

12/11/2014

CIS 4911 Senior Project

Intelligence Inference Engine  
Design Document

Group Members

Jose Acosta  
Lazaro Herrera

Mentor

Eric Kobrin

Instructor

Masoud Sadjadi

# Intelligence Inference Engine : `iie-dev.cs.fiu.edu`

Copyright © [2014] Florida International University. All Rights Reserved. This work is distributed under the W3C® Software License [1] in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

[1] <http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231>

## Abstract

Most web searches currently depend on matching keyword or phrases in order to return results. Matching keyword and phrases works as long as you know what you are searching for and you only want information to that specific thing. In the modern world, when it comes to cyber security new exploits and attacks are constantly coming out. It is not feasible to keep track and know everything about these attacks because there are so many. Fortunately, most new attacks and exploits are usually related to old attacks and exploits. This is where the Semantic Web comes in. Data in Semantic Web is represented by relations, this makes it easy to search for things even without knowing exactly what you are looking for. The only problem with the Semantic Web is that there is a barrier of entry, knowing a specific querying language. This project aims to lower that barrier of entry.

# Table of Contents

[Abstract](#)

[Table of Contents](#)

[Introduction](#)

[Problem definition](#)

[Design methodology used](#)

[Definitions, acronyms, and abbreviations.](#)

[Overview of document](#)

[System Design \(i.e., overall system design\)](#)

[Overview](#)

[Subsystem Decomposition](#)

[User Interface Subsystem](#)

[Query Generation Subsystem](#)

[Hardware and Software Mapping](#)

[Persistent Data Management](#)

[Security/Privacy](#)

[Detailed Design](#)

[Overview](#)

[Static model](#)

[Dynamic model](#)

[Code Specification](#)

[Glossary](#)

[Appendix](#)

[Appendix A - Use case diagram for use cases being implemented.](#)

[Appendix B - Use cases being implemented](#)

[Appendix C – Documented class interfaces](#)

[Appendix D - Diary of meeting and tasks.](#)

[Date: 9/3/2014](#)

[Date: 9/6/2014](#)

[Date: 9/21/2014](#)

[Date: 10/4/2014](#)

[Date: 10/19/2014](#)

[Date: 10-28-2014](#)

[Date: 10-31-2014](#)

[Date: 11/1/2014](#)

[Date: 11/17/2014](#)

[Date: 12/1/2014](#)

[References](#)

# Introduction

In this section of the document the problem that is being addressed by the design in this document is introduced. Apart from the problem being introduced, the design methodology that was used to address the problem is mentioned along with the different models that were used to aid designing the solution. After that some terms were defined, as to better define the problem and an overview of the rest of the document is given.

## Problem definition

The Semantic Web is about linked data. Linked Data is resource-based linking of information. The Semantic web is built of by large of linked data which is defined by the Resource Description Framework(RDF). Each RDF data point consist of three parts: a subject, a predicate, and an object. In essence, RDF give you little building blocks of data that can be connected to other building blocks of data in both directions. When you build complicated webs of connected information, you end up with really specific detailed structures of conceptual knowledge over which you can answer complex question programmatically.

Right now, if you went to Google and put a search query like “Everything related to exploits on SSL that affects Linux and similar systems ”, the results would mostly be articles that cover SSL, some that may cover Linux, and maybe some that cover exploits on SSL. That search result would be useless because it does not give you what you asked for, instead it just matched the keywords you put in, Google does not know what systems are similar to Linux by keywords alone, it does not know what how to put all that together to find specifically what your are looking for . This means that you have to know what you are looking for before you look for it. With Semantic Web you can use relations between different data points and tries to answer the query using relations.

## Design methodology used

For this system, we utilized the Agile development method with regular two week sprint cycles. We created our design from our client’s specific requirements for ease of use and specific storage technologies that the client specs demanded. We utilized both static (Deployment Diagrams, Class Diagrams, Use Case Diagram) and dynamic (Sequence Diagrams) models to represent our system design.

## Definitions, acronyms, and abbreviations.

OSINT - Open-source intelligence

Cyber-attack - Any type of offensive maneuver employed by individuals or whole organizations that targets computer information systems, infrastructures, computer networks, and/or personal computer devices by various means of malicious acts usually originating from an anonymous source that either steals, alters, or destroys a specified target by hacking into a susceptible system [1]

Triple store - A triple-store is a purpose-built database for the storage and retrieval of triples through semantic queries. A triple is a data entity composed of subject-predicate-object. [2]

RDF - a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata data model. It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax notations and data serialization formats. [3]

Semantic Web - collaborative movement led by international standards body the World Wide Web Consortium (W3C). The standard promotes common data formats on the World Wide Web. By encouraging the inclusion of semantic content in web pages, the Semantic Web aims at converting the current web, dominated by unstructured and semi-structured documents into a "web of data". The Semantic Web stack builds on the W3C's Resource Description Framework (RDF). [4]

SPARQL - an RDF query language, that is, a semantic query language for databases, able to retrieve and manipulate data stored in Resource Description Framework format. [5]

## **Overview of document**

This document covers the design process and software processes utilized while designing the Intelligence Inference Engine. The document begins by going over the overall system design, performing a decomposition of each individual subsystem along with the related use cases, maps the specific subsystem to hardware or software where applicable, deals with security and persistent data storage and dives into the detailed design behind our system.

## System Design (i.e., overall system design)

In this chapter, there will be a discussion of the architecture and design patterns utilized while developing this system along with a decomposition of the user interface and query generation subsystems, followed by the software mapping and data/security analysis.

### Overview

The system has a client-server architecture and uses the repository design pattern. The client side consists of the web pages that include the forms and the JavaScript library which generate the queries. The server side consists of several PHP scripts which deal with some minor validation and connect to the two repositories in the system (the SQL database and the RDF triple store). The client-server architecture was used because it was the best way to have the system be scalable. The repository design pattern was crucial for this system. It was crucial because the system depended on being able to store data that could be queried at a later time. The core of this solution is to store and search both queries and data using queries and without the repository design pattern that would not be feasible.

### Subsystem Decomposition

#### User Interface Subsystem

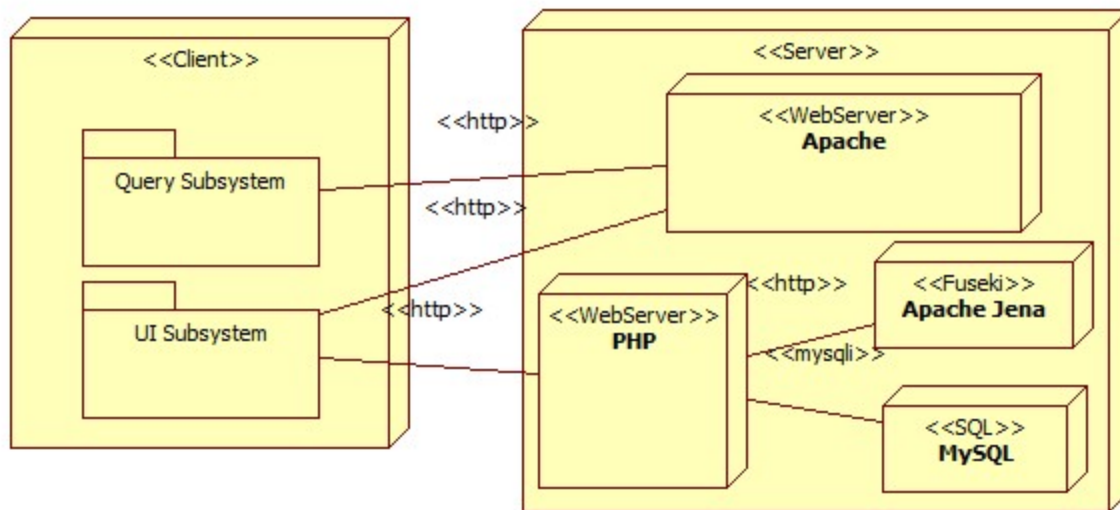
- Simple Web Form to Search For Data
- Simple Web Form to Submit Data
- Historical Autocomplete
- Evidence Evaluation
- Storage and Retrieval of Custom Queries
- Execution of Custom Queries
- Generate A Diagram Based on RDF Data

#### Query Generation Subsystem

- Generate SELECT Queries
- Generate UPDATE Queries

### Hardware and Software Mapping

Below is a Hardware/Software map of how our software is executed. Apache serves up the QueryJS and UI Subsystems and they interact from the client machine to PHP which can then in turn interact with our database.



## Persistent Data Management

For our Jena/Fuseki database, the data is always in the form of Subject - Predicate - Object pairs. Due to the nature of triple-stores this is a given and does not require a Persistent Data Management strategy

For our standard MySQL database, we will be using this storage for custom query storage and retrieval. As such, the following data dictionary is in effect.

Custom Query Data Dictionary			
Name	Length	Type	Rules
query	1024	TEXT	SPARQL code
tags	255	TEXT	CSV data
submission_time	1	TIMESTAMP	default: now()

## Security/Privacy

Our main security concern for this system was blocking port 3030. In the end, we could not due to this requiring Windows Firewall to be enabled in order for a firewall rule to be set up. If we enable the Windows Firewall, Windows Desktop Connection would be blocked. This issue would not exist if we were in front of the hypervisor or had VNC capabilities.



## Detailed Design

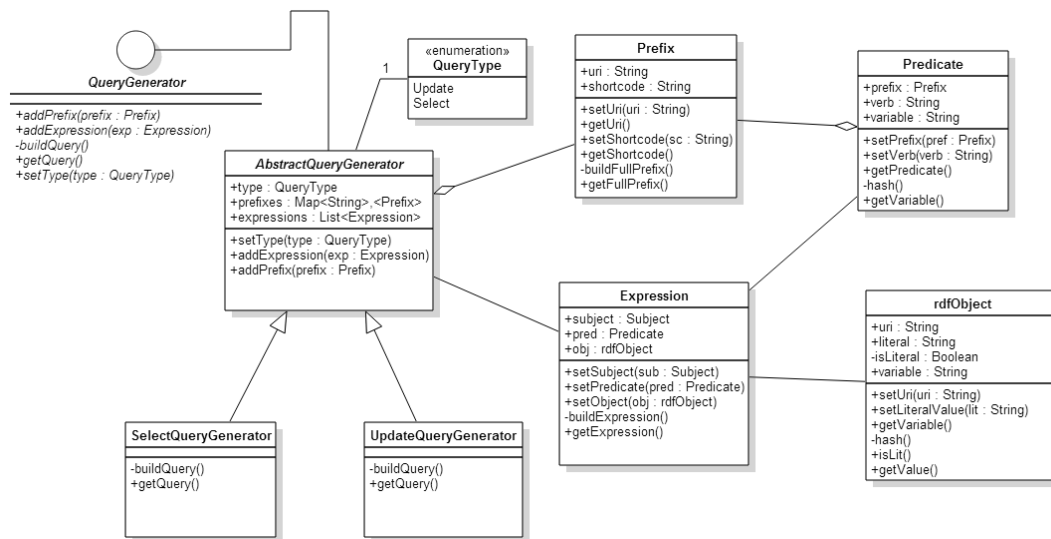
In this section of the document the subsystem that were introduced in the previous section will be further examined. There will be an overview that broadly covers what both of the subsystems do and how they are structured. Next, a detailed description of both subsystems will be given in terms of their static models and dynamic models. Finally, the core pieces of code that make up the two subsystems will be explained in terms of that they do, and what conditions allows them to do function properly.

### Overview

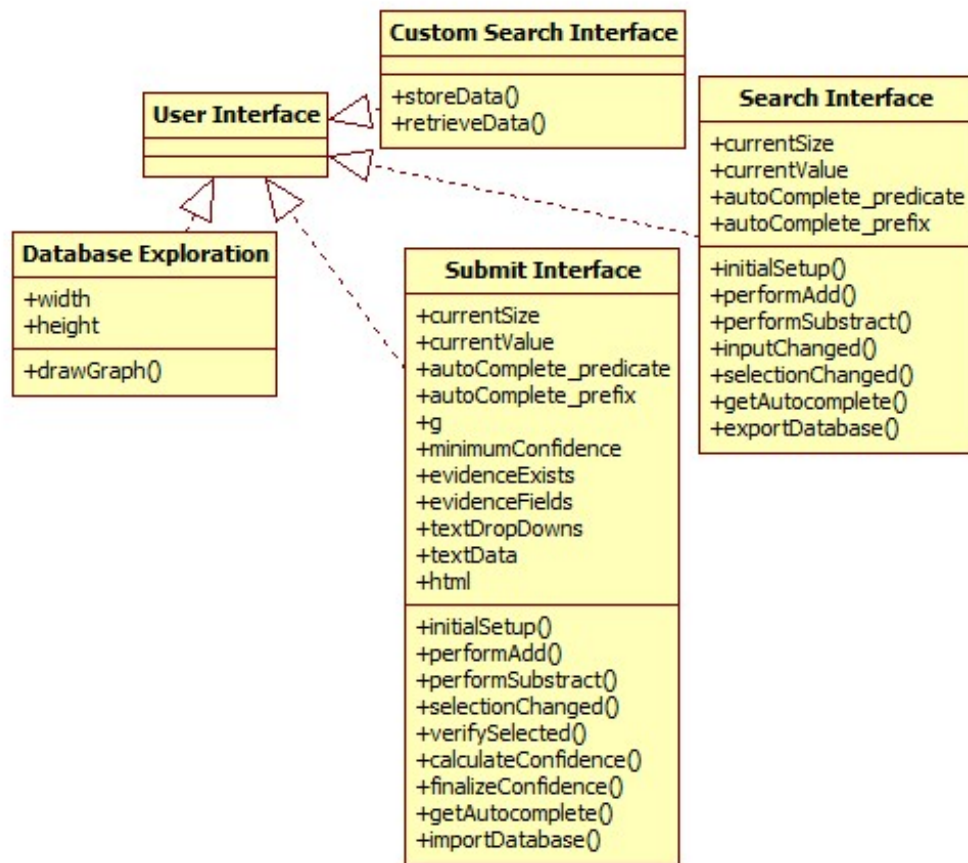
The QueryJS subsystem behaves as a lazy initialized factory class. Storage of the prefix and expressions occurs ahead of time but the query is only constructed on demand. This both saves resources from recalculating the query each update and allows for no “stale” references to older queries.

The UI subsystem is designed to create lightweight but powerful UIs on jQuery. Most of the code is shared amongst classes with only a few lines being devoted to the unique features of each UI.

### Static model

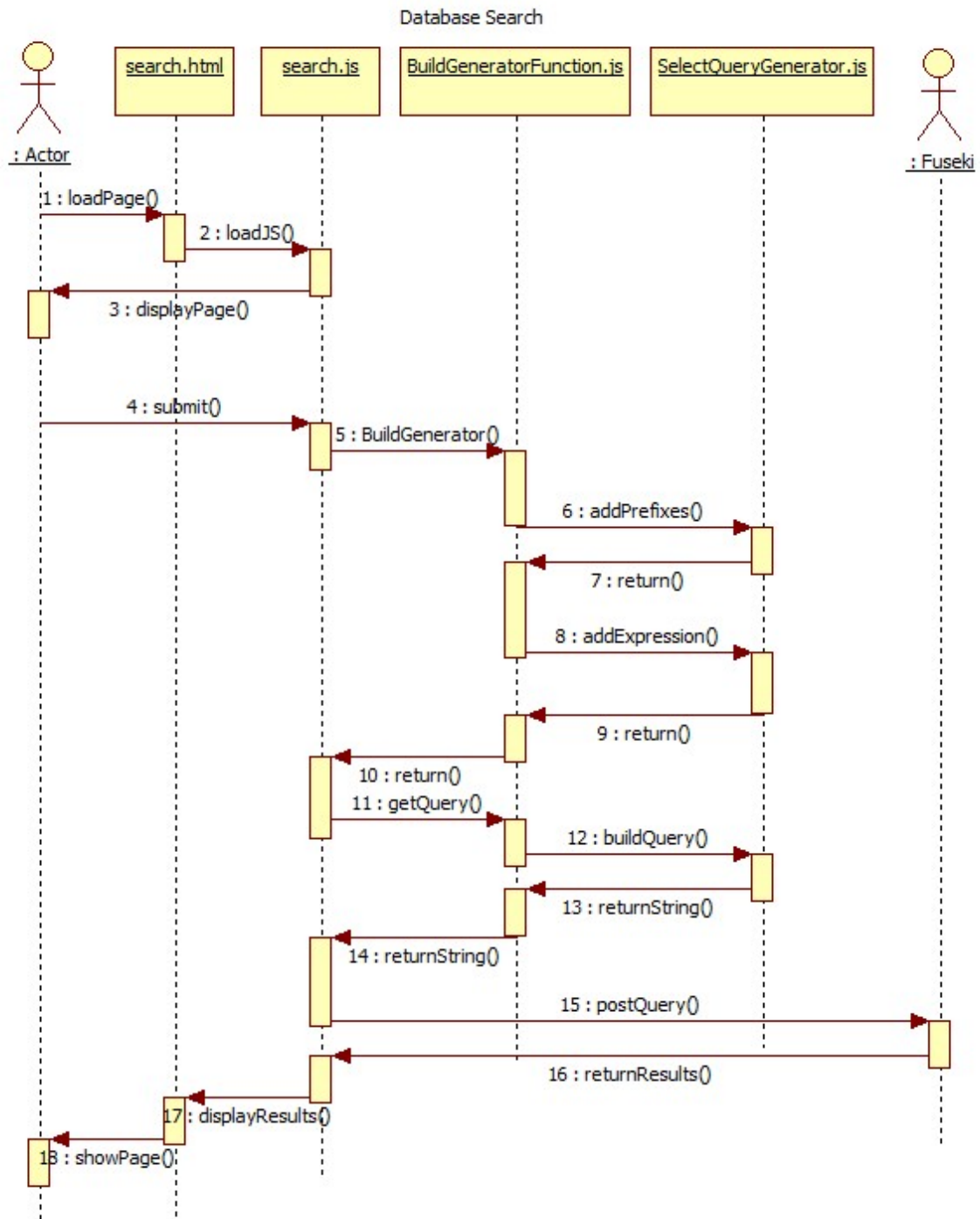


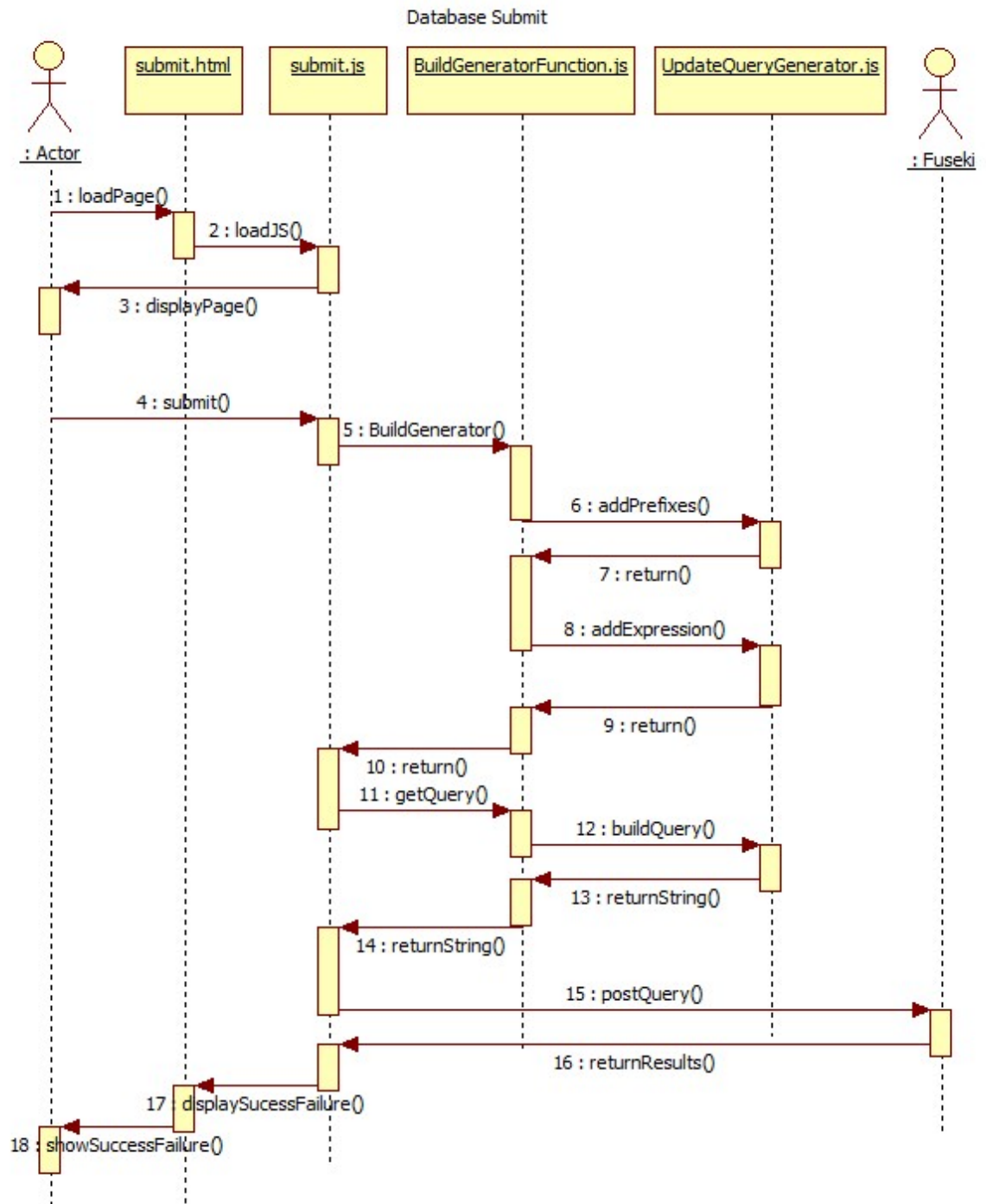
For the Query Generation Subsystem, the following design patterns apply (Abstract Factory, Builder and Lazy Initialization) as we delay the construction of the final query until the last possible moment while the factory and builder can be abstracted and controlled by a parameter sent to a method.



For the UI Subsystem, the following design patterns apply (Facade) as we wrap the forward facing capabilities of both QueryJS and Fuseki in a easier to use context. On top of this, we provide full facade capabilities to the custom query storage and retrieval through MySQL.

## Dynamic model





Here we can see the high-level interaction between the UI Subsystem and the Query Subsystem and how the calls can go from the scope of one subsystem to another before returning. The main algorithms are the building of queries and the dynamic gathering of UI data and dynamic displaying of the data based on how many returns are given from the queries that

are being executed. The generation of queries is left until the last minute and the loading of data from the dynamic context is done upon the submission click.

## Code Specification

In the Query Generation subsystem there are three main classes that control how the library works. The first two are both derived from the same abstract class so I will describe them first, as they share the same interface.

The classes `SelectQueryGenerator` and `UpdateQueryGenerator` both have methods that are vital parts of the library. The first of those methods is the `AddPrefix` function. This function takes in a valid `Prefix` object and adds it to the specific query generator. In order for this function to work properly the `QueryJS` library must be loaded, or else it will not know what a `Prefix` is. This method adds the prefix given to it to an internal list of prefixes that will later be used for query generation. Another method that is core to the generator objects is the `AddExpression` method. This method takes in an `Expression` Object. As with the prefix method `QueryJS` must be loaded so that it understands what an `Expression` object is. This method adds the expression given to it to an internal list of expressions that will later be used to generate a query. Another method that is in both the generator classes is the `setType` function. This function takes in a `QueryType` value and will assign it to the generator as its type if it is valid. The most important method of both generator is the `BuildQuery` function. This function will generate a valid SPARQL query if the prefixes is not null and the expressions is not null or empty. This method is the core of class because it combines the functionality of all the other classes in the library.

Another method that is crucial to the Query Generation subsystem is the `BuildGenerator` function. While this class is not inside the `QueryJS` library itself, it is the method which converts what the user interface subsystem gives and converts it into parts that are useable by the `QueryJS` library. This method has very strict guidelines on what the proper input is, but if the proper input is given and the `QueryJS` library has been loaded then it functions as a very important method that interfaces the two subsystems.

The UI Subsystem has it's main control interface under the submit and search web pages. This method is the core as it packages all of the UI material and passes it to the `QueryJS` Library, issues the query that is returned from `QueryJS` and displays either results or success/failure parameters.

## Glossary

OSINT - Open-source intelligence

Cyber-attack - Any type of offensive maneuver employed by individuals or whole organizations that targets computer information systems, infrastructures, computer networks, and/or personal computer devices by various means of malicious acts usually originating from an anonymous source that either steals, alters, or destroys a specified target by hacking into a susceptible system [1]

Triple store - A triplestore is a purpose-built database for the storage and retrieval of triples through semantic queries. A triple is a data entity composed of subject-predicate-object. [2]

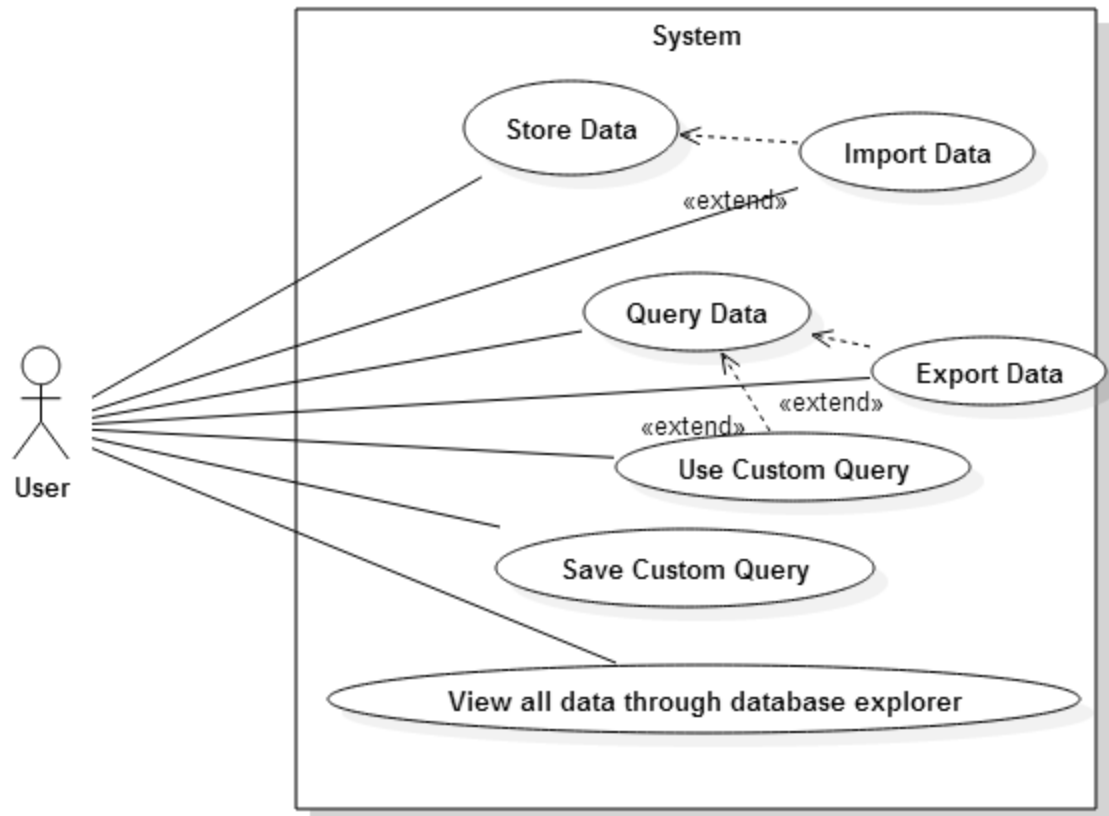
RDF - a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata data model. It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax notations and data serialization formats. [3]

Semantic Web - collaborative movement led by international standards body the World Wide Web Consortium (W3C). The standard promotes common data formats on the World Wide Web. By encouraging the inclusion of semantic content in web pages, the Semantic Web aims at converting the current web, dominated by unstructured and semi-structured documents into a "web of data". The Semantic Web stack builds on the W3C's Resource Description Framework (RDF). [4]

SPARQL - an RDF query language, that is, a semantic query language for databases, able to retrieve and manipulate data stored in Resource Description Framework format. [5]

## Appendix

### Appendix A - Use case diagram for use cases being implemented.



### Appendix B - Use cases being implemented

Name:

Store Data

Participants:

User

Events:

1. From the homepage of the website, the user must click on the "Contribute Now" button.
2. The user will be taken to a form which must be filled out.
3. On the form the user may enter his/her name or leave it blank.
4. The user must fill out the prefixes textbox with the prefixes they will be using for their properties.
5. The user must enter the subject of the data he is about to enter in the "Subject of Lead" textbox

6. The user must be enter at least one property of the subject by selecting a property from the properties drop down selecting the appropriate prefix in the prefix drop down and entering the value of the property in the textbox next to it.
7. If the user wants to submit more than one property they may click on the “+” button to add more property segments . If they want to remove any extra property input segment they can click on the “-” button.
8. The user clicks on the “Submit Lead” button.
9. The user will be taken to the Data Entry form and a message will appear on the top of the page notifying the user if their data was accepted or not.

Alternative Events:

- At step 8, if the user clicks the button and any of the data is invalid then an exception will occur that notifies the user there is invalid data.
- At step 1, or step 8, if the user clicks on the button and the system is down an exception will occur which will tell the user the system is currently down.

Entry Conditions:

User has data that they want to submit

User must be at homepage of the system

Exit Condition:

Data has been stored in the triple store and the user receives acknowledgment of it.

Exceptions:

Data that was input was invalid

The system is down.

Name:

Query Data

Participants:

User

Events:

1. From the home page, the user must click on the “Search Now!” button which will take them to the submit query form.
2. In the submit query form the user must enter the prefixes they will be using in the prefix textbox.
3. The user must then fill out one input segment by selecting at least one property from the drop down box along with an appropriate prefix from the prefix drop down and must enter a value in the text box.
4. If the user wants to submit more than one property they may click on the “+” button to add more property segments . If they want to remove any extra property input segment they can click on the “-” button.
5. The user must then click on the “Search Database” button.



6. The user will be taken to a page that has results of the query they submitted, if any exist.

Alternative Events:

- Instead of having the system generate a query for them, a user may submit their own query by selecting the “Custom Search” button, which will take them to the custom search form.
- At step 5, if the user clicks the button and any of the data is invalid then an exception will occur that notifies the user there is invalid data.
- At step 6, if no results are generated from the query that the user has used then an exception will occur which tells the user there are no results.
- At step 1, or step 5, if the system is down and the user goes through one of these steps then an exception will occur which will notify the user that the system is currently down

Entry Conditions:

User must be at homepage of the system

Exit Condition:

User has successfully entered a query and has gotten a list of results.

Exceptions:

User enters invalid data.

There are no results for the query specified by the user.

The system is down.

Name:

Use a custom query

Participants:

User

Events:

1. From the data retrieval page, the user clicks on the “Custom Search” button. This will take the user to the custom search page.
2. At the custom query page, the user will enter a query that they have generated themselves into the textbox provided.
3. The user will click on the “Execute Query” button
4. If the query is valid and produces results then the user will be taken to a results page.

Alternative Events:

- At step 2, If the user wants to use a query that another user has saved instead of one they generated themselves, then the user will enter the appropriate tags for that query in the tags textbox then click on “Find Queries”.
- At step 3, if the user has entered an invalid query then an exception will occur notifying the user that their query is malformed.
- At step 1, or 3, if the user follows these steps and the system is down an exception will occur notifying the user that the system is currently down.

- At step 4, if the user's query provides no results then an exception will occur and the user will be notified that their query produced no results.

Entry Conditions:

The user must be at the data retrieval page

The user must have a query or the user must know the tags of a query that has been saved

Exit Condition:

User has entered a valid query and has gotten a list of results the query generated

Exceptions:

User entered an invalid query.

The query has returned no results.

The system is down.

Name:

Saving a custom query

Participants:

User

Events:

1. From the data retrieval page, the user clicks on the "Custom Search" button. This will take the user to the custom search page.
2. At the custom search page the user will enter the query that they want to store and enter the tags that will be used to identify that query in the tags textbox.
3. The user will click on the "Add Query" button.
4. The user will receive a message telling them whether the query they submitted was saved.

Alternative Events:

- At step 1, or step 3, if the system is down then an exception will occur and the user will be notified that the system is currently down.

Entry Conditions:

The user must be at the data retrieval page

The user must have a query and the tags he wants to save that query under

Exit Condition:

The query has been saved into the system.

Exceptions:

The system is down.

Name:

Import Data

Participants:

User

Events:

1. From the data entry page, the user will click on the "Bulk Import Data" button.

2. When the button is clicked, a new window will open asking the user to select the file they want to import.
3. The user selects the file that they want to import.
4. The file will be processed entry by entry and valid entries will be added to the database.

Alternative Events:

- At step 3, if the user selects a file that is invalid the web page will refuse the file and do nothing.
- At step 4, if there is an invalid entry then the page will tell the user that an entry has failed to be inserted.

Entry Conditions:

The user must be at the data retrieval page.

The user must have a file that they want to import in the correct xml format.

Exit Condition:

The data has been imported into the system.

Exceptions:

The system is down.

File in unsupported format.

Name:

Bulk Data Export

Participants:

User

Events:

1. At the data retrieval page, the user must click on the "Bulk Export Database" button.
2. Once the button has been pressed the user will download a file containing all the data that is currently in the database.

Alternative Events:

- N/A

Entry Conditions:

The user must be at the data retrieval page.

The user must be able to save a file to their local machine.

Exit Condition:

The user has downloaded a file with all the data in the database.

Exceptions:

The system is down.

Name:

Database Visualizer

Participants:

User

Events:

1. From the homepage, the user will click on the “Explore our databse” tab at the top of the page.
2. Once the user has pressed the tab, then the user will be moved to the visualizer page.
3. At the visualizer page, a graph will be constructed using all the data that is currently in the database and the page will display the graph.

Alternative Events:

- At step 3, if there is no data in the database then a graph showing two null points and a line connecting them will be displayed.

Entry Conditions:

The user must be on the homepage of the system.

Exit Condition:

The user will be able to see a graph with all the data in the database.

Exceptions:

The system is down.

## Appendix C - Documented class interfaces

UpdateQueryGenerator.js

-----  
var type;

var prefixes;

var expressions;

var UpdateQueryGenerator = function UpdateQueryGenerator()

{

//Description : sets the type of the generator

//Input : QueryType enum value

//Output : N/A

//Pre-Condition : QueryJS library must be loaded

//Post-Condition : The type of the generator is set to the type supplied

UpdateQueryGenerator.prototype.setType = function(type)

{

//Description : add an expression object to the list of expressions of the generator

//Input : an Expression object

//Output : N/A

//Pre-Condition : QueryJS library must be loaded

```

//Post-Condition : expression is added to the generator
UpdateQueryGenerator.prototype.addExpression = function(exp)
{
//Description : add a Prefix object to the list of prefixes of the generator
//Input : a Prefix object
//Output : N/A
//Pre-Condition : QueryJS library must be loaded
//Post-Condition : prefix is added to the generator
UpdateQueryGenerator.prototype.addPrefix = function(prefix) //make sure it is the same
prefix object as the one the expression is using
{
//Description : creates a valid update SPARQL query
//Input : n/a
//Output : a string
//Pre-Condition : prefixes and expressions are not null or expressions is not empty
//Post-Condition : the string returned is a valid SPARQL update query
UpdateQueryGenerator.prototype.buildQuery = function()
{
//Description : returns a valid SPARQL update query
//Input : N/A
//Output : a string that is a SPARQL query
//Pre-Condition : buildQuery has been implemented
//Post-Condition : a valid SPARQL update query is returned
UpdateQueryGenerator.prototype.getQuery = function()
{

```

SelectQueryGenertor.js

---

```

var type; //the type of the generator, is QueryType enum value
var prefixes; // an array of Prefix objects
var expressions; // an array of Expression objects

```

```

var SelectQueryGenerator = function SelectQueryGenerator()
{
//Description : sets the type of the generator
//Input : QueryType enum value
//Output : N/A
//Pre-Condition : QueryJS library must be loaded
//Post-Condition : The type of the generator is set to the type supplied
SelectQueryGenerator.prototype.setType = function(type)
{
//Description : add an expression object to the list of expressions of the generator
//Input : an Expression object
//Output : N/A
//Pre-Condition : QueryJS library must be loaded
//Post-Condition : expression is added to the generator
SelectQueryGenerator.prototype.addExpression = function(exp)
{
//Description : add a Prefix object to the list of prefixes of the generator
//Input : a Prefix object
//Output : N/A
//Pre-Condition : QueryJS library must be loaded
//Post-Condition : prefix is added to the generator
SelectQueryGenerator.prototype.addPrefix = function(prefix) //make sure it is the same
prefix object as the one the expression is using
{
//Description : creates a valid select SPARQL query
//Input : n/a
//Output : a string
//Pre-Condition : prefixes and expressions are not null or expressions is not empty
//Post-Condition : the string returned is a valid SPARQL select query
SelectQueryGenerator.prototype.buildQuery = function()
{
//Description : returns a valid SPARQL query

```

```

//Input : N/A
//Output : a string that is a SPARQL query
//Pre-Condition : buildQuery has been implemented
//Post-Condition : a valid SPARQL query is returned
SelectQueryGenerator.prototype.getQuery = function()
{}

```

#### BuildGeneratorFunction.js

---

```

//Description : A function that takes in the parts of a SPARQL query and creates a
generator from the parts
//Input : data : an array whose elements are an array of strings and length 4 with the
elements being subject, prefix shortcode, prefix definition, and object in that order
//           pref : an array whose elements are an array of strings and length 2 with
the elements being prefix shortcode, prefix definition
//           type : a string or QueryType enum value that determines the type of generator
being created
//Output : A query generator (either SelectQueryGenerator or UpdateQueryGenerator
depending on the value of type given)
//Pre-Condition : The QueryJS.js library has been loaded
//Post-Condition : The function returns a Query generator
var BuildGenerator = function(data, pref, type)
{}

```

#### submit.html inner js

---

```

//Global variables for autocompletion and the evidence graph.
var currentValue = 2;
var currentSize = 1;
var autoComplete_prefix = null;
var autoComplete_predicate = null;
var g;

```

```
//Global variables for handling confidence intervals.
```

```
var minimumConfidence = 100;
```

```
var evidenceExists = false;
```

```
var evidenceFields = "";
```

```
var textConcat = "";
```

```
var textDropDowns = "";
```

```
var textData = "";
```

```
var html;
```

```
//Class definition for an empty userInterface;
```

```
function userInterface() {}
```

```
/*
```

Description: Initialize the size value on the page and the gauge variable.

Pre-Condition: The currentSize, the input element with id "size" exists and the div element with id "gauge" exists.

Post-Condition: The input element with id "size" has the initial UI size and the "gauge" id element has basic settings.

```
*/
```

```
userInterface.prototype.initialSetup = function()
```

```
{}
```

```
/*
```

Description: Add a modular element to the page for an extra entry.

Pre-Condition: The div element with id "holder" exists.

Post-Condition: The input element with id "size" has the current UI size and the div element with id "holder" has a new element.

```
*/
```

```
userInterface.prototype.performAdd = function()
```

```
{}
```

```
/*
```



Description: Remove the last modular element from the page (assuming there is more than one)

Pre-Condition: The elements with id "xtra-clear, xtra-prefix, x-tra-select,x-tra-text-input, and size" exist.

Post-Condition: The elements with id "xtra-clear, xtra-prefix, x-tra-select,x-tra-text-input" no longer exist.

The input element with "size" contains the current UI element count.

\*/

```
userInterface.prototype.performSubtract = function()
{}
```

/\*

Description: Handler for change events for select fields.

Pre-Condition: Page is loaded.

Post-Condition: UI changes ("drop!"/ "short-code:prefix"/ "prefix-split") have occurred.

Post-Condition: Confidence evaluation has occurred.

\*/

```
userInterface.prototype.selectionChanged = function()
{}
```

/\*

Description: Verifies if evidence is existing. If it is, recalculates the evidence interval.

Pre-Condition: Page is loaded.

Post-Condition: Evidence data is extracted into variables.

Post-Condition: UI changes ("text") has occurred.

\*/

```
userInterface.prototype.verifySelected = function( index )
{}
```

/\*

Description: Extracts page info for confidence intervals.

Pre-Condition: Confidence has been found in the page.

Post-Condition: Data from page extracted.

\*/

userInterface.prototype.calculateConfidence = function()

{}

/\*

Description: Calculates confidence ranking.

Pre-Condition: Data from page extracted into variables.

Post-Condition: Number from 0 to 100 assigned for confidence.

\*/

userInterface.prototype.finalizeConfidence = function()

{}

/\*

Description: Extractor for auto-complete data.

Pre-Condition: Page is loaded.

Post-Condition: autoComplete\_prefix and autoComplete\_predicate have autocomplete items from the RDF database.

\*/

userInterface.prototype.getAutocomplete = function()

{}

/\*

Description: Imports database from XML.

Pre-Condition: Page is loaded and database

Post-Condition: autoComplete\_prefix and autoComplete\_predicate have autocomplete items from the RDF database.

\*/

userInterface.prototype.importDatabase = function()

{}

/\*

Description: Input Sanitizing

Pre-Condition: String is not null.

Post-Condition: Sanitized string for query creation.

\*/

```
userInterface.prototype.sanitizeInput = function(str)
```

```
{}
```

/\*

Description: Submission method.

Pre-Condition: Page loaded.

Post-Condition: Table created with search results.

\*/

```
userInterface.prototype.submit = function()
```

```
{});
```

```
}
```

```
$(document).ready(  
    function()
```

```
{
```

```
    var UI = new userInterface();
```

```
    UI.initialSetup();
```

```
    //Deal with adds.
```

```
    $( "#plus" ).click(UI.performAdd);
```

```
    //Deal with removals.
```

```
    $( "#minus" ).click(UI.performSubtract);
```

```
    $("#submit").click(UI.submit);
```

```
    //Deals with bulk import disabling if it's not supported in other browsers.
```

```
    if (typeof window.FileReader !== 'function') {
```

```
        $("#bulk-import").attr("disabled", "true");
```

```
    }
```

```
    else
```

```

    {
        $("#bulk-import").click(UI.importDatabase);
    }

    //Get all selected values to be printed.
    $("div").on('change', 'select.form-control', UI.selectionChanged);
    $("div").on('change', 'input.form-control', UI.selectionChanged);

    //Attach auto-complete handlers to each input field on click.
    UI.getAutocomplete();
    $( "div" ).on( "click", "input.form-control" , function() {
        if($(this).attr("name").indexOf("prefix") > -1 &&
autoComplete_prefix.length > 0)
            $(this).autocomplete({
                source: autoComplete_prefix,
                select: function(event, ui) {
                    setTimeout(function() {
                        UI.inputChanged();
                    }, 500);
                }
            });
        if($(this).attr("name").indexOf("data") > -1 &&
autoComplete_predicate.length > 0)
            $(this).autocomplete({
                source: autoComplete_predicate,
                select: function(event, ui) {
                    setTimeout(function() {
                        UI.inputChanged();
                    }, 500);
                }
            });
    });
});

```

```
    }  
);
```

submit.html inner js

---

```
//Global variables for autocompletion and the evidence graph.
```

```
var currentValue = 2;  
var currentSize = 1;  
var autoComplete_prefix = null;  
var autoComplete_predicate = null;  
var g;
```

```
//Global variables for handling confidence intervals.
```

```
var minimumConfidence = 100;  
var evidenceExists = false;  
var evidenceFields = "";  
var textConcat = "";  
var textDropDowns = "";  
var textData = "";  
var html;
```

```
//Class definition for an empty userInterface;
```

```
function userInterface() {}
```

```
/*
```

Description: Initialize the size value on the page and the gauge variable.

Pre-Condition: The currentSize, the input element with id "size" exists and the div element with id "gauge" exists.

Post-Condition: The input element with id "size" has the initial UI size and the "gauge" id element has basic settings.

```
*/
```

```
userInterface.prototype.initialSetup = function()
```

```
{}
```

```
/*
```

Description: Add a modular element to the page for an extra entry.

Pre-Condition: The div element with id "holder" exists.

Post-Condition: The input element with id "size" has the current UI size and the div element with id "holder" has a new element.

```
*/
```

```
userInterface.prototype.performAdd = function()
```

```
{}
```

```
/*
```

Description: Remove the last modular element from the page (assuming there is more than one)

Pre-Condition: The elements with id "xtra-clear, xtra-prefix, x-tra-select,x-tra-text-input, and size" exist.

Post-Condition: The elements with id "xtra-clear, xtra-prefix, x-tra-select,x-tra-text-input" no longer exist.

The input element with "size" contains the current UI element count.

```
*/
```

```
userInterface.prototype.performSubtract = function()
```

```
{}
```

```
/*
```

Description: Handler for change events for select fields.

Pre-Condition: Page is loaded.

Post-Condition: UI changes ("drop!" / "short-code:prefix" / "prefix-split") have occurred.

Post-Condition: Confidence evaluation has occurred.

```
*/
```

```
userInterface.prototype.selectionChanged = function()
```

```
{}
```

/\*

Description: Verifies if evidence is existing. If it is, recalculates the evidence interval.

Pre-Condition: Page is loaded.

Post-Condition: Evidence data is extracted into variables.

Post-Condition: UI changes ("text") has occurred.

\*/

userInterface.prototype.verifySelected = function( index )

{

/\*

Description: Extracts page info for confidence intervals.

Pre-Condition: Confidence has been found in the page.

Post-Condition: Data from page extracted.

\*/

userInterface.prototype.calculateConfidence = function()

{

/\*

Description: Calculates confidence ranking.

Pre-Condition: Data from page extracted into variables.

Post-Condition: Number from 0 to 100 assigned for confidence.

\*/

userInterface.prototype.finalizeConfidence = function()

{

/\*

Description: Extractor for auto-complete data.

Pre-Condition: Page is loaded.

Post-Condition: autoComplete\_prefix and autoComplete\_predicate have autocomplete items from the RDF database.

\*/

userInterface.prototype.getAutocomplete = function()

{

```
/*
```

Description: Imports database from XML.

Pre-Condition: Page is loaded and database

Post-Condition: autoComplete\_prefix and autoComplete\_predicate have autocomplete items from the RDF database.

```
*/
```

```
userInterface.prototype.importDatabase = function()
```

```
{}
```

```
/*
```

Description: Input Sanitizing

Pre-Condition: String is not null.

Post-Condition: Sanitized string for query creation.

```
*/
```

```
userInterface.prototype.sanitizeInput = function(str)
```

```
{}
```

```
/*
```

Description: Submission method.

Pre-Condition: Page loaded.

Post-Condition: Table created with search results.

```
*/
```

```
userInterface.prototype.submit = function()
```

```
{}
```

```
$(document).ready(
```

```
    function()
```

```
{
```

```
    var UI = new userInterface();
```

```
    UI.initialSetup();
```

```
    //Deal with adds.
```

```
    $( "#plus" ).click(UI.performAdd);
```



```

//Deal with removals.
$("#minus" ).click(UI.performSubstract);

$("#submit").click(UI.submit);

//Deals with bulk import disabling if it's not supported in other browsers.
if (typeof window.FileReader !== 'function') {
    $("#bulk-import").attr("disabled","true");
}
else
{
    $("#bulk-import").click(UI.importDatabase);
}

//Get all selected values to be printed.
$("#div").on('change', 'select.form-control', UI.selectionChanged);
$("#div").on('change', 'input.form-control', UI.selectionChanged);

//Attach auto-complete handlers to each input field on click.
UI.getAutocomplete();
$("#div" ).on( "click", "input.form-control" , function() {
    if($("#this").attr("name").indexOf("prefix") > -1 &&
autoComplete_prefix.length > 0)
        $(this).autocomplete({
            source: autoComplete_prefix,
            select: function(event, ui) {
                setTimeout(function() {
                    UI.inputChanged();
                }, 500);
            }
        });
});

```

```

        if($(this).attr("name").indexOf("data") > -1 &&
autoComplete_predicate.length > 0)
            $(this).autocomplete({
                source: autoComplete_predicate,
                select: function(event, ui) {
                    setTimeout(function() {
                        UI.inputChanged();
                    }, 500);
                }
            });
    });
}
);

```

database\_explorer.html inner js

---

//Class definition for an empty userInterface;

function userInterface() {}

/\*

Description: Draws the entire database graph.

Pre-Condition: Page is loaded.

Post-Condition: A graph is created on the page.

\*/

userInterface.prototype.drawGraph = function(value)

{

\$(document).ready(

    //Get the entire database.

    \$.ajax({

        type : "GET",

        url : "http://iie-dev.cs.fiu.edu/scripts/bulk\_export.php",

        success: function(data)

```

    {
        //Parse the XML into CSV format.
        var value = "";
        $(data).find("result").each(function()
        {
            $(this).find("binding").each(function()
            {
                var targs = $(this).first().text().split("/");
                value +=
targs[targs.length-1].replace("<", "").replace(">", "").trim() + "," ;
            });
        });

        //Remove extra commas and draw the graph.
        value = value.substring(0, value.length - 1);
        var UI = new userInterface();
        UI.drawGraph(value);
    }
})
);

```

```

custom_search.html inner js
<script type="text/javascript">
$(document).ready(
    //Basic Functionality.
    function()
    {
        $("#tags").val("untaggedQuery");
        $( "#add" ).click(

        //Store the data.

```

```

function()
{
    var tagData = $("#tags").val();
    var queryData = $("#query").val();

    //Ajax call to adding-query to DB.
    $.ajax({
        type : "POST",
        data : {query : queryData, tags : tagData},
        url :
"http://iie-dev.cs.fiu.edu/scripts/add-to-db.php",
        success: function(data){
            if(data.indexOf('unsuccessful') < 0)
            {
                console.log(data);
                $("#query-area").append('<div
class="alert alert-success" align="center" role="alert">Query Successfully
Stored</div>');

                setTimeout(function () {

$("#query-area").children().remove('div');

                }, 5000);
            }
        }
    });
}

$( "#search" ).click(
    //Retrieve the data.
    function()
    {
        var tagData = $("#tags").val();

```

```

//Ajax call to retrieve query from DB.
$.ajax({
    type : "POST",
    data : {tags : tagData},
    url :
"http://iie-dev.cs.fiu.edu/scripts/search-db.php",
    success: function(data){
        $("#query").val(data);
    }
});
}
);
}
);

```

## **Appendix D - Diary of meeting and tasks.**

**Date: 9/3/2014**

Meeting Begins: 6:03PM

Meeting Ends: 6:45PM

Medium of Communication: Skype

Present Members: Eric, Jose, Lazaro

6:03PM - 6:15PM Eric (Formal Introductions, Intro to Inference Engine)

6:15PM - 6:30PM Use Case Elicitation

6:30PM - 6:33PM Architecture concerns and discussion

6:33PM - 6:33PM Meeting with client formally disbands

6:33PM - 6:45PM Jose and Lazaro discuss required documentation, recap about requirements.

Assignments:

Lazaro and Jose will install Mulgara and attempt to execute basic queries, we'll also be looking at bootstrap-based approach to generating simple queries from a web form.

Lazaro and Jose will also begin writing the required initial drafts for our Feasibility Study, Requirements Document and Project Plan.

**Date: 9/6/2014**

Meeting Begins: 11:30AM

Meeting Ends: 6:00PM

Medium of Communication: Physical Meeting

Present Members: Jose, Lazaro

11:30AM - 1:30PM: Trello Board Upgrades and Discussion

1:30PM - 4:30PM: Feasibility Study Documentation

4:30PM - 5:30PM: Requirements Documentation

**Date: 9/21/2014**

Meeting Begin: 10:00 AM

Meeting Ends: 3:00 PM

Medium: Physical Meeting

Present members: Jose, Lazaro

In this meeting Lazaro started work on implementing the UI of the website. The initial UIs were done. He also installed Mulgara and testing it to make sure it worked. Jose begin looking for a library to do the REST calls that the UI needed to make in order to make queries. He found a library and started testing it to make sure that it had everything that was needed for the project

**Date: 10/4/2014**

Begin : 11:00 AM

End : 4:00 PM

Medium : Physical Meeting

Present Members : Jose, Lazaro

In this meeting Lazaro worked mainly on getting the confidence ranking to show up on the forms when evidence was added. Apart from that, he also included a lot of bug fixes for code that was previously submitted. Jose worked on the PHP scripts that are going to generate SPARQL queries based on what users entered in the forms. He also worked on some bug fixes for the scripts.

**Date: 10/19/2014**

Begin : 11:00 AM

End : 4:00 PM

Medium : Physical Meeting

Present Members : Jose, Lazaro

In this meeting Lazaro began working on the custom query subsystems of the system. He also continued to work on the previous forms. Jose continued testing and developing the PHP scripts so that they generated valid SPARQL queries and correctly connected to the server.

**Date: 10-28-2014**

Meeting Begins: 4:30PM

Meeting Ends: 5:15PM

Medium of Communication: Skype

Present Members: Jose, Eric

4:30 - 4:40 Discussed what parts of the system I was showing today

4:40 - 4:50 Talked about the data entry form and how predicates worked

4:50 - 4:55 Discussed how predicates currently worked and changes to be made

4:55 - 5:05 Talked about the data search form and some specific queries

5:10 - 5:15 Talked about some good ontologies to use

**Date: 10-31-2014**

Meeting Begins: 11:30

Meeting Ends: 1:30

Medium of Communication: Physical Meeting

Present Members: Jose, Eric, Lazaro

11:30 - 11:50 Tour of Eric's workplace

11:50 - 12:20 Lazaro showed Eric some of the features of the system and asked for comments

12:20 - 12:30 Eric talked to us about how he would like to be implemented

12:30 - 12:50 We discussed about some good ontologies to use and which default ontologies we should use



12:50 - 12:55 Jose asked about the web crawler and if Eric has any recommendations

12:55 - 1:10 Eric explained what he wanted the crawler to do as we thought it had a different function

1:10 - 1:30 We got some minor features that Eric wanted us to implement and also got comments on the current system

**Date:11/1/2014**

Begin : 11:00 AM

End : 4:00 PM

Medium : Physical Meeting

Present Members : Jose, Lazaro

In this meeting, we discussed the need to change our core technology. We decided it was necessary to replace Mulgara with something that supported everything that was necessary for the project. Lazaro continued work on the front end forms. Jose continued work on the scripts and modified them for the new back end triple store that was going to be used.

**Date: 11/17/2014**

Begin : 11:00 AM

End : 4:00 PM

Medium : Physical Meeting

Present Members : Jose, Lazaro

Jose modified the PHP scripts so that they returned XML and prepared the PHP scripts to be retired. He also began work on the javascript library that were going to replace the PHP script for query generation. Lazaro worked on getting auto-complete for the different text fields working. He also added the base code import and export.

**Date: 12/1/2014**

Begin : 11:00 AM

End : 4:00 PM

Medium : Physical Meeting

Present Members : Jose, Lazaro

Lazaro began the testing of the front end part of the system. Jose added some of the javascript files for query generation.

Date: 12/6/2014

Begin : 11:00 AM

End : 6:00 PM

Medium : Physical Meeting

Both Jose and Lazaro finalized their code for their respective parts and began integrating the parts together. Both also fixed any bugs that came up from integration in their respective parts. Both continued to test their parts and tested the integration.

## References

- [1] - S. Karnouskos: *Stuxnet Worm Impact on Industrial Cyber-Physical System Security*. In: *37th Annual Conference of the IEEE Industrial Electronics Society (IECON 2011)*, Melbourne, Australia, 7-10 Nov 2011. Retrieved 20 Apr 2014.
- [2] - Jack Rusher, [Semantic Web Advanced Development for Europe](#) (SWAD-Europe), Workshop on Semantic Web Storage and Retrieval - Position Papers
- [3] - <http://www.w3.org/TR/PR-rdf-syntax/> "Resource Description Framework (RDF) Model and Syntax Specification
- [4] - Berners-Lee, Tim; James Hendler; Ora Lassila (May 17, 2001). "[The Semantic Web](#)". *Scientific American Magazine*
- [5] - Hebel, John; Fisher, Matthew; Blace, Ryan; Perez-Lopez, Andrew (2009). *Semantic Web Programming*. Indianapolis, Indiana: [John Wiley & Sons](#). p. 406. ISBN 978-0-470-41801-7