr>
<td>design 6</td>
<td>sony</td>
<td>polk</td>
<td>polk</td>
<td>1950</td>
<td>9.33</td>
<td>0.90</td>
</tr>
<tr>
<td>design 7</td>
<td>sony</td>
<td>bose</td>
<td>bose</td>
<td>1828</td>
<td>10.00</td>
<td>0.87</td>
</tr>
<tr>
<td>design 8</td>
<td>sony</td>
<td>klipsch</td>
<td>bose</td>
<td>1800</td>
<td>8.33</td>
<td>0.80</td>
</tr>
<tr>
<td>design 9</td>
<td>sony</td>
<td>polk</td>
<td>bose</td>
<td>1900</td>
<td>10.00</td>
<td>0.87</td>
</tr>
<tr>
<td>design 10</td>
<td>sony</td>
<td>klipsch</td>
<td>bose</td>
<td>2348</td>
<td>7.67</td>
<td>0.80</td>
</tr>
<tr>
<td>design 11</td>
<td>sony</td>
<td>polk</td>
<td>klipsch</td>
<td>2320</td>
<td>8.00</td>
<td>0.73</td>
</tr>
<tr>
<td>design 12</td>
<td>sony</td>
<td>polk</td>
<td>polk</td>
<td>2420</td>
<td>7.67</td>
<td>0.80</td>
</tr>
<tr>
<td>design 13</td>
<td>sony</td>
<td>bose</td>
<td>polk</td>
<td>2320</td>
<td>8.67</td>
<td>0.97</td>
</tr>
<tr>
<td>design 14</td>
<td>sony</td>
<td>klipsch</td>
<td>polk</td>
<td>2300</td>
<td>7.00</td>
<td>0.80</td>
</tr>
<tr>
<td>design 15</td>
<td>sony</td>
<td>polk</td>
<td>polk</td>
<td>2400</td>
<td>8.67</td>
<td>0.87</td>
</tr>
<tr>
<td>design 16</td>
<td>sony</td>
<td>bose</td>
<td>bose</td>
<td>2270</td>
<td>9.33</td>
<td>0.83</td>
</tr>
<tr>
<td>design 17</td>
<td>sony</td>
<td>klipsch</td>
<td>bose</td>
<td>2250</td>
<td>7.67</td>
<td>0.77</td>
</tr>
<tr>
<td>design 18</td>
<td>sony</td>
<td>polk</td>
<td>bose</td>
<td>2350</td>
<td>3.33</td>
<td>0.83</td>
</tr>
<tr>
<td>design 19</td>
<td>lg</td>
<td>bose</td>
<td>klipsch</td>
<td>1998</td>
<td>6.67</td>
<td>0.77</td>
</tr>
</tbody>
</table>

Figure 3.3: Partial view of the user-interface table of designs.
Figure 3.4: Cost vs. Performance visualization
Figure 3.5: Cost vs. Reliability visualization
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 similar 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 design 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 compatibility 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


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(','); // Append with a comma
  });
}
print(']'); // Close the JSON array