DRACULA

CSM Turner

Connor Taylor, Trevor Worth

June 18th, 2015

Acknowledgments

Support for this work was provided by the National Science Foundation Award No. CMMI-1304383 and CMMI-1234859. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Table of Contents

Acknowledgments 1
Introduction 3
Requirements: 4
Functional Requirements: 4
Non-Functional Requirements: 4
System Architecture 5
Technical Design 6
The Optimization Algorithm 6
Basic Design Structure 10
The Database 11
The User Interface 12
The Admin Interface 12
Design & Implementation/Lessons Learned 13
Results 15
Appendix I 16
Option 1 – Javascript 17
Option 2 - Java Applet 19
Option 3 - Desktop Application 21
Appendix II 23

Introduction

Dr. Cameron Turner is a Mechanical Engineering professor at the Colorado School of Mines. Dr. Turner is part of a group of professors and graduate students working here at Colorado School of Mines and at the Georgia Institute of Technology to allow engineers to more easily use analogies in their designs. For the last few years they have been working on developing the Design Repository and Analogy Computation via Unit-Language Analysis (DRACULA) system, and have enlisted us to aid them in reinventing their previous user interface and data storage frameworks.

The DRACULA project aims to allow users to specify parameters of a design they are working on, and search through a database to find other designs that may contain elements that are useful for their design. In order to do this, the project needs interfaces to both allow the users to easily search through the database, and for the scientists on the project to manage the database.

When we began, the project had an interface in C++ designed to allow the user to look for designs and a working Excel database. However, the original software was difficult to use and manage, which prevented the project from being used efficiently. Our goal was to create graphical user interfaces for these purposes that are easy to use, create a web page to access the interfaces, as well as update the current database to a format that is more suited for use.

Requirements:

The DRACULA system must allow users to simply relate problems in one engineering environment to another using analogies, and must store a list of known analogies and their graphical data points in a database which can be edited and maneuvered by someone with only basic knowledge of the system. Specific requirements are:

Functional Requirements:

1. A database that efficiently stores data corresponding to the analogies
2. An interface for adding, deleting, and editing database entries
3. An interface for searching the database and displaying results to the user
4. Hyperlinks that are shown in the search interface are opened in a browser once clicked
5. The search results were originally displayed in a log files and in the final product must be shown directly to the user.
6. Search results return a “good” answer on the first page
7. The program should use data from the database to compute the analogies
8. Database edits should be accepted by an administrator before being implemented

Non-Functional Requirements:

1. The program should be runnable in a web browser
2. The interfaces should be easy to use
3. The interfaces should be clean and contain no extraneous information
4. The Optimization function is modular, and can be changed or replaced without affecting other code
5. The database must support an ever-growing list of data
6. Our system must be portable amongst most operating systems.

System Architecture

Figure 1 shows the overall structure of our design. We have two main interfaces, the Database Management Interface and the Search Interface. Both of these interfaces are located on one server and the user interacts with the interface through their browser. The Database Management Interface serves to allow administrators to create, modify, and delete entries. We query the database for the information it currently contains, and we then display this to the administrator for modification. Once the administrator decides what they wish to do, whether it is adding, editing, or deleting information, we send a new query to the mySQL database to make this change.

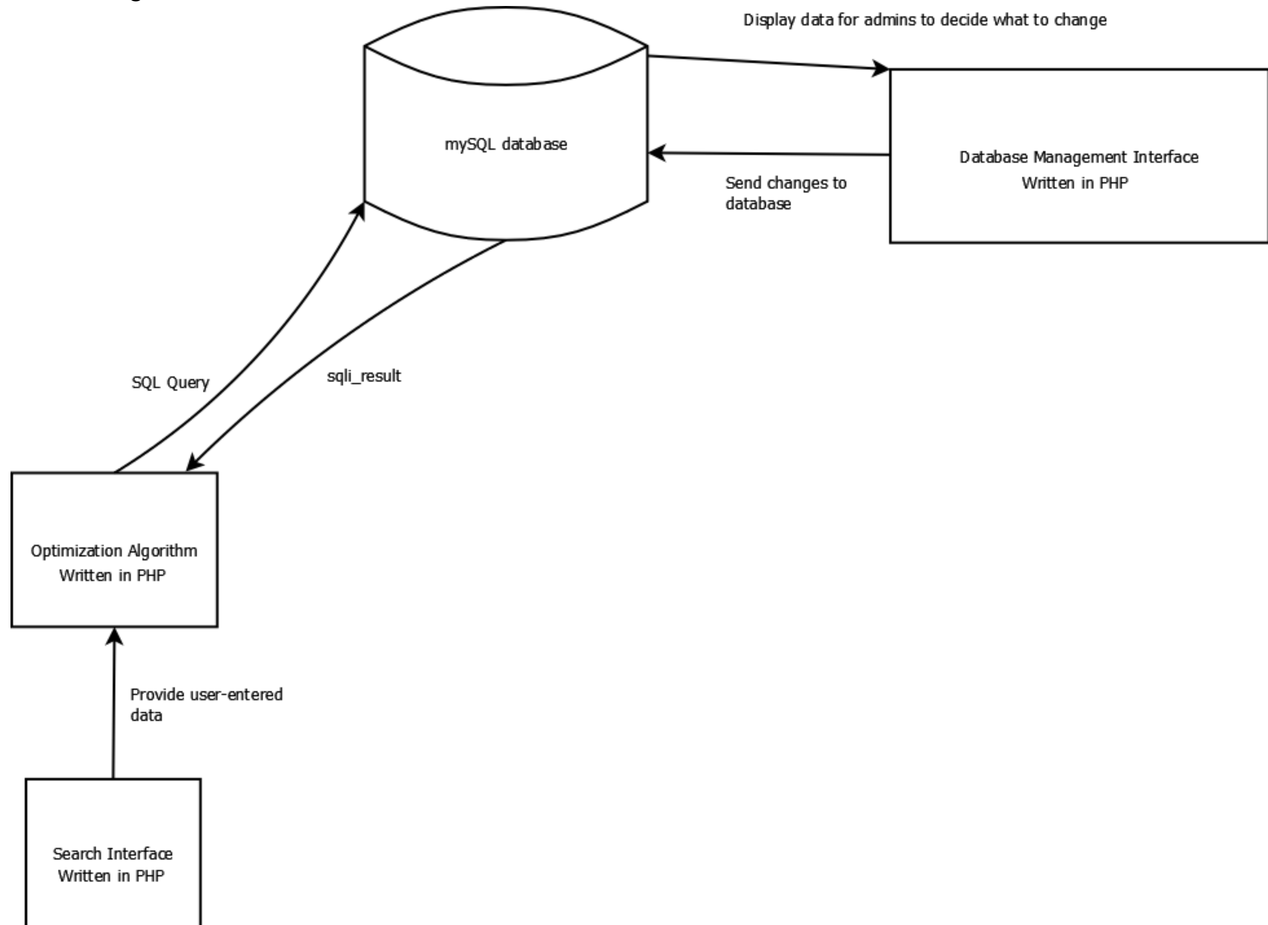

Figure 1: System Architecture

The Search Interface allows a user to provide some basic information on their design and receive analogous designs that may help them with their project. There are two main parts of doing a search. The Search Interface is where the user enters in their data and where the data is displayed once the search is finished. The Optimization Algorithm serves to do the search through our database. Since the DRACULA project does not use a keyword search, the searching through the database is more intensive. The basic structure of how a search works is shown in Figure 2. This search requires 3 main interactions with our database in order to determine the most relevant analogies.

Technical Design

The Optimization Algorithm

The DRACULA system is based on an algorithm developed by Dr. Turner's team designed to relate different structures in both engineering and in nature by their design rather than by keyword. What this means is that the program relates data based on the units (mass, length, time, electric charge etc) of parameter being processed, referred to as "Engineering Parameters" and bond graph representations of the functions which transform these materials, known as "Critical Functions," to score different designs, referred to as "Analogies," which are stored in the database and rank them on their relevance to a user-entered query. Each Analogy has the same general breakdown of data, and because of this we store each Analogy during the search as an object. Analogies are specifically made up of a series of "Critical Pairs," a collection of more specific data stored in an order. Each Critical Pair has a variable number of "Critical Flows," each with its own input Flow, input Engineering Parameter, Critical Function, output Flow, output Engineering Parameter and Relationship. A "Flow" is the material that transfers the engineering parameter between Critical Flows. the "Relationship" field indicates which flows follow that flow other than the next one in the list. A list of valid Engineering Parameters, Critical Functions and Flows is included in figures 9-11 in the appendix, as well as a visual example of an Analogy in figure 2.

EntryNumber		1			
EntryName		Tubules on whale fin			
Description		Bumps on the leading edge of fin allow for a decrease in drag, increase in lift and a greater efficiency in converting biological energy into forward momentum			
ExampleUsage		Bumps placed on the leading edge of wind turbine blades can result in a greater efficiency in the turbine performance			
Domain		Biological			
Field		Pressure			
DesignForX		Efficiency			
CriticalPair1	InFlow1	Liquid	Liquid	Gas	Gas
	InEngParam1	Volumetric flow	Viscosity	Lift	Drag
	InParamUnits1	03-1000000	1-1-1000000	11-2000000	11-2000000
	CritFunc1	Convert	Convert	Convert	Convert
	BGType1	00011	00011	00011	00011
	OutFlow1	Biological E.	Biological E.	Biological E.	Biological E.
	OutEngParam1	Velocity	Velocity	Velocity	Velocity
	OutPramUnits1	11-1000000	11-1000000	11-1000000	11-1000000
	Relationships	1->2	1->2	1->2	1->2
CriticalPair2	InFlow2	Biological E.	Biological E.		
	InEngParam2	Velocity	Velocity		
	InParamUnits2	11-1000000	11-1000000		
	CritFunc2	Transfer	Transfer		
	BGType2	10100	10100		
	OutFlow2	Energy	Energy		
	OutEngParam2	Energy	Efficiency		
	OutPramUnits2	12-2000000	000000000		
	Relationships	2->3	2->3		
CriticalPair3	InFlow3	Energy	Energy	Energy	Energy
	InEngParam3	Energy	Energy	Efficiency	Efficiency
	InParamUnits3	12-2000000	12-2000000	000000000	000000000
	CritFunc3	Regulate	Regulate	Regulate	Regulate
	BGType3	10000	10000	10000	10000
	OutFlow3	Energy	Energy	Energy	Energy
	OutEngParam3	Energy	Efficiency	Energy	Efficiency
	OutPramUnits3	12-2000000	000000000	12-2000000	000000000
	Relationships				

Figure 2: Example Analogy Entry in Excel. Analogy outline in black, Critical Pair in red, Critical Flow in blue and sample Critical Flow unit in green.

To process this data the computer must construct what is referred to as a "Function Chain," which is made up of the units of each engineering parameter in succession, and is stored as a two-dimensional integer array. To construct a function chain the algorithm iterates through each Critical Flow in Critical Pair one. It then recursively goes through each Critical Pair sequentially, comparing each Critical Flow to the previous flow and matching those which have matching input/output flows.

To better explain, each Critical Pair can be thought of as a collection of dominoes, and each Critical Flow as a single domino in that collection. Each collection is organized vertically, so that the Critical Pair one collection is above Critical Pair two, and so on. The algorithm compares the bottom part of each domino in Critical Pair one (the output Flow) with the top of each domino in Critical Pair two (the input flow). If the two dominos match, the algorithm continues on until it has found a match in every collection all the way down, and adds this one to this list of function chains.

Once the algorithm has found every possible sequential permutation, it repeats this process but organizes the collections based on Relationship instead of sequentially. For example, if a Critical Flow in in Critical Pair one has Relationship "3,5" it will be compared first to dominos in Critical Pair three and a Function Chain will be built, and then compared to dominos in Critical Pair 5, building another set of Function Chains. These function chains will not necessarily be as long as the ones built sequentially.

The process of going through the search in detailed below in Figure 3. In step 1, a user initiates a search using an Engineering Parameter and one to three Critical Functions, for example, a user would enter in Volumetric Flow for the Engineering Parameter and Convert and Transfer for the Critical Functions. In Step 2, we retrieve information from our database related to what the user entered. In our example we would translate Volumetric Flow to its units, which are 03-1000000 representing that it is volume($length^3$) per time($seconds^{-1}$). We would translate the Critical Functions to their bond graphs, which would be 00011 for Convert and 10100 for Transfer. The bond graphs represent the different properties of the Critical Function, but also exist for Engineering Parameter. These properties look at if the Function or Parameter is Resistive, Capacitive, Inertial, a Transformer, and/or a Gyrator in that order. The bond graph has a 1 for a property if the property is true, and 0 if it is false. Convert is both a Transformer and a Gyrator while Transfer is Resistive and Inertial.

Once we have this data, we move to step 3. Here we look through all 114 Engineering Parameters in our database and find the Parameters that are compatible with the bond graph of the first Critical Function. Bond Graph compatibility is defined as sharing a value of one in at least one category. In our example, Convert was our first critical function so we would look for a Parameter that is a Transformer, a Gyrator, or both. After this, we move on to step 4 where we see if the Parameters we found in step 3 have the same units as the Parameter that the user entered. In our example we would be look at all the Parameters that are compatible with Convert's bond graph and that have the same units as Volumetric Flow. If we find any Parameters that meet these restrictions we add the units to a vector that will later become the function chain for the user's data.

We then repeat essentially the same process if a second Critical Function exists. We find all the Parameters whose bond graphs are compatible with the second Critical Function(Transfer in our case) and move to step 5. In step 5, instead of looking for Parameters

that match the units of the user-entered Engineering parameter, we look at all the combinations of two Engineering Parameters and sum up their units. We then compare this sum to see if it matches the units of the user-entered Engineering parameter. For example, if we had two parameter with units of 010000000($length^1$) and 02-1000000($length^2$ and $seconds^{-1}$) we would have a match for Volumetric Flow. We then pair the units of these two Engineering Parameters together and add the pair to the results vector. If we had a third critical function, we would move to step 6, where we repeat the process, but instead of looking at combinations of two Engineering Parameters, we look at combinations of three.

Once we have done this process for all the Critical Functions, we move to step 7. Here the results vector is a Function Chain. If we found one Parameter in step 4 and one pair of Parameters in step 5 for our example, the vector would have (03-1000000) for the first element and (010000000, 02-1000000) for the second. This forms the function chain for the data our user entered in. In step 8, we then make Function Chains for all of the Analogies in our database, using the method that was described previously. We score each analogy based on how similar its Function Chain is the to the Function Chain created from the user's data in step 9. Once this is done we show the top scoring entries to the user.

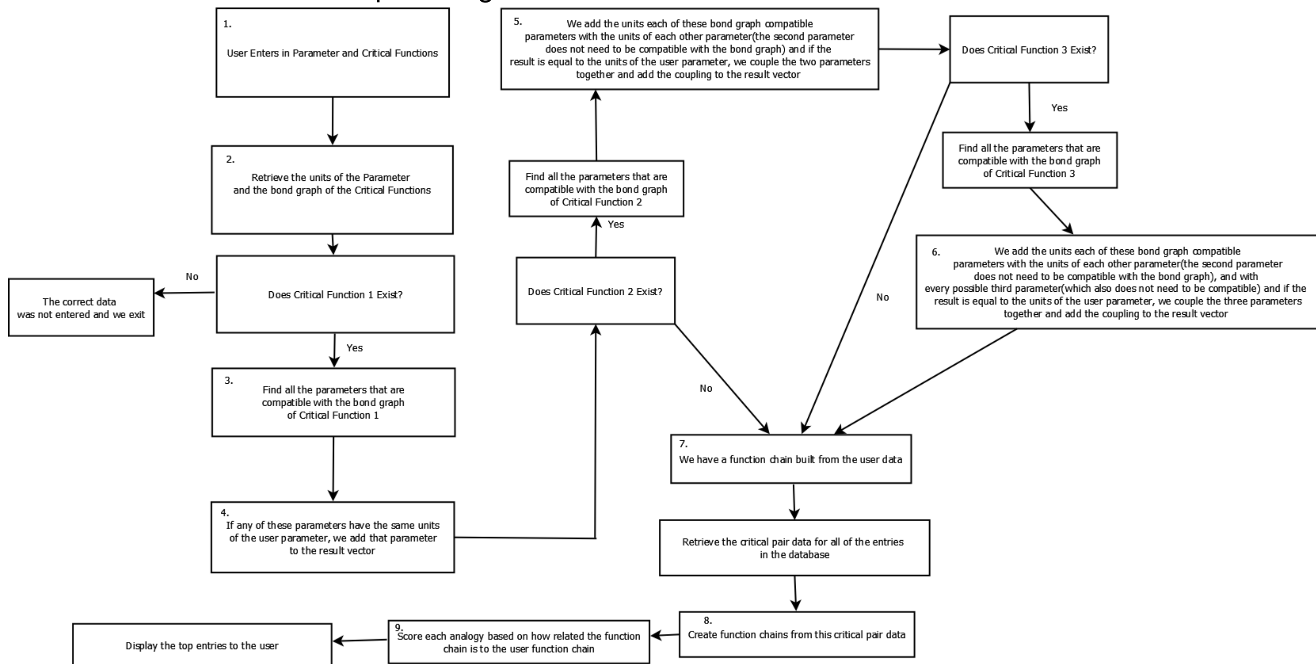

Figure 3: Optimization Algorithm

Basic Design Structure

Figure 4 illustrates the general form of our search algorithm's data flow and necessary interactions with the database. There are three distinct interactions that our program must have with the database. The current PHP version uses more than just these three interactions to improve performance, but still follows the basic structure

The first interaction occurs when the optimization algorithm is started. We are given data in a form that is design to make sense to the user. In order for us to transform this data into something meaningful for the program, we need to query the database. We are given a parameter id and up to three critical function ids from the user interface. We query the database for the units corresponding to the parameter id, the bond graphs corresponding to the critical function ids, and all of the parameter bond graphs. Using this data we construct function chains based off the user-entered data and move to the next database interaction.

The second interaction occurs after the user function chains are built. We query our database for the critical pairs of all of the analogies of the database. This interaction serves to create a score for each analogy based on how relevant it is to the user. The query returns every critical pair with the units for the input and output flows. We then process this data into a series of function chains for each analogy. We can then check how closely the function chains match the user function chains and score the analogy based off the match. Once every analogy has received a score, we move onto the third interaction.

The third interaction serves to get data that would make sense to the user. Once we have a the scores, we look at the entries with the highest score. The user specifies how many entries they wish to see, and the top few entries, we query our database for the corresponding information in the entry info table. This gives the user some details about what the analogy means and how it could be used in their design. This information is then displayed for the user in our user interface.

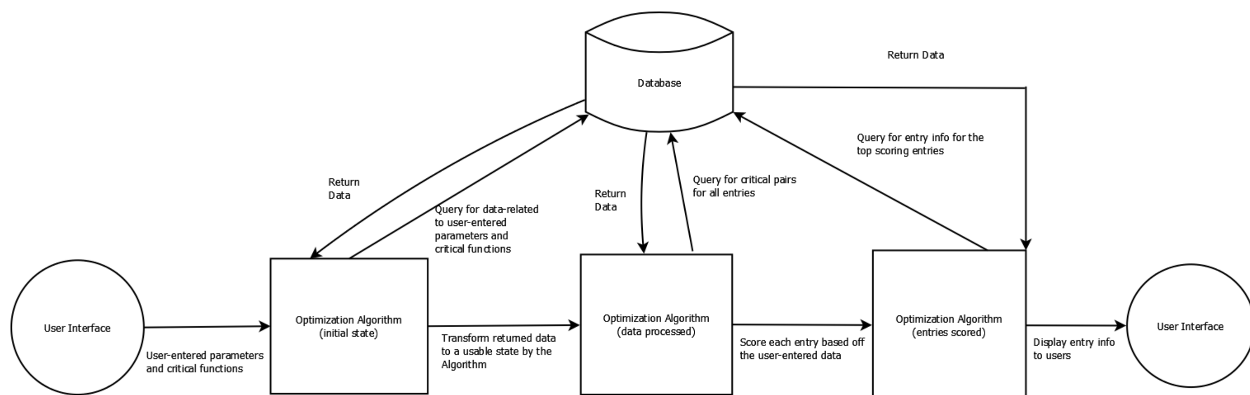

Figure 4: Basic design structure for the Search Interface

The Database

The database stores all of the information relevant to storing Analogies (as "Entries") and their associated data. The general database schema is outlined in Figure 5.

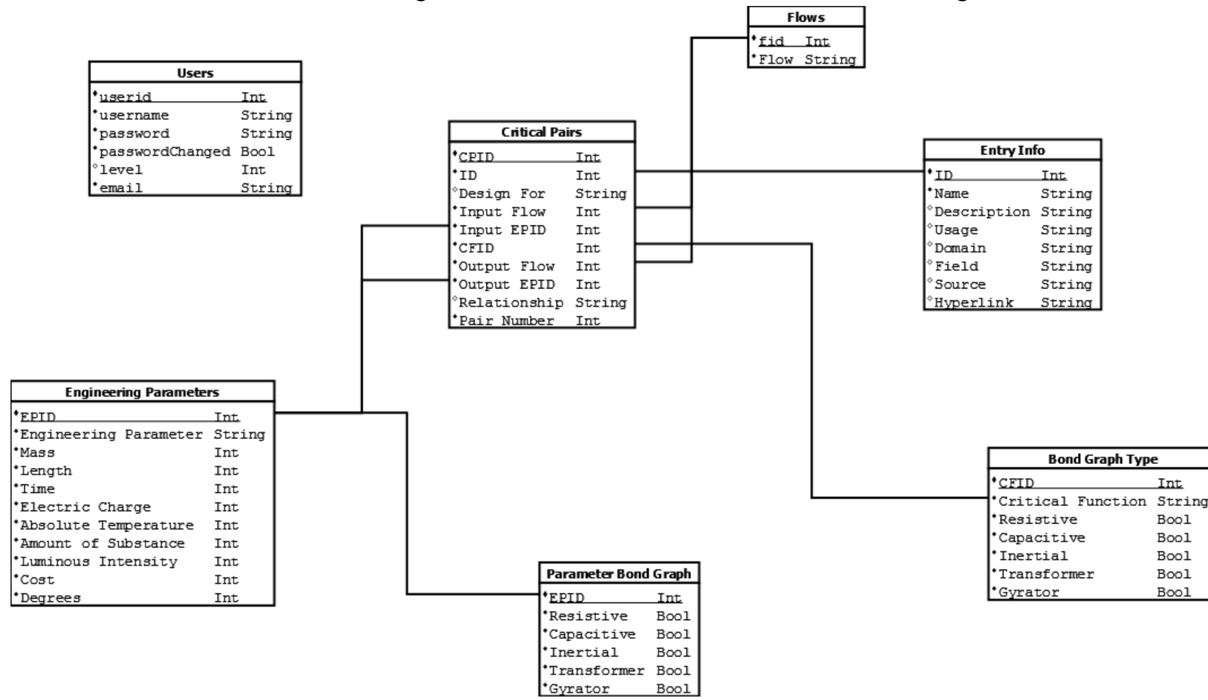

Figure 5: Database Schema

Each entry is stored with its associated ID and information only critical to a user and not the search algorithm, such as the name, description, etc. in the Entry Info table. Associated Critical pair data is stored in the Critical pair table, where it is referenced by its Critical Pair ID (CPID). Each Critical Pair has an ID that relates it back to the Entry Info table, a "Design For" object which is displayed to the user and does not affect the search algorithm, but is associated to each Critical Pair. Critical Flows are stored here implicitly, and are related to each other using the Pair Number section, which indicates which pair (one, two, three etc) the Critical Flow is related to. All other Critical Flow elements are also stored here as various ID's which relate them to entries in the Engineering Parameters, Flows and Bond Graph Type tables.

The Flows table stores a list of valid Flows for each Analogy. Bond Graph Type stores a list of valid Critical Functions and their bond graphs. Engineering Parameters stores the list of valid Engineering Parameters and their units, as well as an ID to associate each Engineering Parameter with its bond graph stored in the parameter Bond Graph table.

The Users table holds information used to authenticate administrators when they log on to edit the database using the PHP interface. This table stores the username, hashed password, email, level and a passwordChanged boolean. The level is not currently relevant information, but was included for potential future changes when users other than just administrators may be allowed to log in. The passwordChanged boolean is checked upon login to force the user to change their password when they log in the first time from the default password, and can be used later by a system administrator to allow admins to change their password later on.

The User Interface

The user interface is built using both PHP and basic HTML. When a user visits the search page, index.php, PHP code is run and generates the web pages relevant to the place the user is in the search. On the initial page, users submit search information through and HTML form by selecting an engineering parameter and one to three critical functions. These options are retrieved from a list stored in the database, and represent the basic structure of an engineering project. For example, a user might enter "energy" as their engineering parameter and then select "regulate" and "transfer" as their critical functions, indicating that they are searching for analogies which relate to the regulation and transfer of energy. These options send the respective ID's for their choices on to the next page.

Afterwards, optimizationAlgorithm.php takes these ID's and matches them to unit data for the engineering parameter and bond graph data for the critical functions and builds the user choice function chain. The algorithm then pulls down all data from the Critical Pairs table and assembles it into Function Chains as detailed above. The function chains are then compared to the user entered chain, and a score is built. An array of Analogies is constructed after pulling down all relevant user information from Entry Info for all Analogies whose score is above a threshold (default is 0), and this array is then sorted by score and displayed to the user using html tables. During the search, the user selects a maximum number of entries they want to see on the next page. If less than that number are found with a score above 0, only that many Analogies will be displayed. If more Analogies are found, but they have the same score, all will be displayed. For example, if a user requests only 10 results to be displayed, but the 9th, 10th and 11th Analogy all have the same score the page will display the top 11. If a user decides they want to see even more entries, they may enter a new number of entries to display and hit the "display more" button. This redirects them to displayMore.php and rather than redoing the search, it sends the previously compiled display array. All entries up to the number requested by the user will be displayed again in the same manner as before.

The Admin Interface

An Administrator accesses the Admin Interface by going directly to the URL and staring at login.html. They submit login information, loginHandler.php checks the information, and they are redirected to editPage.php to begin making changes to the database. Administrators can select using an html form whether they want to create, modify or delete entries, engineering parameters, engineering parameter bond graphs, critical function bond graphs or flows. From each of these options they are directed to createPage.php, modifyPage.php and deletePage.php where relevant html forms are displayed for easy editing. These are handled in createEntry.php, modifyEntry.php and deleteEntry.php, respectively.

Design & Implementation/Lessons Learned

We decided on a multiple-phase design scheme for our project, because of the many features requested by the client. Our client was concerned with the efficiency of our search algorithm, and that once many users started running the search, it would take a long time to do all of the searches. Because of this concern, we looked into running our search algorithm on client-side and having the database on the server side. This led us into doing a multiple phase design.

We started work on the first phase of the design, creating features which are universal to all overarching designs for this project. For this phase we created: a mySQL database that served to improve organization and ease of storage for the data, an HTML/PHP based user interfaces to allow users to search the database and admins to maintain it, and we rewrote the search algorithm from C++ into PHP so that it may be used online rather than having to be installed on a user's computer. These features are necessary for the next phases of the project and prepare the project to move to a client-side search algorithm. By implementing a PHP based search interface, we allow the project to be functional while the client-side interface is being worked on. The focus in this phase was on moving the database to a mySQL form as well as the software to allow for the administration of the database. Completing this tier of the project gives the client all of the basic features that they have requested and supports them for small-to-medium scale use of the program, and sets the basis for further work later on.

The second tier of work involves researching and providing documentation for the project on potentially moving the search algorithm and other resources to the client's computer, with the intention of freeing up resources on the server for long-term growth. We looked into possibilities such as writing the search GUI as Javascript or as a Java Applet during this phase of the project. While this tier does not involve much actual coding or production of working materials, it will represent a starting place for whichever group takes on the third tier, whether that is our group, the project leads or another field session group down the line. Details of the 3 options that we looked at are provided in Appendix I.

The third tier of work will be actually moving the search interface to a client-side language. This is outside the scope of what we will be able to accomplish during this field session.

We decided to use this multi-tier design to maximize the benefit for the client while still allowing us to "complete" the project - that is, to allow us to accomplish the minimum requirements for the project first without getting lost under the weight of potentially more complicated and difficult-to-produce features in the beginning and producing nothing of value. The goal for our project is to provide as much value to the client as possible, and we believe this project structure allows us to do that.

Beyond overall project structure, our first major design decision was which database type to use. Several database services were considered, including Microsoft Access, MongoDB and mySQL. We eventually settled on mySQL because it is open source, more or less standard and has fairly comprehensive documentation. We determined that MongoDB was feasible, but we did not see a clear reason to use MongoDB, so we decided to stay with the more traditional SQL databases. We moved away from Access because the project previously attempted storing the database in Access and failed, so we were unsure whether or not it would be the wisest

choice. Unlike Access, mySQL is generally standard on most servers and does not require a licence to use.

Next, our clients requested that the user interface, currently in C++, be made so that it could be opened uniformly in a web browser. For this reason, we decided to use PHP because it would allow us to not only make a web-based GUI but also efficiently talk to and retrieve data from the mySQL database. PHP also operates server-side, making it platform-independent for our users, so long as they have a web browser to view the page. We opted to use PHP over other server-side languages because it is accessible on most servers, and because of its many similarities to languages we were already familiar with.

Results

We were able to successfully port the database of the project from its current Excel form to a mySQL form. To do this we had to understand and implement a system to structure the data for access by the administrators, users, and the search program. This was important as many aspects of the current database were done manually, and we were able to automate much of this process which makes it much easier to use and maintain. The new form is intuitive to use, directly accessible by our programs, and simplifies the data entry process, as well as eliminates the need to translate the database into a text file so programs can access it. We believe that this new database will make the work of our client easier and meets the requirement that we should create a formal database.

We also have created a web-based interface for managing the database. This allows an administrator to create, edit, and delete the majority of entries in our database. We have tested our software in Google Chrome and Internet Explorer, and has been completely functional so far. This meets our client's requirements that the software is portable to any system, and is openable in a web browser. This software allows our client to easily access and manage the database.

We have translated the current search algorithm from C++ into PHP to allow it to be accessed by a web-based interface. The PHP form allows the client to have a functional search interface for the coming fall semester and sets up the project for future phases beyond the current field session.

We have also implemented a simple login process for the editing of the database. As this will be accessible from the web, we wanted to make sure that only administrators can make edits to the database. The login has only one tier of users in the current form. Our client had asked for additional functionality with the login, such as requiring that multiple administrators approve any modifications and deletions of entries. However, we did not have time to implement this feature

Much of the work has been focused on finding the correct design of the project. We spent most of our time figuring out how the data should flow between interfaces, and what the proper form of the database should be. Once this was done we were able to easily implement the interfaces to interact with the database. Our finished product sets up the project for an implementation of a client-side search interface in future phases.

Appendix I

DRACULA Phase Two

Phase Two exists to outline and explore different client-side adaptations of the DRACULA search algorithm. The goal is to dissect and understand different methods of moving server-side processing to the client's machine to reduce potential wait times due to server load. In this appendix, we explore three options for a client side move: 1) JavaScript (figure 6), 2) Java Applet (figure 7), and 3) a Desktop Application(figure 8). For each option, we will give an overview of what the option entails, a description of how it works, a diagram of network processes and a table of pros and cons discussing the general benefits and issues with each option.

Currently, the DRACULA search algorithm is done server-side, allowing users to connect and search via the web without using their own computer resources. When a user navigates to the web page, PHP scripts are run using information pulled down from the database to fill drop-down boxes, which the user then uses to select the exact queries for their search. The search algorithm then pulls each analogy entry from the database, scores it, ignores any entry with a score below the threshold, and displays results ordered from highest to lowest score to the user. This has the advantage of running the scripts "near" the database, in a way that minimizes the impact of database queries over the network and allows for many versatile queries to be launched in succession, but puts the effort of actually running the search algorithm on the server processor.

These designs follow the basic structure of our Search Interface, which can be found in figure 2. By following this basic structure we minimize the amount of interactions with the database, ensuring we do not spent too much time waiting for information to be sent from the server to the client.

Option 1 – Javascript

Option 1 would implement our Search Interface, including both the user interface and the optimization algorithm, in html/JavaScript. To do this, a JavaScript program would be written and a link would be embedded in the html, to be run by the client when they open the web page. This structure utilizes client resources to construct the interface and run the search algorithm. Our queries to the database would then be done by using an AJAX GET request which would send the necessary data to a PHP program on the server. This would actually query the database. The query would return an sqli_result which would then be translated into a JSON file and then sent back to the search interface. The advantages and disadvantages of this structure are shown in table 1.

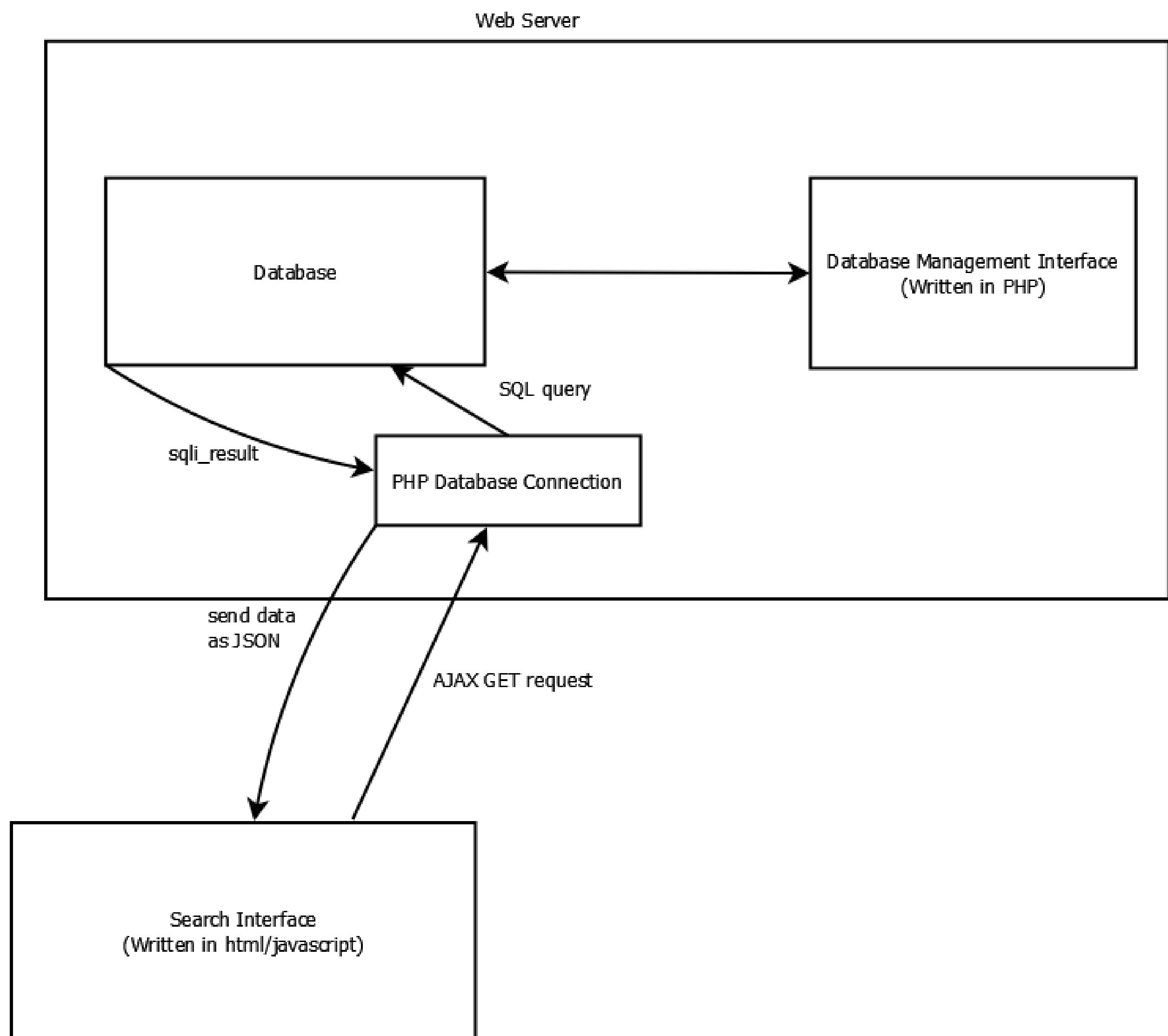

Figure 6: JavaScript Search Interface

Advantages	Disadvantages
Executes the search algorithm on the client's computer	Forces large amounts of data to be transferred over the web
Design is generally portable, most browsers support JavaScript	Increased network requests take up time saved by not running the search algorithm on the server
	As the database grows in size, it becomes less practical to send this amount of data to the client
	Database functionality can not effectively be used to improve the search algorithm

Table 1: Advantages and Disadvantages of the JavaScript Design

Option 2 - Java Applet

Option 2 implements the search interface as a Java Applet. To do this, a Java program is written using the Applet framework and embedded in an html file, which is opened and viewed by the user, but run on their computer. This will require both browser support and for the client to have the correct version of Java installed on their computer. By using a Java Applet we can directly connect to and query our database for the necessary information. In order to set up this connection, the server must have the Oracle Connection Manager installed. The advantages and disadvantages of this structure are shown in table 2.

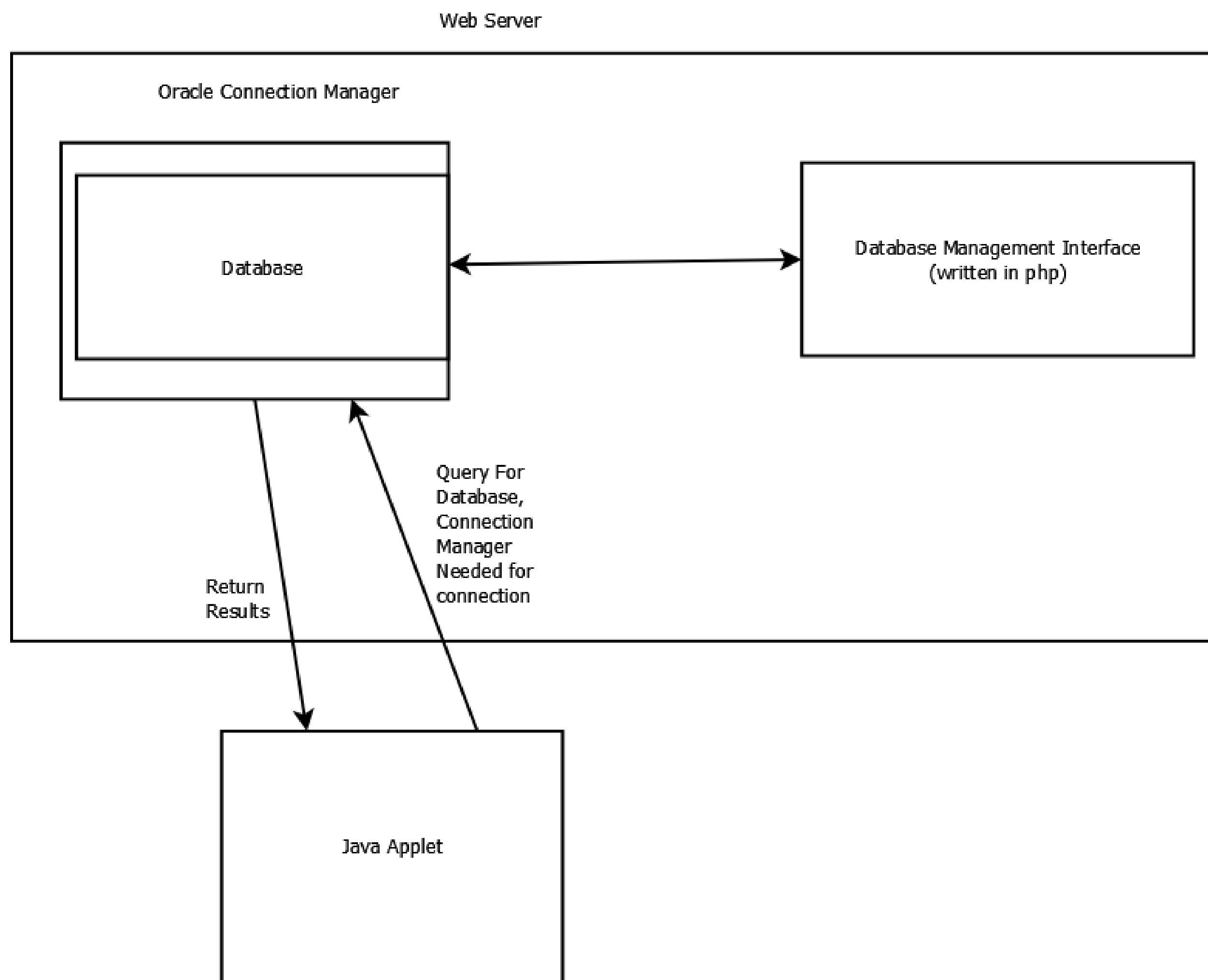

Figure 7: Java Applet Search Interface

Advantages	Disadvantages
Executes the search algorithm using the client's computer resources	Requires the client to have the correct version of Java
Directly connects to the database, rather than using a php connector	Requires the browser that the client is using to support Java Applets
Can be opened in a web browser	Requires the server to have Oracle Connection Manager installed
	Java Applets are losing support in the industry, which could reduce compatibility
	Increased network requests take up time saved by not running the search algorithm on the server
	As the database grows in size, it becomes less practical to send this amount of data to the client

Table 2: Advantages and Disadvantages of the Java Applet Design

Option 3 - Desktop Application

Option 3 creates the search interface as a desktop application. In order to run the interface as a desktop application, a local copy of the database must be present on the client's computer. This would initially be installed along with the search interface and would check for updates when the search is run. The user would need to access a web page to download the installer for the program initially, and updates would need to be done whenever the local database is outdated. There are no specific language requirements for the desktop application, any language that can connect to a mySQL database would work. The advantages and disadvantages of this structure are shown in table 3.

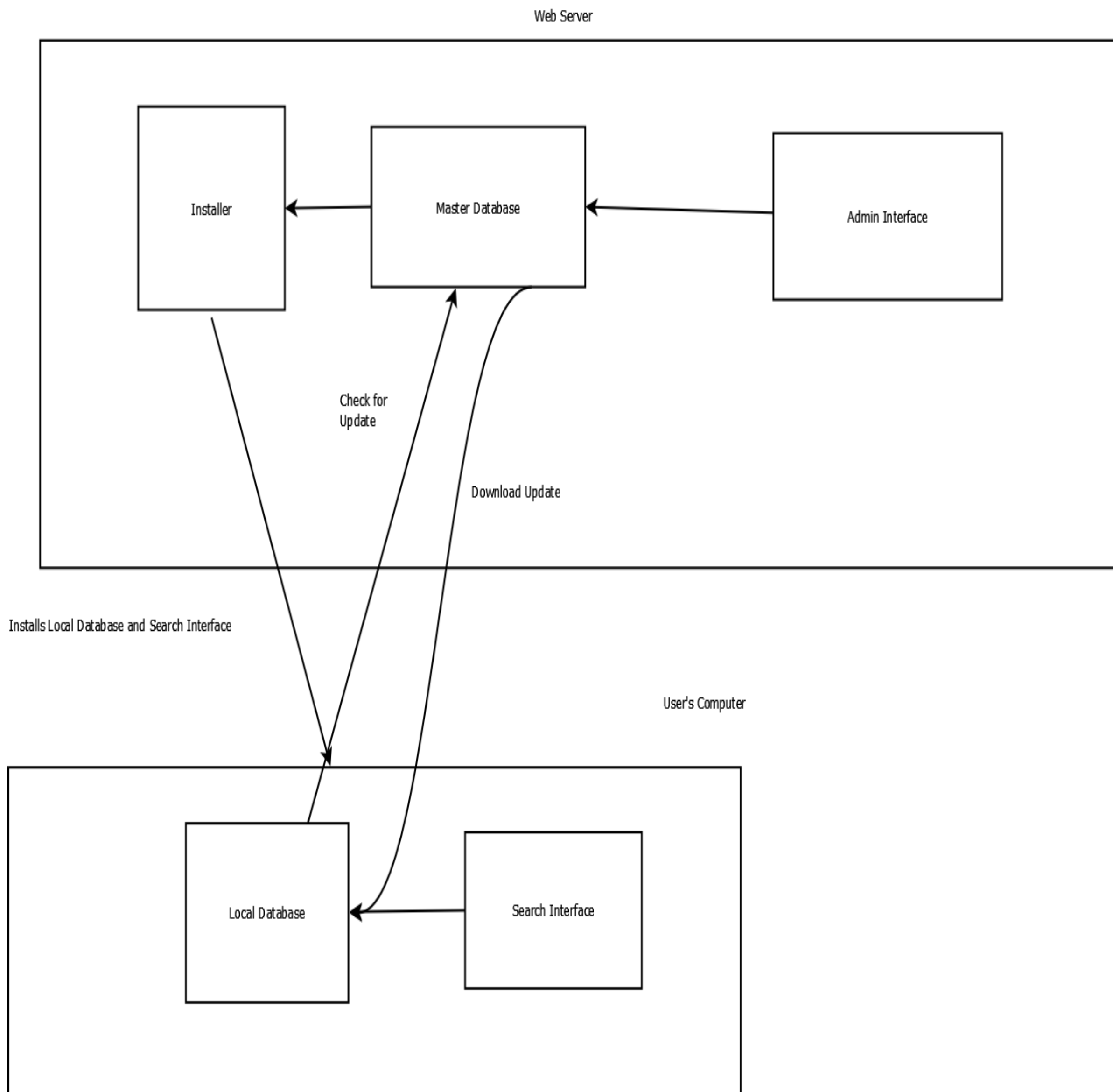

Figure 8: Desktop Application Search Interface

Advantages	Disadvantages
Executes the search algorithm on the client's computer	Is platform dependent - different programs will be needed for different operating systems.
Does not have a specific language requirement	Database format cannot be universally installed on all computers, different versions of the database would be needed for different operating systems For example, on Windows Machines, mySQL runs as a Windows service, a local database such as SQLite would be butter, but then the server would not have the same database management system as the client
Search interface could be made more efficient with SQL queries	Excess network traffic is created pinging the server to check current database versions
	Downloading the initial installer creates a large amount of network traffic.
	A full version of the database must be sent for each change, rather than updating the current one on each computer
	Users have direct access to the database, and could easily modify it
	Future updates to operating systems could cause the program to no longer be functional on new machines
	Industry is moving away from these frameworks and more toward cloud- and server-based options

Table 3: Advantages and Disadvantages of the Desktop Application Design

Appendix II

Magnetomotive Force
Mass
Mass Flow Rate
Molar Energy
Molar Entropy(molar heat capacity)
Molar Flow Rate
Molar Mass
Moment of Inertia
Momentum
Monetary Flow
Number of Particles
Particle Velocity
Permeability
Permittivity
Power
Pressure
Pressure Momentum
Radian
Radiation Potential
Radioactivity
Reaction Rate
Resistivity
Revolutions Per Second
Solid Angle
Specific Energy
Specific Enthalpy
Specific Entropy
Specific Heat Capacity(at constant pressure)

Figure 9: A sample of the 114 Engineering Parameters of the DRACULA Application

Actuate
Branch
Change
Channel
Connect
Control Magnitude
Convert
Couple
Distribute
Export
Guide
Import
Indicate
Mix
Position
Process
Provision
Regulate
Secure
Sense
Separate
Signal
Stabilize
Stop
Store
Supply
Support
Transfer

Figure 10: Critical Functions of the DRACULA Application

Acoustic
Biological Energy
Chemical Energy
Electrical Energy
Electromagnetic Energy
Energy
Gas
Human Energy
Human Material
Hydraulic Energy
Liquid
Magnetic Energy
Material
Mechanical Energy
Mixture
Plasma
Pneumatic Energy
Radioactive/Nuclear Energy
Signal
Solid
Thermal Energy

Figure 11: Flows of the DRACULA Application