DEVELOPMENT OF MOBILE SEARCH APPLICATIONS OVER STRUCTURED WEB DATA THROUGH DOMAIN-SPECIFIC MODELING LANGUAGES

Supervisor: Assistant Prof. Marco BRAMBILLA

Master graduate thesis by:

Atakan ARAL    Id number: 749327

Ilker Zafer AKIN   Id number: 749253

Academic Year: 2010/2011
Abstract

Demand on mobile phone applications is arising day by day, because users want to reach everything they do from their Personal Computers. Therefore, there is an increasing interest on mobile phone applications.

Search Computing is a project which enables users to make multi-domain search. By having chance of mobility, in order to make users use this system more, a mobile version was needed. That’s why; the aim of this project is developing a mobile search application using a language which is compatible with all mobile phones. In this sense, mobl Language has been chosen since it works on the most important mobile operating systems which lead to the market. By this application, while users don’t need to download any application to their phones, thanks to HTML5, they will enjoy the advantages of the newest technology.

The main aim of this project is letting users to search for things to do connecting them to each other. That is, if the user wants to go to a concert that is close to the restaurant in which he/she wants to eat before and stay in a hotel after the concert, he/she can find the best combination via the Search Computing system through his/her mobile phone. The application is connected to the Search Computing (SeCo) system via API of the server and uses JSON files coming from the server. Moreover, the application contains Google Map on which users can interact.

In this document, it can be found information about the Search Computing project, information about the API of SeCo, the analysis of the system and the language, the design principles in order to make the application more usable and the implementation of the application.

Keywords: Mobl Language, Search Computing (SeCo), Structured web data, Mobile Application, Multi-domain Search.
# Table of Contents

1 Introduction ................................................................................................................. 1

2 Theoretical Background ................................................................................................. 2
   2.1 Search ..................................................................................................................... 2
       2.1.1 Multi-domain Search .................................................................................... 2
       2.1.2 Exploratory Search ....................................................................................... 5
   2.2 Brief Mobile History .................................................................................................. 6
   2.3 mobil Language .......................................................................................................... 7
   2.4 Design Principles of Mobile GUIs ............................................................................ 8
       2.4.1 General Design Principals ............................................................................ 9
       2.4.2 User Interfaces for Searching ...................................................................... 10
       2.4.3 Visualization of Information ........................................................................ 11

3 Background on Search Computing .................................................................................. 14
   3.1 What is Search Computing (SeCo)? ......................................................................... 14
   3.2 Infrastructure of SeCo .............................................................................................. 15
       3.2.1 Search Computing Framework ..................................................................... 15
       3.2.2 Service Marts for SeCo .............................................................................. 17
       3.2.3 Web Service Registration and Adaptation .................................................. 21
   3.3 SeCo API ................................................................................................................ 24
       3.3.1 Mart Repository ............................................................................................ 24
       3.3.2 Query Repository ........................................................................................ 26
       3.3.3 metadata.JSON ............................................................................................ 34
       3.3.4 Connection in Mart Repository .................................................................... 35
       3.3.5 Connection in Query Repository .................................................................. 37

4 Analysis and Design ....................................................................................................... 39
   4.1 Overview of the Project ........................................................................................... 39
   4.2 Design of Mobile Interface ....................................................................................... 43
       4.2.1 Tab Set ........................................................................................................... 43
       4.2.2 Item List ........................................................................................................ 44
       4.2.3 Accordion ...................................................................................................... 45
       4.2.4 Google Map .................................................................................................. 46
       4.2.5 Table ............................................................................................................. 46
       4.2.6 Form Elements .............................................................................................. 47
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.7</td>
<td>Buttons</td>
<td>48</td>
</tr>
<tr>
<td>5</td>
<td>Implementation of the Project</td>
<td>49</td>
</tr>
<tr>
<td>5.1</td>
<td>Data Model</td>
<td>49</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Server-Side Technologies</td>
<td>50</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Client-Side Technologies</td>
<td>51</td>
</tr>
<tr>
<td>5.2</td>
<td>User Interface</td>
<td>53</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Root Screen</td>
<td>54</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Service Mart Screen</td>
<td>55</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Access Pattern Screen</td>
<td>56</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Interfaces Screen</td>
<td>57</td>
</tr>
<tr>
<td>5.2.5</td>
<td>Form Screen</td>
<td>58</td>
</tr>
<tr>
<td>5.2.6</td>
<td>Results Screen</td>
<td>61</td>
</tr>
<tr>
<td>5.2.7</td>
<td>Compare Selection Screen</td>
<td>64</td>
</tr>
<tr>
<td>5.2.8</td>
<td>Compare Screen</td>
<td>64</td>
</tr>
<tr>
<td>5.2.9</td>
<td>Show Map Screen</td>
<td>65</td>
</tr>
<tr>
<td>5.2.10</td>
<td>Marker Details Screen</td>
<td>67</td>
</tr>
<tr>
<td>5.2.11</td>
<td>History Screen</td>
<td>67</td>
</tr>
<tr>
<td>5.3</td>
<td>Application Logic</td>
<td>68</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Form Screen Script Blogs</td>
<td>69</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Get Caption Function</td>
<td>70</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Fields in Constraints Function</td>
<td>71</td>
</tr>
<tr>
<td>5.3.4</td>
<td>Results Screen Script Blogs</td>
<td>71</td>
</tr>
<tr>
<td>5.3.5</td>
<td>Get Main Output Function</td>
<td>73</td>
</tr>
<tr>
<td>5.3.6</td>
<td>Compare Screen Script Blog</td>
<td>73</td>
</tr>
<tr>
<td>5.3.7</td>
<td>Choose Result Function</td>
<td>73</td>
</tr>
<tr>
<td>5.3.8</td>
<td>Delete Function</td>
<td>75</td>
</tr>
<tr>
<td>5.3.9</td>
<td>Reset Function</td>
<td>75</td>
</tr>
<tr>
<td>5.3.10</td>
<td>Go To Interface Function</td>
<td>76</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion and Future Work</td>
<td>77</td>
</tr>
<tr>
<td>7</td>
<td>Bibliography</td>
<td>78</td>
</tr>
</tbody>
</table>
Figure Index

Figure 1: An example of ambiguous domain result list layout for Query 2. [3] ................................................. 4
Figure 2: An example of multi-domain result list layout for Query 3. [3] .......................................................... 5
Figure 3: Types of the search activities and the role of exploratory search [14] .................................................. 6
Figure 4: Map Visualization of results for hospital search ................................................................................. 12
Figure 5: Timeline Visualization of results for transportation search ............................................................. 13
Figure 6: Overview of the Search Computing Framework [8] ............................................................................ 17
Figure 7: Data extraction from query results [8] ............................................................................................... 23
Figure 8: Materialization process [8] .............................................................................................................. 24
Figure 9: Hierarchy of objects in the data source registry [9] ............................................................................. 30
Figure 10: Hierarchy of the Mart Repository ..................................................................................................... 36
Figure 11: Flowchart of the Query Execution ................................................................................................... 38
Figure 12: Use Case Diagram .......................................................................................................................... 41
Figure 13: Component Diagram ........................................................................................................................ 41
Figure 14: Activity Diagram ............................................................................................................................. 42
Figure 15: Sample tab sets from amazon.com ................................................................................................... 43
Figure 16: Two sample item lists from iPhone web applications .................................................................... 44
Figure 17: A sample accordion ........................................................................................................................ 45
Figure 18: A Sample Google Map view ............................................................................................................ 46
Figure 19: A sample table with advanced features .......................................................................................... 47
Figure 20: Date picker and time picker for iPhone ............................................................................................ 47
Figure 21: a) Tab set with search tab selected b) Tab set with history tab selected ........................................... 54
Figure 22: a) Access patterns screen b) Interfaces screen .................................................................................. 56
Figure 23: Two examples of form screens ......................................................................................................... 58
Figure 24: Two examples of Results Screen ...................................................................................................... 61
Figure 25: a) Compare selection screen b) Compare screen ............................................................................. 63
Figure 26: a) Show Map Screen b) Marker Details Screen .............................................................................. 65
Table Index

Table 1: URL Organization of Mart Repository [10] ................................................................. 26
Table 2: URL Organization of Query Repository [9] .................................................................... 29
Table 3: Mapping Between Data Types and Input Elements .......................................................... 60
1 Introduction

In today’s world, the importance of mobile technology and the importance of search have been increasing. In the past, since everything was on paper, searching something was taking days, nowadays; it only takes a few milliseconds. This improvement brings the expectations with itself. The user wants to find what he/she is looking for instead of unrelated results. That’s why; the companies are enhancing their search engines day by day.

Today, mobility is one of the most important measures in business. It can be seen that every website has a mobile version or an application which the user can do same things with the full site or desktop application. This increase has happened since the 3G (third generation) has been used. People would like to reach everything using their mobile phones or even today using their tablet PCs. This demand leads to increase of selling of smartphones/tablet PCs. Since the users ask for it, the companies produce the services according to the market request.

By developing the ability of mobile phones and services, not only general search applications but also specific ones are wanted. In this sense, Search Computing project which aims to serve a multi-domain search system needs to have a mobile version to make the system more reached using mobile phones. By this way, the user can easily search for something to do right away via his/her phone. Moreover, by using this system he/she can combine the things that he/she will do. For example, the user wants to first having a nice dinner and after that going to a cinema which is close to the restaurant and last staying in a five star hotel. Search Computing project provides this capability and more to the user.

In the following chapters, it can be found that some background data related to multi-domain search, exploratory search and Search Computing. Afterwards, it can be seen the analysis that is done on SeCo and on design of mobile applications. Next, the information about the implementation of the mobile platform for the Search Computing project can be found. In the end, conclusion about this project and recommendations for the future work has been indicated.
2 Theoretical Background

2.1 Search

Search is becoming the key feature of the computer science. When the search word is said, the first thing comes to mind is Web-search. However, it is not the only search in the computer science. It can be local search in the directories of computer, a word in a file, etc. Solely, a brief explanation will be given about web-search since it is related to the project.

A web search engine looks for information on the World Wide Web based on the keyword that is given by the user. The results can be web pages, images, videos and etc. Nowadays, web-crawling which searches the web automatically is a standard for the web search engines as well as indexing and searching. Indexing is used to store the crawled web pages and to match the best with the query user identified.

In the following sub-section, multi-domain search which is related to this project has been studied.

2.1.1 Multi-domain Search

In the last years, digital data increased rapidly which causes that the applications need to deal with huge repositories and to help users in finding information. The best example for this is search engines which quickly had an important role in any kind of information system. Since the user requirement changed depending on the amount and kind of information became larger, search engines were supposed to adapt themselves. [5]

There are two different trends for searching information. First trend is to look for information and objects of interest instead of Web pages which illuminate such objects. Hence, they expect to be satisfied for their need by search engine directly. For example, when the temperature of New York City is asked to search engine, he/she expects it to show directly this information instead of a list of pages handling it. The other trend is related to user experience. When the users get confident about using search engines, the queries become more complex which can cause their formulation mean more than what can be expressed with a few keywords. Therefore, answers must be more structured than a list of Web pages. [4]

In order to meet the requirements, Web search engines should change not only their algorithms, but also interfaces to different information domains, i.e. semantic fields of interest such as cities, people, hotels, etc. Moreover, result sets can contain different types, such as
images, videos, etc., for each domain of interest. The domain, content type and layout of the result set should meet the user expectations. [3]

Here are 3 different query examples [3]:

- Query 1: “Washington D.C.”, it is a mono-domain query which concerns only one semantic domain, the City domain. The only issue is visualization of the specified domain.
- Query 2: “Washington”, it is an ambiguous domain query which can include many semantic domains. Actually, it may be the capital of the U.S., the first U.S. president or the state. In order to show the results related to user intent, the query should be identified. If it is not possible, the result set should grant coverage of the most expected intents.
- Query 3: “rock concert Washington July 2010 good restaurant”, this is a multi-domain query which needs to deal with several domains. The query refers to cities, restaurants and concerts. The main issue is to build a result set connecting different concepts together.

These queries can produce different type of information. For example, the result of US capital can be map, news, images etc. Generally, different result types come from different search engines and need to be aggregated. This operation should be done carefully but sadly no attention has been devoted of displaying of search results. [3]

The visualization of search results is an important issue for search engines in order to make user perceive quality. [6] Web search engines such as Google, Yahoo! And Bing, are also called general-purpose Web search engines, nowadays contain domain-specific functionalities such as image, video, and so on in order to present better results. However, capability of dealing with mono-domain and ambiguous domain queries limit their performance. In the Query 1 above, the results coming from different domain-specific search engines (for showing map of the city) are aggregated by the engine. For Query 2, a result diversification task has been performed by a general-purpose engine, splitting the result list in order to include information related to all the identified domains. [3]
Besides, search systems do not manage multi-domain queries correctly. Search Computing (SeCo) is the only approach which takes into account this issue. SeCo proposes some methods and tools for handling multi-domain queries by automatically unite the results coming from diverse search services to create an extensive answer, combining relevant pieces of information from different domains. It aggregates results at system level and it produces a single result list including the combinations of results from different domain. “Engines like the one provided by SeCo enable the computation of queries like Query 3, possibly adding functionalities such as global ranking, top-k result calculation, multiple visualization, and so on.” [3]
2.1.2 Exploratory Search

Interaction carried out by the user in attempt to search for information is getting more and more complex with the recent user behavior trends. New applications for searching are required to fulfill the need of a method for information acquisition that is effective throughout the lifecycle of the search. User should be aided in formulating his/her interest, in exploring most relevant and credited information sources and in correlating the elements of those sources. [22]

Exploratory search provides the information exploration for the users that are under either of the following categories: [19]

- Users who are not knowledgeable in the domain they are conducting their search
- Users who are not knowledgeable about how to conduct their search due to either technology or process
- Users who are not sure about the goal of their search

[19]
Search activities can be categorized into three: lookup, learn and investigate. First kind of search activity in which exploratory search is not particularly effective is already one of the most successful applications of computers. However exploratory search can make significant contributions to the other two categories of search activities. It should be also note that those categories are not independent sets as most users perform those three kind of searches in parallel or nested [14].

2.2 Brief Mobile History

Before 2007, mobile devices were not capable enough to run applications which are not pre-installed on them. Smartphones could do but the market was fragmented. All the brands had their own operating systems, like Nokia, Samsung, Ericsson and etc. The aim of Sun by announcing Java 2 Mobile Edition was to give change to developers building an application only once and run it on any platform. However, it was not much successful because of the screen sizes, J2ME implementation and etc. [12]

Many phones only supported WAP (Wireless Application Protocol), which is a sub-set of HTML, was very limited. Also, internet connection was not fast as well. After iPhone is introduced mobile internet experience was good enough to develop applications for the platform. But it was not successful because hardware of iPhone was still slow for web experience, the browser was also slow, and HTML5 did not exist and web applications didn’t
have offline capabilities. That’s why, iPhone SDK and AppStore were introduced by Apple in order to let developers build and sell the iPhone applications. [12]

Although there are many tough problems with AppStore; such as approval for an application, updating an application, etc.; many companies such as Android by Google tried to follow the same way after the success of Apple. However, having many platforms have made a trouble for developers, since they all use different languages. But today, smartphones support many new HTML5 technologies, including Web Databases, ability to find out the location, offline support, multi-touch, canvas, CSS3. “With support from most of the manufacturers of modern mobile phones we can conclude that developing mobile web applications are an interesting alternative to developing native applications for each platform individually.” [12]

2.3 mobl Language

Nowadays, the importance of mobile technology is known by everyone. Mobl is a free and open source language to build mobile applications. Its first aim is speeding up the development and it allows you to build applications for many platforms with only one code, thanks to HTML5 technology. [11]

Although mobl is not the first toolkit for developing mobile web applications, it is identified a few problems with the others. In mobl, another approach has been chosen: Instead of building a JavaScript framework it is built a domain-specific language for mobile web applications. “That is, a language specifically designed to build mobile applications, with special keywords and language constructs for concepts such as controls, screens and model entities.” [12]

The features offered by Mobl can be grouped as follow: [11]

- Platform:
  - mobl generates 100% client-side HTML5-based mobile web applications.
  - The applications that are developed by mobl has offline working capability. That is, there is no need an internet connection.
  - Static JavaScript/HTML5/CSS files are generated by compiler, and they are ready to be deployed.
  - iOS, Android, WebOS, Safari and Chrome support the applications developed with mobl.
- **Language:**
  - Mobl is a statically typed language which enables IDE support and error detection.
  - All aspects of a mobile web application are integrated into a single language: data modeling, user interfaces, application logic, styling and web services.

- **Data Modeling:**
  - Data is stored on the device, no server required.
  - SQL is not needed since mobl handles all database access, creates the database schema and manages object persistence.
  - Mobl’s query language is used to perform queries instead of SQL embedded in strings.

- **User Interface:**
  - User interfaces automatically update to reflect changes in application state.
  - Domain-specific language (DSL) for defining user interfaces.

- **Application Logic:**
  - It is scripting language which has a very similar to JavaScript syntax
  - It allows to access existing JavaScript libraries and widgets easily.

- **Webservice Access:**
  - It can access to web services through AJAX easily.
  - It allows you to import JSON data into local database lightly.

### 2.4 Design Principles of Mobile GUIs

User interface is one of the most important factors an application is assessed by its user since it the part where they mostly interact. Even if the application is capable of performing expected tasks exactly, a poorly designed interface can harm its usefulness significantly. Design of the user interface is mostly about the way its users think and work rather than the capabilities of the device. [15]

Design principles that are applicable to both mobile devices and personal computers or the ones that are specific to mobile devices are explained in the following sub-section. User interface ideas related to searching and mobile searching in particular are given in the next one while the last sub-section is about the visualization of information which is useful for search applications in order to display their results efficiently.
2.4.1 General Design Principals
Following design principals are applicable to any system with user interaction including pc programs, mobile applications, web applications and web sites [18]

- Accessibility: It should be designed to be usable by as many people as possible, without modification.
- Aesthetical Beauty: Its elements should be aesthetically in contrast, aligned, grouped and colored.
- Availability: All of its objects should always be available.
- Clarity: It should be visually, conceptually and linguistically clear.
- Compatibility: It should be compatible with the user, with the task and with the product. It should adopt the user’s perspective.
- Consistency: Look, use and operation of its similar components should be the same and it should always produce the same result for the same action.
- Controllability: Interaction should be controlled by the user; ways to perform actions should be flexible and customizable.
- Directness: Direct methods for performing actions should be provided with visible alternative methods.
- Efficiency: Amount of control actions such as eye and hand movements should be minimal.
- Familiarity: Concepts and language should be natural and familiar to the user possibly by using metaphors for real world objects.
- Flexibility: It should be sensitive to different needs of its users or different conditions.
- Forgiveness: It should be tolerant and able to recover human errors if it cannot prevent them.
- Immersion: It should be as unobtrusive as possible in order to foster immersion.
- Obviousness: Usage of it should be easy to learn.
- Operability: It should be usable by everyone, regardless of a person's physical capabilities.
- Perceptibility: It should be perceivable by everyone, regardless of a person's sensory capabilities.
- Positive First Impression: It should lure the user with a positive first impression.
• Predictability: Its way of operation should be easily anticipated by its user with the help of cues and recognizable screen elements.
• Recovery: It should be able to recover itself after a user error or a technical problem.
• Responsiveness: User actions should be responded rapidly and with useful visual, and/or auditory feedback.
• Safety: It should protect its user against human errors by providing cues.
• Simplicity: It should be designed simple for example by hiding things until they are needed and/or by providing defaults.
• Transparency: It should not bother the user with the technical issues of the application.
• Trade-Offs: Peoples requirements should always take precedence over technical ones when they conflict.
• Visibility: Its status and usage should be clearly visible.

[18]

As distinct from personal computers (laptop or desktop), mobile devices share some common characteristics such as:

• Less computational capacity
• Smaller screen size and resolution [15]
• Ability to use in different orientations like landscape and portrait [15]
• Responds to hand gestures instead of clicks [15]
• User can interact with a single application and a single screen of it at the same time. [15]
• Different application goals and user expectations [16]
• Different kind of usage environment [16]

While most of the previously given principals are perfectly valid for mobile devices, those characteristics and limitations cause some changes on the significance of them. Some design principals such as directness, responsiveness and simplicity are more important in the case of mobile devices, while the ones such as operability and perceptibility are not directly applicable. [16]

2.4.2 User Interfaces for Searching

Searching interfaces are typically designed extremely simple, sometimes as simple as only a field to enter the query [16]. This tradition is legitimate one supported by following reasons:
• Searching and its interface is not the goal for the user. He/she is interested with the information.
• Searching is a mentally intensive task which require user’s attention without any distracting factors.
• User base (people who seek information through computers) is highly diverse. That situation causes an incline towards simpler interfaces.

[6]

In addition to simplicity, following design principles are proposed for search interfaces [6]. Although they usually coincide with general design principals, they are presented again with examples for searching.

• Efficient and informative feedback should be provided. Some examples are: relevance indicators, term suggestion or correction, document surrogates.
• Amount of the user control about matters such as ordering of results or transformations applied to user query, should be delicately balanced.
• Relevant information to the user search such as search history or usability hints, should be displayed.
• Hints and shortcuts for the experienced users should be provided.
• User errors should be minimized by spelling correction, term expansion and supporting synonymies.
• Pages should be designed consistently without being distractive.
• User should be able to reverse his/her actions, for example cancel a query.

[6]

Surveys show that average query length is smaller and variation is narrower for mobile devices [20]. These results can be easily associated with the slowness and the difficulty of typing with smaller devices. Techniques such as dynamic term suggestion (auto-completion), anticipating common queries and voice-entered query terms can be used efficiently to decrease the effects of this limitation and improve query entry [6].

2.4.3 Visualization of Information
Although result interfaces for mobile searches do not differ significantly from desktop counterparts [21], it is also possible to apply visualization methods for particular result types.
For example, using map visualization is useful for results that contain location information [6].

When the results of the user search contain geographic coordinates, those results can be intuitively visualized on a map. Each result is then represented with a point (or marker) on the map which is selected so as to include points in the context. Additional information can also be displayed on the map by using different symbols, sizes or colors for markers [17].

This kind of visualization allows only a relatively small number of results to display on the map. As a result, scrolling and zooming mechanisms are required to be able to see more results [17].

Figure 4: Map Visualization of results for hospital search

Another kind of visualization appropriate for particular result types is timeline visualization. Timelines are useful for displaying results with time or date information. Similar to map visualization, extra information can be displayed by using different symbols [17].
Figure 5: Timeline Visualization of results for transportation search
3 Background on Search Computing

3.1 What is Search Computing (SeCo)?

Search computing is a multi-disciplinary science which will provide the abstractions, foundations, methods, and tools required to answer multi-domain queries over heterogeneous data sources. It is a project started on November 1st, 2008 and lasts until October 31, 2013 funded by European Research Council (ERC). Its aim is finding answers to complex search queries such as “Where can I attend an interesting conference in my field close to a sunny beach?” by cooperating search services, user ranking and joining of results. [1]

“Search Computing aims at responding to queries over multiple semantic fields of interest; thus, Search Computing fills the gap between generalized search systems, which are unable to find information spanning multiple topics, and domain-specific search systems, which cannot go beyond their domain limits.” [8]

Since it is a multi-domain search platform, it needs to combine its results extracted from multiple web sources. That’s why, common techniques for crawling and indexing, which check only one Web page, are not decent for this. [2]

Let’s talk about a detailed example given above: “Where can I attend a scientific conference in a city within a hot region served by luxury hotels and reachable with cheap flights?” A knowledgeable user would do a multi-domain search step by step. In the first phase, the user would search conferences by city in a database. According to results of the cities, he/she would check the temperature of the city if it is warm enough or not. After that, he/she would check the flights in the manner of price to go that city. In the last step, a hotel is looked for from another system. Instead of these steps, SeCo aims to provide a system for supporting another type of search process shown below.[1] By this type of search, it is easy to use and find the exact solution.

- Several solutions which integrate all dimensions are built.
- A global rank function is used to rank solutions which are shown in rank order
- Browsing the result and typing query are supposed to be user-friendly.
- Search domains while the search proceeds can be added.
- The relative weight of each ranking is supposed to change.
“Answering multi-domain queries requires the combination of knowledge from various domains. These queries are hardly managed by general-purpose search engines, because they cannot be found on a single page, where a page is the classical unit of crawling and indexing.”[2] On the contrary, domain-specific systems normally depend on knowledge of field and user expertise.

The individual search results in a specific domain are likely to be ranked by some criteria. In order to integrate the results coming from different domains, multi-domain search, it has to be done manually or by a custom program. However, it is not supported by data integration platforms. Search Computing supplies a platform in order to imply requests over multi search services, where the rankings of individual search results are taken into account of the results of the integrated requests. [7]

The vision of SeCo is to develop technologies and architectures for two types of users:

- Content providers who would like to arrange their content for search access by third parties.
- Application developers who would like to offer new services built by composing domain-specific content against the existing general-purpose search engines such as Google. [2]

3.2 Infrastructure of SeCo

3.2.1 Search Computing Framework

As it is expressed in the previous sections, Search Computing is a multi-disciplinary approach upon related past researches which contain data integration, query generation and several variations of ranking in heterogeneous datasets. Abstractions, methods, tools and computing systems which are supposed to be in order to express multi-domain queries and their answers are provided. For instance, “Which drugs threat diseases those are likely to be associated with a given genetic mutation?” can be answered using a multi-domain query. The query can be broken down into sub queries (like “Which drugs threat which diseases?”,”Which diseases are likely to be associated with a given genetic mutation?”) and each domain-expert server which are registered in the system take care of related sub-query (in this case, calls to servers named “Drug4Disease”, “GeneticMutation2Disease”). After decomposition phase, since an internal format should be used after analyzing, it is supposed to be translated to that format. After being optimized, an efficient plan should be used for query execution which is
supported by an execution engine which sends service calls to services via a service invocation framework. Query results are built by combining the outputs created by service calls and global rankings of query results are computed. In the end, outputs of query results are served in the order that indicates their global ranking. [7]

A standard format, called service mart, has been implemented to make services available to Search Computing. “This is a conceptual abstraction that masks the different implementation styles of services and is tailored to the specific need to expose search services – i.e. services whose primary purpose is to produce ranked lists of results.” Through service interfaces, wrappers or direct access to extensional data collections such as databases, excel files; data sources are used to produce these results. In order to make use of Search Computing, in the Search Computing Framework these sources must be registered as services. The relationship between the service mart and the operation to be invoked on the service are defined for this purpose. [7]

The Service Mart Framework, shown in Figure 6, provides structure for wrapping and registering data sources, meanwhile, the Service Invocation Framework controls the technical issues concerning interaction with the Service Mart like Web service protocol and data caching.

The User Framework supplies storage for registering users which can have different authority and skills. The Query Framework manages and stores queries in order to execute, save, modify and publish for other users. The Query Processing Framework can be thought as the central component of the architecture since it gives service for multi-domain queries. The Query Manager divides the query into sub-queries and connects them to the responsible data sources, while the Query Planner creates a query execution plan for sequence. “Finally, the Execution Engine actually executes the query plan, by submitting the service calls to designated services through the Service Invocation Framework, building the query results by combining the outputs produced by service calls, computing the global ranking of query results, and producing the query result outputs in an order that reflects their global relevance.” [8]

There are two types of users in Search Computing. One of them is end users who are allowed to reach predefined applications and submit input through forms. On the other hand, expert users can also create queries to repositories of service marts and composition patterns. It can
be seen on the upper part of Figure 6 that an external API is used to access end-user applications and interfaces in order to call them from any client environment. [7]

![Diagram of the Search Computing Framework](image)

Figure 6: Overview of the Search Computing Framework [8]

### 3.2.2 Service Marts for SeCo

“Service Marts are simple schemas which match "Web objects" by hiding the underlying data source structures and presenting a simple interface, consisting of input, output, and rank attributes; attributes may have multiple values and be clustered within repeating groups.” Search Computing operations are supported by Service Marts, such as ranked access. Responses are ranked list of objects, when objects are accessed through Service Marts. This list is cut to avoid receiving too many irrelevant objects which is a typical behavior of search services to show only best ones. [8]

In the Search Computing Framework, Service Mart is defined as the data abstraction for data source. The aim behind defining Service Mart is to simplify the publication of search services, whose responses are ranked list of objects. Every Service Mart is matched to one “Web Object” exists on Internet, so there are Service Marts for “hotels”, “flights”, “doctors” and so on. Furthermore, some Service Marts are connected to each other to support their linking. Service Marts and their connections form a network which may be used as a high-level interface for queries. [8]
Mapping to several data sources, which may be an API, a Web Services or a materialized data collection, is required for implementing a Service Mart. Therefore, the Service Mart concept provides a regular view of the world. [8] In the following sections, a top-down view is given from the conceptual level to the physical level through the logical view.

### 3.2.2.1 Conceptual Level

“Service Marts have atomic attributes and repeating groups consisting of a non-empty set of sub attributes that collectively define a property. Atomic attributes are single-valued, while repeating groups are multi-valued.” [8] For instance, Service Mart for “Movie” has both single-value attributes (“Title”, “Director”, “Score, “Year”, “Language”) and repeating groups (“Genres”, “Openings”, “Actor”), each has sub-attributes. One level of parentheses is used for repeating groups, like in the following:

- Movie(Title, Director, Score, Year, Genres(Genre), Language, Openings(Country, Date), Actors(Name))
- Cinema(Name, Address, City, Country, Phone, Movies(Title, StartTimes))
- Restaurant(Name, Address, City, Country, Phone, Url, Rating, Category(Name)) [8]

In the Cinema and Restaurant Service Marts, “Movies” and “Category” are repeating groups respectively. Repeating groups mean many-valued properties in the object of the Service Mart. By this way, it is modeled 1:M or M:N relationships where purpose of conceptual elements is bridging real world objects. Between “actor” and “movie”, there is an “acts-in” relationship which is modeled by repeating groups, by putting actors in a repeating group of movie or movies in a repeating group of actor. This is done in order to keep the Search Computing infrastructure and connection between two Service Marts as simple as possible. In SeCo Framework, it is not used top-down process; instead, data sources are modeled bottom-up. Furthermore, since most data sources have a simple schema, they can be presented by a one-level nesting. [8]

### 3.2.2.2 Logical Level

In this level, each Service Mart is correlated with one or more specific access patterns which express the road can be used to access the Service Mart. It contains the characterization of each attribute or sub-attribute which is input (I) or output (O). Also, if the results are produced in an order, there is an output attribute called ranked (R) because the ranking process is done according to this attribute’s value. All the ranked values are normalized within the interval
[0…1] in order to make it simple. Here is an example of access pattern for the “Movie” Service Mart: [8]

- $Movie_1(Title^0, Director^0, Score^R, Year^0, Genres.Genre^I, Language^0, Openings.Country^I, Openings.Date^I, Actor.Name^0)$

- $Movie_2(Title^I, Director^0, Score^R, Year^0, Genres.Genre^0, Language^0, Openings.Country^I, Openings.Date^I, Actor.Name^0)$

In these access patterns, same attribute is used for ranking, “Score”, in descending order of movies’ score. “Country” and “Date” are input parameters for openings to search movies shown in a specific country with specific date. The difference between these two access patterns is that first one is used for searching movies according to its genre, while in the second one it is searched according to its title. “Other access patterns could be used for accessing movies by providing the director or one actor. The choice of access patterns is a limitation on the way in which one can obtain data, typically imposed by existing service interfaces. Therefore, defining access patterns requires both a top-down process (from query requirements) and a bottom-up process (from service implementations). In general, this tension between top-down and bottom-up processes is typical of service design.” [8]

In some cases, Service Mart may have less attribute than access patterns. Let’s explain this with an example: If a cinema or restaurant is considered, address is an important object for them. However, the user may search them according to his/her address as input and look for by proximity. That’s why; there are two versions of attributes for “Address”, “City”, and “Country”, one is for user’s location and the other is for cinema/restaurant’s location. [8]

- $Cinema_3(\text{Name}^0, UAddress^I, UCity^I, UCountry^I, TAddress^0, TCity^0, TCountry^0, TPhone^0, Distance^R, Movie.Title^0, Movie.StartTimes^0)$

3.2.2.3 Connection Patterns

“Connection patterns introduce a pair-wise coupling of Service Marts.” Each pattern has a name and a specification which shows the sequence of pairs of attributes or sub-attributes of two services. Connection patterns may be directed or undirected. For instance, “Shows” is an undirected connection pattern which uses a join on titles of Movies and Cinemas: [8]

- Shows(Movie, Cinema): [(Title=Title)]
In this case, if the title of movie is equal to the title of any movie shown in the cinema, then it is satisfied. Here is another connection between cinemas and restaurants which is a directed pattern. The direction is “from” the first “to” the second which means query first search cinemas and then close restaurants. The address of the cinema will be input location of a restaurant service, after finding a cinema close to the user’s address. [8]

- DinnerPlace(Cinema, Restaurant): [(TAddress=UAddress),
  (TCity=UCity), (TCountry=UCountry)]

“Logically, connection patterns are expressed among pairs of orderly type compatible attributes. A connection pattern must be supported by a pair of access patterns. All the attributes of both selected access patterns must have the same labels, either I or O, and they should not both be labeled I.” In order to be an undirected pattern, both left and right operand have an O label. If label O occurs in the left operand and label I occurs in the right operand, the pattern is directed from left to right. If it is visualized, connection patterns and Service Marts can be shown as resource graphs where nodes are marts and arcs/edges are connection patterns. Therefore, this model presents a simplification of reality. [8]

3.2.2.4 Physical Level

Service Interfaces are modeled at the physical level of Service Marts in which a concrete data source maps to each service interface. A service interface is not obliged to support all the attributes of the Service Mart. “A service interface is a unit of invocation and as such must be described not only by its conceptual schema or logical adornment, but also by its physical properties.” There are possibilities to characterize data-intensive services, both in a manner of performance and quality. Four types of parameters describe service interfaces: [8]

- **Ranking Descriptors**: It classifies the service interface as a search service which produces ranked result or an exact service which produces objects not ranked. In exact services, there is selectivity which is a positive number shows the average number of tuples created by each call. If a search service is connected with an access pattern having at least one output attributes as R, it is an explicit ranking which can be either ascending or descending; if not, it is an opaque ranking. It is not mandatory that search services present a result with ranked attributes. [8]

- **Chunk Descriptors**: “It deals with output production by a service interface. The service is chunked when it can be repeatedly invoked and at each invocation a new set of
objects is returned, typically in a fixed number, so as to enable the progressive retrieval of all the objects in the result; in such case, it exposes a chunk size (number of tuples in the chunk). Search Computing is focused on the efficient data-intensive computation and therefore most service interfaces are chunked.” [8]

- **Cache Descriptors**: It manages repeated invocations of the service. Caching the result at the requester side and then using it is a very efficient way to speed up but it is not acceptable with many services, for example systems give real-time answers. “Hence, parameters indicate if a service interface is cacheable and in such case what is the cache decay, i.e. the elapsed time between two calls at the source that make the use of stored answers tolerable.” [8]

- **Cost Descriptors**: A cost characterization is associated with each service call. It can be expressed as the response time which is the total duration from the request to response and/or monetary cost which is making a specific query. [8]

Here are some examples that have been used for the Search Computing Project. Every access pattern may include several service interfaces. For example, in “movie” Service Mart there is an access pattern which can filter movies by time and genre, and then it extracts them by their quality score. IMDB archive (http://www.imdb.com), which keeps information about movies and their scores voted by users, is used for this purpose. An ad-hoc wrapper makes periodic downloads in order to keep the system up-to-date. Another example is for “cinema” Service Mart in which “Movie Showtimes - Google Search” (http://www.google.com/movies) is used to get information about the cinemas located close to an input location. The result contains information about cinemas sorted by distance from the input location, but it does not give the actual distance. Last example is “restaurant” Service Mart which uses Yahoo Local source (http://local.yahoo.com/) in order to find Businesses & Commercial Services (e.g. restaurants) which are close to an address. [8]

### 3.2.3 Web Service Registration and Adaptation

Publishing data sources on the Web is a trend that supports users. There are tools and environments which are built by Google, Yahoo and Microsoft in order to help the users to publish their data. In Search Computing, ranked output should be produced by data sources and chunks should perform data extractions so that a search can be suspended and resumed by users. Tools are designed in Search Computing in order to build search service adapters. There are three different scenarios: [8]
• A Web service can be used to query data or results from different Web Services can be combined.
• Wrappers must be used in order to extract the data which is available on the Web.
• Data cannot be accessible, thus it must be materialized first.

Results are in an interchange format written in JSON which is not only easy to read and write by humans but also easy to parse and generate by machines. All instances of a Service Mart use the same interchange format, without regard to the service interface which creates them. Here is a JSON of “movie” instance: [8]

```json
{"title": "Highlander",
"director": "Russell Mulcahy",
"score": "0.7",
"year": "1986",
"genres": [{"genre": "action"}],
"openings": [{"country": "US", "date": "31-10-1986"}],
"actors": [{"name": "Christopher Lambert"},
{"name": "Sean Connery"}]
```

### 3.2.3.1 Web Services

A web service which is the typical service implementation returns the output in arbitrary format that may be HTML, XML and JSON. It is mentioned before that interchange format of Service Mart is a well-defined JSON structure. In order to combine the results of different Web services, Service Mart Framework has been built which includes some software modules to manipulate data. One of them is the “invocation module” which is used to invoke a service and to return a list of tuples. After that, “tuple reader” reads the tuples coming from invocation module and they are possibly copied by “tuple cloner”. Projections, string replacements, data conversions and splitting or concatenation of attributes are performed by other modules. Since a search service could return too many results, “Chunker module” may be needed to change the chunk size. [8]

### 3.2.3.2 Web Pages

HTML pages are wanted to use as second type of sources and good quality information is stored in HTML Web sites. “In the context of Service Marts, wrappers can be used to capture data which is published by Web servers in HTML format, because in such case a data conversion is needed in order to support data source integration – data must be rearranged according to the Service Mart normalized schema. Another typical use of wrappers in Search
Computing occurs when services respond with HTML documents which must be translated in the normal schema and encoded in JSON.” Lixto is used in order to build wrapper. By marking a region a set of rules are built by user. By this way, a wrapper which queries Web site in real time has been generated. In Figure 7, the relation between data an extracted on the Web can be seen. [8]

![Figure 7: Data extraction from query results](image)

### 3.2.3.3 Materialized Databases

In some cases data should be summarized and materialized to be stored at the engine site. In order to transform the format, to eliminate redundancy and to improve the quality data materialization can be applied to sources. By this way, data preparation is moved from query execution to source registration time. A materializer, whose scope is to read data sources and organize data in a normalized format, has been developed for Search Computing. Data extraction layer works directly on data sources, which can be of any formats. The aim of this layer is to transform the data into relational tables of arbitrary format (called primary materialization) which are temporary, used only in the materializer and are not seen to outside. Some SQL procedures are applied to these processes to produce a normalized schema. [8]

A materializer uses the modules mentioned in 3.1.3.1 to merge results coming from different Web services and includes some new units working together with the previously mentioned
one. For instance, Tuple writer unit writes data into rows in a database table. An example of materialization process is shown in Figure 8. [8]

![Figure 8: Materialization process][8]

### 3.3 SeCo API

SeCo API is divided into two sub-groups which are Mart Repository and Query Processor.

#### 3.3.1 Mart Repository

Mart Repository is the group that the repositories are kept in the system. The API provides access to the functionalities of the server, containing to the registry of registered Service Marts, Access Patterns and Service Interfaces. HTTP is the protocol to communicate to the API and JSON is the data serialization format. [10]

#### 3.3.1.1 Operations on Resource Collections

The API introduces several collections of resources, e.g. service marts, access patterns, service interfaces, and so on. Each resource collection has its own URL (e.g. /mart/registry/marts) and supports the basic operations showed below: [10]

- **GET**: Lists all the resource in the collection. Resources are generally sorted where the sorting is specific to the collection.
3.3.2 Operations on Resources

A resource is reached by concatenating the collection URL and the ID of the resource, for example XYZ mart is reached by the URL /mart/registry/marts/XYZ. These operations are supported according to the resource: [10]

- GET: Returns a resource representation.
  - 404 not found – case: the URL does not address a resource.

- PUT: Registers or updates a resource representation.
  - Body: A new representation of the resource to modify or register
  - Result: 201 created + assigned ID – case: a new resource is created.
  - 204 no content – case: an existing resource has been updated.
  - 400 bad request – case: the resource representation is not valid.

- DELETE: Deletes the resource
- Result: 201 no content – case: on success.
404 not found – case: no resource registered for the ID.

### 3.3.1.3 URL Organization

Here is the list of URL organization of Mart Repository in the API:

<table>
<thead>
<tr>
<th>A: /mart/registry</th>
<th>-</th>
<th>Mart repository root URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>B: A/marts</td>
<td>GET/POST/DELETE</td>
<td>Collection of registered Service Marts (SM)</td>
</tr>
<tr>
<td>B/{martId}</td>
<td>GET/PUT/DELETE</td>
<td>Specific registered SM</td>
</tr>
<tr>
<td>C: A/access-patterns</td>
<td>GET/POST/DELETE</td>
<td>Collection of registered Access Patterns (AP)</td>
</tr>
<tr>
<td>C/{patternId}</td>
<td>GET/PUT/DELETE</td>
<td>Specific registered AP</td>
</tr>
<tr>
<td>D: A/interfaces</td>
<td>GET/POST/DELETE</td>
<td>Collection of registered Service Interfaces (SI)</td>
</tr>
<tr>
<td>D/{interfaceId}</td>
<td>GET/PUT/DELETE</td>
<td>Specific registered SI</td>
</tr>
<tr>
<td>E: A/mart-connections</td>
<td>GET/POST/DELETE</td>
<td>Collection of registered Mart Connections (MC)</td>
</tr>
<tr>
<td>E/{martConnId}</td>
<td>GET/PUT/DELETE</td>
<td>Specific registered MC</td>
</tr>
<tr>
<td>F: A/connection-patterns</td>
<td>GET/POST/DELETE</td>
<td>Collection of registered Connection Patterns (CP)</td>
</tr>
<tr>
<td>F/{connPatternId}</td>
<td>GET/PUT/DELETE</td>
<td>Specific registered CP</td>
</tr>
<tr>
<td>G: /engine</td>
<td>-</td>
<td>Engine interface root URL</td>
</tr>
<tr>
<td>H: G/sources</td>
<td>GET/POST/DELETE</td>
<td>Collection of registered, system-level sources</td>
</tr>
<tr>
<td>I: H/{sourceId}</td>
<td>GET/PUT/DELETE</td>
<td>Specific registered source</td>
</tr>
<tr>
<td>F/{accessMethodId}</td>
<td>GET/PUT/DELETE</td>
<td>Specific access method inside a source</td>
</tr>
</tbody>
</table>

Table 1: URL Organization of Mart Repository [10]

### 3.3.2 Query Repository

Query Processor server manages the queries and the API is used to reach it as in the Mart Repository.
3.3.2.1 Operations on Resource Collections

Sources, sessions, invocations and so on can be reached via API and each has its own URL in which the operations can be supported: [9]

- **GET:** Lists all the resources in the collection. Resources are generally sorted where the sorting is specific to the collection.
  - Parameters: 
    - `_start`: integer [0..1] – indicates the index of the resource to return.
    - `_count`: integer [0..1] – indicates the upper limit of resources to return.
    - `<propertyXXX>`: string [0..1] - requires only resources having propertyXXX equal to the value specified to be returned; multiple constraints are AND.
  - Result: 
    - 200 OK + a list of resource descriptors – case: on success.
    - 400 bad request – case: unknown resource property.

- **DELETE:** Deletes the sources in the collection
  - Parameters: 
    - string [0..1] – requires only resources having propertyXXX equal to the value specified to be deleted; multiple constraints are AND; all registered sources are deleted if no constraint is specified.
  - Result: 
    - 200 + the number of deleted resources – case: on success.
    - 400 bad request – case: unknown resource property.

- **POST:** Registers a new resource with/without its ID. The post body contains the resource descriptor. If the ID is given, it will be registered on this ID, otherwise, the system gives the ID. If the ID is already in the system, the operation will update this resource.
  - Body: 
    - the descriptor of the resource to register.
  - Result: 
    - 201 created + assigned ID – case: a new resource is registered.
    - 204 no content – case: resource with the same ID was updated.
    - 400 bad request – case: resource descriptor is not valid.
3.3.2.2 Operations on Resources
A resource is reached by concatenating the collection URL and the ID of the resource, for example XYZ source is reached by the URL /sources/XYZ. These operations are supported according to the resource: [9]

- **GET:** Returns a resource representation.
  - **Result:**
    - 200 OK + resource representation – case: on success.
    - 404 not found – case: the URL does not address a resource.

- **PUT:** Registers or updates a resource representation.
  - **Body:** A new representation of the resource to modify or register
  - **Result:**
    - 201 created + assigned ID – case: a new resource is created.
    - 204 no content – case: an existing resource has been updated.
    - 400 bad request – case: the resource representation is not valid.

- **DELETE:** Deletes the resource
  - **Result:**
    - 201 no content – case: on success.
    - 404 not found – case: no resource registered for the ID.

3.3.2.3 URL Organization
Here is the list of URL organization of Query Repository in the API:

<table>
<thead>
<tr>
<th>A</th>
<th>/sources</th>
<th>GET/POST/DELETE</th>
<th>Collection of registered, system-level sources</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A/{sourceId}</td>
<td>GET/PUT/DELETE</td>
<td>Specific registered source</td>
</tr>
<tr>
<td>B</td>
<td>/engine</td>
<td>-</td>
<td>Engine root URL</td>
</tr>
<tr>
<td></td>
<td>B/options</td>
<td>GET/PUT</td>
<td>Execution options</td>
</tr>
<tr>
<td></td>
<td>B/invoke</td>
<td>GET</td>
<td>Invocation of an access method</td>
</tr>
<tr>
<td>C</td>
<td>B/sessions</td>
<td>GET/POST/DELETE</td>
<td>Collection of execution sessions</td>
</tr>
<tr>
<td>D</td>
<td>C/{sessionId}</td>
<td>GET/DELETE</td>
<td>Session management</td>
</tr>
<tr>
<td></td>
<td>D/options</td>
<td>GET/PUT</td>
<td>Session options</td>
</tr>
<tr>
<td></td>
<td>D/source</td>
<td>PUT</td>
<td>Updates timeout of the session</td>
</tr>
<tr>
<td></td>
<td>D/sources</td>
<td>GET/POST/DELETE</td>
<td>Global view of session sources (system + session specific sources)</td>
</tr>
<tr>
<td></td>
<td>D/transient-sources</td>
<td>GET/POST/DELETE</td>
<td>Collection of session specific sources</td>
</tr>
</tbody>
</table>
### Table 2: URL Organization of Query Repository [9]

<table>
<thead>
<tr>
<th>D/persistent-sources</th>
<th>GET/POST/DELETE</th>
<th>Collection of system-level sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>D/persist</td>
<td>POST</td>
<td>Persists a session specific source in the persistent repository</td>
</tr>
<tr>
<td>E</td>
<td>D/executions</td>
<td>GET/POST/DELETE</td>
</tr>
<tr>
<td>F</td>
<td>E/{executionId}</td>
<td>GET/DELETE</td>
</tr>
<tr>
<td>F/options</td>
<td>GET</td>
<td>Execution options (read-only)</td>
</tr>
<tr>
<td>F/touch</td>
<td>PUT</td>
<td>Updates timeout of the execution</td>
</tr>
<tr>
<td>F/skip</td>
<td>POST</td>
<td>Skip execution task</td>
</tr>
<tr>
<td>F/more</td>
<td>POST</td>
<td>More execution task</td>
</tr>
<tr>
<td>F/results</td>
<td>GET/POST</td>
<td>Result management</td>
</tr>
</tbody>
</table>

#### 3.3.2.4 Data Source Registry [9]

A registry of data sources are kept in Query Processor in which each keeps access methods to be used to query to source data.

- **✓ URL: /sources**
  - GET: Lists the sources in the registry
  - DELETE: Deletes the sources in the registry
  - POST: Registers a new source or update

Each source is located in:

- **✓ URL: /sources/{sourceId}**
  - GET: Returns a representation of the source
  - PUT: Updates a source or registers a new one
  - DELETE: Deletes a source
3.3.2.5 Execution Engine [9]

The execution engine is located:

✓ URL: /engine

The engine interface has three main components, session, source and execution:

- Sessions involves and identifies the interaction between a client and the engine. All the interactions have to occur in a session. While the sessions expire if not used for a given amount of time, they also may be opened and closed manually.
- Sources can be accessed and managed within a session.
- An execution lets extracting data from the source within a session and related to a source.

3.3.2.5.1 Engine Options

✓ URL: /engine/options

Figure 9: Hierarchy of objects in the data source registry [9]
3.3.2.5.2 Sessions

Active sessions are available in:

- **URL:** /engine/sessions

  - GET: Active sessions are listed according to creation date

  ```
  
  
  
  
  
  
  
  
  
  
  
  ```

  - DELETE: Closes and deletes the active ones

  - POST: Opens and registers a new session. (no-data is needed to post)

Sessions can be accessed in:

- **URL:** /engine/sessions/{sessionId}

  - GET: Returns the session

  ```
  
  
  
  
  ```

  - DELETE: Closes and deletes the session
3.3.2.5.3 Session Options

✓ URL: /engine/sessions/{sessionId}/options
  - GET: Returns the session options
  - PUT: Sets the session options

3.3.2.5.4 Session Timeout Update (touch)

Timeout of session can be updated in:

✓ URL: /engine/sessions/{sessionId}/touch
  - PUT: Updates the session (no-data is needed)

3.3.2.5.5 Session Sources

✓ URL: /engine/sessions/{sessionId}/sources
  - GET: Shows the sources in the registry
  - DELETE: Deletes the sources in the registry
  - POST: Registers a new source or updates

3.3.2.5.6 Executions

The list of execution tasks belongs to a session can be seen in:

✓ URL: /engine/sessions/{sessionId}/executions
  - GET: Active execution tasks are listed and sorted by creation date

```json
{
  "type": "List",
  "size": 1,
  "items": [
    {
      "type": "ExecutionBean",
      "accessMethodId": "random",
      "creationTimestamp": "1288727124196",
      "executionId": "BAAAAA",
      "inputTuple": {
        "tupleId": "yHWnmW4C7nIX10NSFivIkfhYhmAB",
        "tupleScore": 0.0,
        "address": "Time Square",
        "city": "New York",
        "country": "USA",
        "genre": "Drama",
        "year": 2009
      },
      "lastAccessTimestamp": "1288727124196",
      "sessionId": "PAAAAA",
      "sourceId": "qep_movies",
      "status": "idle"
    }
  ]
}
```
DELETE: Closes and deletes the active execution tasks

POST: Creates a new task (an example of post data is shown below)

```json
{
    "sourceId" : "qep_movies",
    "accessMethodId" : "random",
    "inputTuple" : {
        "address" : "Time Square",
        "city" : "New York",
        "country" : "USA",
        "genre" : "Drama",
        "year" : "2009"
    }
}
```

A single execution task can be reached in:

- URL: `/engine/sessions/{sessionId}/executions/{executionId}`
  - GET: Returns an execution task

```json
{
    "sourceId" : "qep_movies",
    "accessMethodId" : "random",
    "inputTupleId" : "dfu342Hd8Hdsa8hasf83",
    "inputExecutionId" : "CAAAAA"
}
```

- DELETE: Closes and deletes the task

### 3.3.2.5.7 Execution Commands

“more” task gives opportunity to ask for more data from services.

- URL: `/engine/sessions/{sessionId}/executions/{executionId}/more`
  - POST: Submits a more task
    - “maxTuples”, “maxFetches”, “maxTime” and “allowedAtoms” are the keywords.

### 3.3.2.5.8 Execution Results

Results of a task can be reached via:

- URL: `/engine/sessions/{sessionId}/executions/{executionId}/results`
  - GET: Returns the results
POST: Returns the result but it touches to result like filtering, sorting, etc. “filter”, “start”, “count”, “criteria” and “project” are the keywords which can be left null.

3.3.2.5.9 Execution Options
It is read-only since it is not possible to update a task while it is running.

✓ URL: /engine/sessions/{sessionId}/executions/{executionId}/options
  o GET: Shows the execution options

3.3.2.5.10 Execution Timeout Update (touch)
✓ URL: /engine/sessions/{sessionId}/executions/{executionId}/touch
  o PUT: Updates the execution (no-data is needed)

3.3.2.5.11 Access Method Invocation
✓ URL: /engine/invoke
  o GET: invokes an access method. Parameters can be given are:
    “_sourceId”(mandatory), “accessMethodId”(mandatory), “_startTuple”, “_startFetch”, “_maxTuples”, “_maxFetches”, and “_maxTime”.

3.3.3 metadata.JSON
This file includes additional information about the services and has been updated with a field which indicates the main output of the services. For each service, main output, which was missing in the original file, has been selected one by one in order to ease the understanding of the user and was added during the project. It is thought that the user wants to see the most important variable of the results at a glance. So, these outputs are the first things that the user takes the search results. The ID of the main output has been kept in “mainOutput” variable.

This file is also used for taking the name of the inputs of services. “caption” variable is used for this aim.

```
"serviceID": "imdb_movies_by_title",
"serviceName": "IMDB Movies By Title",
"serviceShortName": "IMDB Movies By Titles",
"icon": "movies.png",
"description": "Movies as extracted by IMDB",
"mainOutput": "title",
"inputSignature": [
  {
    "semanticType": "String",
    "default": "",
    "description": "Movie Title",
  }
]`
```
3.3.4 Connection in Mart Repository

In the Mart Repository, the connections between the elements are done by IDs. All the components have a unique ID to be connected to other level of the repository. Service Marts can be listed with the related URL given in the previous sections and can be seen that all the Service Marts have a unique ID for themselves. The type of the entries is given by “__type__” attribute and IDs are kept in “__id__” attribute. Service Marts have “description” attribute, “name” attribute and a list of attributes called “signature” besides “__type__” and “__id__”.

In order to connect Access Patterns to the Service Marts, all the Access Patterns have an attribute called “serviceMartId”. Since a Service Mart can have more than one Access Pattern, a Service Mart ID can be used in the several Access Patterns. Besides Service Mart ID included into Access Patterns, all of them have a unique ID for themselves like Service Marts do. Also, they have a list of attributes called “inputAttributes” which indicates input parameters, “name” attribute, “__type__” attribute and a list of attributes called “outputSignature” that shows output parameters.

For the connection between Access Pattern and Service Interface levels, all the Interfaces have an attribute called “accessPatternId” which keeps the ID of the Access Pattern connected to the interface. As an Access Pattern can have more than one Interface, an Access Pattern ID can be used in many Interfaces. Like Service Marts and Access Patterns, they have a unique ID for themselves and a “__type__” attribute. Moreover, they have a list of attributes called "inputAttributes" to indicate the input parameters and a list of attributes called "outputSignature" to show output parameters.
For example, let’s choose a Service Mart to show the connection with the other elements. The Service Mart "Real Estate" with the "src5m3AB6f1AtlqHZJCR9QrhG" ID has been selected. When we move to level Access Pattern, we can find 4 Access Patterns which keep this Service Mart ID in their body. This means that “Real Estate” Service Mart includes 4 Access Patterns connected to itself. Let’s choose one of them to go further to Interface level. The Access Pattern "Real Estate by Coords" with the ID “JBnYiRJ7joHMjc5MCLx8QK” has been chosen which has description, input and output attributes beside the Service Mart ID. In the Service Interface level, the Service Interface which has this Access Pattern ID is only one. It is "Zillow By Coord" with the ID "amoUyAm2T6CTHSG8Bq0J59J". This is an example flow from the Service Mart level to the Service Interface level through Access Pattern level.

Other element of the Mart Repository is connection-patterns which indicate the connection between the Access Patterns. In order to make multi-domain queries, Access Patterns are connected to each other via this element. Connected pairs are kept in here and used in case of necessity. The relations are directional which means that “an access pattern A is connected to another one B” does not mean “B is connected to A” unless there is another connection from B to A. We can look for the Access Pattern which we chose for the example above. "Real Estate by Coords" with ID “JBnYiRJ7joHMjc5MCLx8QK” is used 21 times in this element while some of them are source and some of them are target. In the body taken from the Mart
Repository shows the connection between real estate and open job position by coordinates. It has two Access Pattern IDs for both source and target. Also, there are two constraints for the attributes which should be connected in both access patterns. Moreover, there are “__id__”, “__type__”, “description” and “name” attributes as well.

```json
{
  "__type__": "ConnectionPattern",
  "__id__": "RealEstateNearJob",
  "constraints": [
    {
      "operator": "equal",
      "sourceAttributeId": "attr_5",
      "targetAttributeId": "attr_8"
    },
    {
      "operator": "equal",
      "sourceAttributeId": "attr_6",
      "targetAttributeId": "attr_7"
    }
  ],
  "description": "Connection between real estate listing and an open job position by coordinates",
  "name": "Real Estate near Job",
  "sourceAccessPatternId": "05mhUJkHDLKyd8MvA9f",
  "targetAccessPatternId": "jBnYiRJ7j0HMjc5MCLx8QK"
}
```

### 3.3.5 Connection in Query Repository

In order to make a query in the Query Repository, first a session must be created. It is done by the URL given in 3.2.2.3 by sending an empty data set. After a session is created, a session ID is returned by the system in order to go further in the query operation. This session ID is necessary because query to the system is not possible without a valid session ID. After a certain amount of time, sessions are killed automatically by the system unless it is not idle. When the session ID is taken, querying can be done with execution process. With specific session ID and JSON data which has the inputs, the query is executed. In this data, Service Mart ID is sent as source ID and Service Interface ID is sent as Access Method ID since the system expects these parameters. After the query is executed, the system gives an execution ID to reach the results. With this execution ID, the query results can be taken via “more” operation with the fetching options as JSON data.
Figure 11: Flowchart of the Query Execution
4 Analysis and Design

4.1 Overview of the Project

The aim of the project is by providing the mobile version of Search Computing to improve the usage of it. The user wants to benefit from this because they can make their plans via Search Computing. However, if there is a mobile version that they can use easily with main properties, it will encourage them since they don’t need to spend their time for managing a full web site. By mobility feature, it is not needed to be in front of the computer for making plans via Search Computing.

The users can start choosing the topic in which they want to search. For example, these can be Real Estate, Car Rental, Restaurant, Hotel, Information about movie and etc. After choosing the topic, they can choose the way they can search for, for instance after choosing the Restaurant topic; they can simply search for restaurants by Address or by its coordinates. After this step, they can choose the service provider, in other words, interface. In this example, after choosing Restaurants by coordinates, it can be searched via Yelp or YQL search provider. When these steps are completed, the user is wanted to complete the form which is retrieved according to the search provider to send the query. After the submission of the form, there are three types of visualization possibility: accordion, map view, and comparison.

The accordion view is the default visualization tool in this mobile application. The aim of this view is to show the main attribute of each result in the small screen of mobile phone. Details of each result can be seen by clicking to the related row and can be selected by clicking to “Choose” button.

Other tool for visualization is Map view which can be used if the results include location information. This visualization method is very useful because sometimes the address information does not make sense but if it is seen on a map, it can be easily understood. That’s why, if the searched thing has location information like restaurant, hotel, etc., it is good to see it on the map in addition to your own location which is also shown on the map even though the mobile phone does not have a GPS module. By this way, the user can notice the distance with respect to his/her own location. The pin can be clicked to see the details of the result and from this detail page it can be selected.
Last visualization tool for the results is comparison view in which the user can choose the attribute to use for comparison in addition to main attribute. The aim of this tool is to give chance to the user for comparing the results in accordance with one attribute. So, he/she can easily see the difference between each result with respect to the attribute selected. After the comparison, he/she can choose one of the items or can pass to the other visualization tools as it can be done in the others.

After the selection of one of the results, the history page is automatically seen with selected item added. In history page, all the items which are selected can be seen with accordion feature. Moreover, map view is also valid in this page in order to let the user see the locations of the selected items and its own location. The user can delete any of the items or can add another type of search linked to the current ones. That is, some of the topics are linked and the user can jump to the linked topics which are available to choose. For example, after choosing a restaurant the user can jump to choose a hotel which is connected to the restaurant search. By this way, various combinations can be done by user using Search Computing platform via mobile.

UML Use Case, Component and Activity diagram are shown below:
Figure 12: Use Case Diagram

Figure 13: Component Diagram
Figure 14: Activity Diagram
4.2 Design of Mobile Interface

Interface of the web application is designed following the design principles given in the section 2.4. In this section user interface elements that are implemented in the application are introduced along with explanation of the design decisions about them and the motives of choosing them.

4.2.1 Tab Set

A tabbed document interface is useful for keeping multiple documents in a single window. It is a metaphor for real world card tabs inside paper files.

Tabbed document interface was chosen for the project as it would allow user to check his/her history anytime without losing the place where he/she left searching. Then, the user will be able to switch tab again to continue searching.

An important design decision about the tab set is the number of the tabs to insert into it. Instead of adding tabs for search steps, map views and other screens they are grouped into two tabs, namely Search and History. All the screens and operations that are related with
searching are displayed inside the Search tab while the ones that are related with keeping previously chosen search results are displayed inside the History tab. Only screen that can be displayed in both tabs depending on where it is called is the Show Map screen. Because, it is possible to display both search results and history elements in a map view. This approach of keeping a limited number of tabs provides benefits to the application in two ways:

1. It is guaranteed that the tab set will fit to user screen even in smaller mobile devices.
2. Complexity of the user interface is decreased while the ease of use is increased.

### 4.2.2 Item List

A vertical list of items is useful in mobile devices to display data which is a collection of relatively short strings (as they should fit in a row). Items may also contain links to other screens or pages. In that case, item is usually indicated with an arrow symbol.

![Item List Example](image)

**Figure 16: Two sample item lists from iPhone web applications**

Item lists are richly used in the web application as navigation lists as there are many circumstances where the user should choose a path among many alternatives i.e. while choosing a service mart for the search.
4.2.3 Accordion

Accordion is a vertical stack of elements with expandable and shrinkable bodies. Clicking or tapping to the title of an element expands its body. It is a metaphor for real life music instrument in which sections of the bellows can be expanded by pulling outward.

Figure 17: A sample accordion

Accordion view is used in the areas of the web application where a large amount of data for each item needs to be displayed. In the list of search results and the history, accordion is a much more understandable element than the group list. The most important information about the result is used as the section title while the rest of the information is embedded to the section body along with other controls.

In order to ensure simplicity, both accordions in the application are designed to be initialized with all sections closed. When the user expands a section by tapping the title, previously expanded section is shrunk to make sure that only one section is open at all times.
4.2.4  Google Map

Map visualization consists of a scrollable and zoomable map with markers to indicate locations on it. It is a useful method for visualizing data with geographical coordinates.

Map view of the search results contains Google Map in the web application. The map is filled with the markers for all 20 search results or indefinite number of results in history if they contain latitude and longitude information. In addition to those markers, current location of the user is also displayed on the map.

Since the number of markers on the map is relatively high different marker symbols are not used for the elements not to harm the understandability of the map. Markers are also designed to be tappable to provide extra information. In the history tab, extra information is given in a balloon without changing the screen, while in the search tab a new screen is called to display information and the choose button.

4.2.5  Table

Table is an arrangement of data in rows and columns. It is useful for comparing data when the information is not vast. Numerical overload is a problem in representing vast data in table form.
Table view in the web application lets the user to compare all the results based on one property. As a design choice user is allowed to add only one column to the table to avoid numeric overload and a table that cannot fit to the mobile device screen.

### 4.2.6 Form Elements

Form elements that are used to receive user input. Typical form elements are: text fields, checkboxes, radio buttons, text areas etc. Advanced input elements are data and time pickers, color picker, file selectors etc.

![Date picker and time picker for iPhone](image)

Form elements in the web application are dynamically chosen according to the data type of the input field. For example, an date input is represented by a date picker element while a regular string input is represented with an text field. Details of the form element decision process are explained in the sub-section 5.2.5.
4.2.7 Buttons

Buttons in the web application are positioned in a consisted way. Following design decisions are applied to all buttons inside the application:

- Back buttons are always on the left side of the header bar. No other button is allowed in this location.
- Buttons that change result views (Compare, Map, List) are always on the right side of the header bar in fixed locations.
- Choose buttons and delete are always near the detailed information of that result since the user would not choose or delete an item without reading its details.
- Reset button is always on the right side of the header bar.
5 Implementation of the Project

Implementation of the web application is explained in this chapter. Implementation aspects can be categorized into three sections. In the first section, details about the data connections between SeCo API and the web application are explained. Then, usage of the web application interface with the technical details of the interaction elements and backend programming specifications of the critical parts are given in the following two sections: User Interface and Application Logic, respectively.

5.1 Data Model

Mart Repository and Query Processor ReST API’s of the Search Computing project is already explained in the previous section. Here, methods and technologies that were employed in order to achieve an efficient and stable data flow are demonstrated.

Connecting to a remote API, sending/retrieving data and using in inside the user interface can be done completely in the client-side with the help of the current web technologies such as asynchronous JavaScript and XML (Ajax). However, currently Search Computing API’s are not open to the public and can be accessed by means of a secure Virtual Private Network (VPN) so there are two ways to accomplish data transfer between API’s and the web application:

- Develop and host the web application in the same server as the Mart Repository and Query Processor servers.
- Use VPN client to connect the network of the Mart Repository and Query Processor servers.

Second option was preferred in the analysis phase of the project for the sake of convenience of development. However, this choice resulted in two issues limiting the client-side data access:

- It is impractical to ask the users to install a VPN client to their mobile devices, so connection to the VPN should take place in the server side.
- It is not allowed by web browsers to make cross-domain requests (calls to any web server other than the one the web page is coming from) [13]. This restriction can be overcome by installing a web proxy on the server.
Those two issues made it necessary to employ some server-side programming for the data transfer.

The easiest and the most straight-forward approach to parse and use the data would be doing it in the server-side and inserting it directly to the output files. However, this approach would harm the generic usability of the web application. In order to keep it generic and isolate the data layer from the user interface layer, following method is proposed:

- User triggers the data connection.
- Client-side of the application makes an asynchronous call to the web service that resides in the same server.
- Web service in the server-side sets up the connection according to the input parameters sent from the client side.
- Web service receives the output data and echoes the received output data as it is (without manipulating or changing the format).
- Client side of the application receives the data, parses, manipulates and displays it.

This approach, although slightly more complex and takes longer time, makes testing, debugging and reusing easier.

- Testing and debugging is easier because separate layers can be tested and debugged individually.
- Reusing is easier because the same code with minor changes can be used when connecting to a different API or when the proxy and VPN are not necessary anymore and the client-side application can directly access data.

Technologies that are employed in this approach are grouped according to the location they perform their tasks and are explained below.

5.1.1 Server-Side Technologies

The web server is virtually inside the same network as the Mart Repository and Query Processor ReST API’s by means of a SSL VPN application called OpenVPN. OpenVPN establishes a secure connection to port 1194 of the hostname seco.como.polimi.it.

The web service that performs a proxy function is developed in PHP scripting language. Its first task is to read the parameters from the query string to get following information:

- Which ReST API and which URL to connect?
• What kind of operation to perform? (POST Request or GET Request)
• Whether to send any input data? If yes, what is the data? (In JSON format)

After answering those questions, the second task of the web service is to perform the operation and echo the output data. Sometimes output data is not in JSON format, in such cases, the third task is to encode it into JSON format for the sake of generic design.

Body of a sample function to perform a POST request and its usage are given below:

```php
function do_post_request($url, $data, $optional_headers = null)
{
    $params = array('http' => array('method' => 'POST', 'content' => $data));
    if ($optional_headers !== null) {
        $params['http']['header'] = $optional_headers;
    }
    $ctx = stream_context_create($params);
    $fp = @fopen($url, 'rb', false, $ctx);
    if (!$fp) {
        throw new Exception("Problem with $url, $php_errormsg");
    }
    $response = @stream_get_contents($fp);
    if ($response === false) {
        throw new Exception("Problem reading data from $url, $php_errormsg");
    }
    return $response;
}

$result = do_post_request($url, $data, 'Content-Type: application/json');
echo $result;
```

5.1.2 Client-Side Technologies
mobl is capable of accessing ReSTful web services. Since Mart Repository and Query Processor API’s conform to the ReST constraints, proxy web service that echoes the API outputs is a ReSTful web service.

In order to connect a web service using mobl, an interface should be defined as well as how it maps to URLs. A service can be defined using the “service” construct which contains a set of resources. A resource contains return type and optionally input arguments as well as attributes
such as URL to connect and encoding type in order to describe how a call to that resource should be mapped to a service call [11]. Following is a code snippet which demonstrates a sample service and resource, taken from “data.mobl” file that deals with data related aspects of the application, as the name suggests.

```java
service queryProcessor{
  resource createExecution(sessionId : String, JSONdata: String): JSON {
    uri = "proxy.php?url=http://testing1.seco:8084/engine/sessions/" + sessionId + "/executions&data=" + JSONdata + "&request=POST"
  }
  ...
}
```

“createExecution” resource has two input arguments with the type of string and its return type is JSON. In the body of the resource there is the uri attribute to map the call. Three query string parameters, namely url, data and request are necessary for the server side web service to perform correct operation. Their meanings are explained in the “Server-Side Technologies” sub-section.

After defining the service and the resource, it can be called as any other static method and it returns the data from the web service in the JSON format. A resource can be called directly to evaluate it synchronously or it can be called within “async” construct to evaluate it asynchronously [11]. Below it is given two example usages of the “createExecution” resource:

```java
var executionId = queryProcessor.createExecution(sessionId, JSONData)
var executionId = async(queryProcessor.createExecution(sessionId, JSONData))
```

When “async” construct is used, the variable that is assigned to the construct is initialized with the value of “null”. When the result of the expression inside the “async” construct is knows, that is the data has been successfully received from the web service, actual value is assigned to that variable. Program will continue to flow while the data is being received. If
there are certain operations that require that data, they should be inserted inside the “whenLoaded” control which waits until the expression passed as an argument is non-null and only then renders its body [11]. A sample code snipped demonstrating the usage of “whenLoaded” control is given below.

```javascript
whenLoaded(executionResults){
    // Any operation that require “executionResults”
}
```

### 5.2 User Interface

Technical details about the user interface elements of the mobl which are employed in the project are explained in this section. Screens are discussed in the order they would be displayed during a typical usage scenario.

For each screen, two descriptive screen captures are provided which are followed by textual explanations of the interactions and interface elements as well as programming aspects of them supported by code snippet examples.
5.2.1 Root Screen

Root is the screen that must be present in all mobl web applications. It is the screen that is first run and displayed to the user. In this project, root screen contains only a control that is called “tabset” which takes an array of tuples with three elements as its first argument. Those three elements are:

1. Title of the tab that is visible to the user
2. URL to an icon to be displayed near the title
3. The control to use as the body of the tab
Second argument of the “tabset” control is a string called “activeTab” which contains the title of the selected tab [11]. In the code snippet below it is assigned to a global variable to be able to manipulate it programmatically later on.

```
screen root() {
  tabSet([("Search", "", serviceMarts),("History", "", history)],
  activeTab=currentTab)
}
```

Above control is rendered into a tab set containing two tabs titled “Search” and “History” as it can be seen from Figure 21. Bodies of those tabs are bound to the controls called “serviceMarts” and “history” respectively.

### 5.2.2 Service Mart Screen

Search tab is the initially active tab of the tab set. Its body is composed of “serviceMarts” control which renders a list of item controls inside a group control. Each item contains the name of a service mart as it can be seen in Figure 21-a. In this initial screen, user chooses the mart he/she wants to begin searching.

```
group{
  list{mart in martResults.items}{
    item(onclick={accessPatterns(mart.__id__, mart.description);}){
      label(mart.description)
    }
  }
}
```

Above source code illustrates most of the main features of group control, item control, label control and list construct. “martResults” variable is in the JSON type and received from the web service through an asynchronous call that is explained in section 5.1.2.

- Group control is a container for item controls. Item controls can only exist inside a group control.
- List construct is used for iterative task and its usage is very similar to “foreach” constructs in many programming languages. Here, there is iteration over the array called “items” inside the JSON data.
- Item control contains an “onclick” event trigger which calls the “accessPatterns” screen and redirects the page to it. The screen also takes the id and the description of the selected mart as input arguments. Those values are taken from the objects of the “items” array inside the JSON data.
- Label control prints the description of the mart inside the item body. [11]

5.2.3 Access Pattern Screen
After choosing the service mart, a similar list is provided to the user which is demonstrated in Figure 22-a. This time, group is filled with descriptions of access patterns belonging to the previously selected service mart. Since there is no direct way of getting the information about
the access patterns that belong to a specific service mart in Mart Repository ReST API, an array of all access patterns are received and the related ones are sorted out programatically.

```javascript
group{
    list(ap in apResults.items){
        when(ap.serviceMartId==martId){
            item(onclick={interfaces(martId, martDesc, ap.__id__);}){
                label(ap.description)
            }
        }
    } 
}
```

Above source code is very similar to the one in service marts. Again, there is iteration over the array from the JSON data received from the web service. One difference is the when construct which is a conditional statement testing whether the “serviceMartId” attribute of the access pattern is equal to the id of the service mart chosen by the user [11]. Clicking an item calls the interfaces screen and redirects the page to it with some input arguments stating the previous choices of the user.

5.2.4 Interfaces Screen
The user should make one last decision before accessing the search form: Interface. Interfaces screen displays a list of all interfaces belonging to the access pattern the user have chosen. Both the design of the screen (as it can be seen from Figure 22-b) and the source code of the screen (as it can be seen below) are very similar to the previous two screens.

Again, it is not possible to access information about the interfaces only if they belong to the specified access pattern from the API, so they are sorted out using the when construct evaluating the identifiers.

Clicking an item in the interfaces screen calls the form screen with the input arguments containing the identifiers of all three choices the user have made, namely: service mart, access pattern and interface.

```javascript
group{
    list(int in intResults.items){
        when(int.accessPatternId==apId){
            item(onclick={formScreen(martId, martDesc, apId, int.__id__);}){
```

57
5.2.5 Form Screen

Form screen is where the user enters queries to be sent to web service in order to filter the results according to his/her needs. As it can be seen from Figure 23, number of input fields, their types and names are not definite, but they depend on previous user choices of service mart, access pattern and interface.
mobl provide a large number of controls for different form elements. A subset of them is used in this project and their specifications are given below:

1. textField control renders a text field with some advanced features such as auto-correct, auto-complete and validation. Some of its input parameters that are used in the project and an example source code from the project are given below.
   - s: a string variable to bind the value of the control to
   - label: a string for label text of the text field
   - validator: a function that takes a string (s) as argument and returns an error message or an empty string to represent success

   ```
   item { textField(field._2, label = field._4, validator = validateDecimal) }
   ```

   ```
   function validateDecimal(s : String) : String {
     return /^-?[0-9]*\.[0-9]+$/.test(s) || s.length == 0 ? "" : "Only decimals";
   }
   ```

2. datePicker control renders a date picker on the screen. Its input parameters and an example source code from the project are given below.
   - d: a variable in the type of “DateTime” to bind the value of the control to
   - onChange: event triggered when the value of the this control changes

   ```
   item { label(field._4) "": " datepicker(field._2) }
   ```

3. timePicker control renders a time picker on the screen. It is very similar to the datePicker and has the same input parameters.

4. checkbox control renders a check box on the screen. Some of its input parameters that are used in the project and an example source code from the project are given below.
   - b: a boolean variable to bind the value of the control to
   - label: a string for label text of the check box
5. button control renders a button on the screen. Some of its input parameters that are used in the project and an example source code from the project are given below.
   - text: a string to display on the button
   - onClick: event triggered when the value of the button is tapped [11]

\[
\text{button("Search",onclick={results(martId, martDesc, apId, intId);})}
\]

Input attributes that are received from the web service interpreted differently and assigned to different input elements depending on their physical and semantic data type. The table below summarizes which data types are represented with which input element under which special treatment.

<table>
<thead>
<tr>
<th>Physical Type</th>
<th>Semantic Type</th>
<th>Input Element</th>
<th>Special Treatment</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>String</td>
<td>Text field</td>
<td>none</td>
</tr>
<tr>
<td>String</td>
<td>URL</td>
<td>Text field</td>
<td>Validation for URLs</td>
</tr>
<tr>
<td>String</td>
<td>ZIP</td>
<td>Text field</td>
<td>Validation for ZIP Codes</td>
</tr>
<tr>
<td>Decimal</td>
<td>Float</td>
<td>Text field</td>
<td>Validation for decimals</td>
</tr>
<tr>
<td>Decimal</td>
<td>CoordLong</td>
<td>Text field</td>
<td>Initialized with user longitude</td>
</tr>
<tr>
<td>Decimal</td>
<td>CoordLat</td>
<td>Text field</td>
<td>Initialized with user latitude</td>
</tr>
<tr>
<td>Integer</td>
<td>Integer</td>
<td>Text field</td>
<td>Validation for integers</td>
</tr>
<tr>
<td>DateTime</td>
<td>Date</td>
<td>Date Picker</td>
<td>Initialized with current date</td>
</tr>
<tr>
<td>String</td>
<td>Time</td>
<td>Time Picker</td>
<td>Initialized with current time</td>
</tr>
<tr>
<td>Boolean</td>
<td>Boolean</td>
<td>Check Box</td>
<td>none</td>
</tr>
</tbody>
</table>

Table 3: Mapping Between Data Types and Input Elements

Finally, a button control on the bottom of the page is used for submitting the query and redirecting to the results screen.
5.2.6 Results Screen

Results screen displays the 20 highest ranked results of the user query. Results are presented in a graphical user interface element called accordion. In an accordion titles of the items are stacked vertically and their content can be revealed by tapping the titles which results in expanded body. In the accordions used for this project, it is decided that all content should come shrunk initially and after expanding one item, only one item should remain expanding.
That results in shrinkage of the active item body when another item is expanded. Accordion element can be seen in the Figure 24.

In mobl, accordion is natively supported and can be created using the accordion control which takes two input arguments: “sections” and “activeSection”. First argument is an array of tuples with two elements each. Those two elements are:

1. A string to display user as the title of the section
2. A control to render as the body of the section.

Second argument of the accordion control is a title to represent currently selected tab [11]. The sample code below demonstrates the usage of an accordion control. In the project, elements of the accordion are inserted into an array instead of hardcoding them as in the sample code.

```javascript
Accordion([("Section 1", section1), ("Section 2", section2),
          ("Section 3", section3)], activeSection="Section 2")
```

Titles and body controls that are inserted into the array are created dynamically according to the data received from the web service. A section title contains the main output and the score of the result. Main output is the most significant output of the query result and it identifies that result. For example, title of a news article is its main output, as well as, name of a restaurant or address of a real estate.

A body control for an accordion item contains the attribute name – value pairs of the result printed by label controls in addition to a button to choose that result. The button chooses that result, changes the active tab to history and refreshes history tab. Source code for that control is given in the code snippet below.

```javascript
List(property in resultTupleProperties){
    when(property._3 == tupleNumber & & ...){
        label(property._1 + ": ",style=boldLabelStyle)
        label(property._2)
        nl()
    }
}
button("Choose",onclick={chooseResult(...); currentTab="History"; 
                history1();})
```
In addition to the accordion of the results, results screen also contains buttons to navigate to two other presentation screens: comparison and map view. Those buttons are embedded to the right hand side of the page header.

```
Header("\""){
  backButton()
  button("Compare", onclick={compareSelection(...)})
  button("Map", onclick={showMap(resultMarkers);})
}
```

![Figure 25: a) Compare selection screen b) Compare screen](image-url)
5.2.7 Compare Selection Screen

This screen is displayed when the user taps the compare button on the search screen. As it can be seen in Figure 25-a, screen includes a group of items containing the names of the output attributes of the search result. The user can tap on any output attribute to access the compare screen to obtain a tabular view of all results and their values for the selected output attribute. Below, it is given a code snippet from the comparison selection screen source code.

```java
Group{
    list(property in resultTupleProperties){
        when(...){
            item(onclick={ compare(...); }) { label(property._1) }
        }
    }
}
```

5.2.8 Compare Screen

Compare contains a table with 20 rows and three columns. Each row represents a search result, while the first column is always main output of each result. Second column, on the other hand, is not constant and its values depend on the selection made in the previous section. The last and the third column is for buttons to choose the result on that particular row. There is also a button in the header of the page to access map view directly from the comparison view. Compare screen is shown in Figure 25-b.

Source code given below demonstrates the table, row and column constructs of the mobl [11]. First two columns are filled with labels containing the output attribute values while the third one contains the choose button performing exactly same as the one in the accordion view. The reason of the when conditional expression in the third column is to exclude first row (header row) from having a choose button.

```java
Table {
    list(element in comparedProperties){
        row {
            col { label(element._1) }
            col { label(element._2) }
            col { when(...) {
                button("Choose",onclick={chooseResult(...); currentTab="History"; history1();})
            }
        }
    }
}
```
Show map screen displays the results coming from the web service on a Google Map if they contain latitude and longitude information. Location of the user and the location of the results are indicated with markers. When the user taps on a result marker, marker details page is called and the page is redirected to it.

**Figure 26**: a) Show Map Screen b) Marker Details Screen

### 5.2.9 Show Map Screen

Show map screen displays the results coming from the web service on a Google Map if they contain latitude and longitude information. Location of the user and the location of the results are indicated with markers. When the user taps on a result marker, marker details page is called and the page is redirected to it.
As it can be seen from the code line above, mobl can natively render a Google Map. Arguments of the `googleMap` construct are the following:

- **coords**: The coordinates of the center of the map. Type of this attribute is `Coordinates` which contains three numeric instance properties: latitude, longitude and the accuracy of the location.
- **markers**: A collection containing `MapMarker` objects to be rendered on the map. `MapMarker` type has four instance properties:
  - `coords`: Coordinates for the marker, of type `Coordinates`  
  - `title`: Title of the marker, of type `String`  
  - `infoHtml`: Popup HTML text to display in the balloon when the user clicks on the marker, of type `String`  
  - `onclick`: Event triggered when the marker is tapped

In the project, markers are received by the show map screen as the input argument. Then, location of the first (best ranked) element in the markers collection is used to center the map there. Map height is set to the screen height of the user device while the zoom level is set to
12. Since the default value of the width argument is already screen width of the user, it is unnecessary to assign that value to it explicitly.

5.2.10 Marker Details Screen

This screen is accessed when the user taps on a result marker in the show map screen. It contains the detailed information about the result. Data displayed here is same as the body of the accordion section for that result in the results screen. There is also the same choose button as in the accordion view and the comparison view.

Source code from this screen is not given here since it is exactly same as the body control of the accordion sections which is provided in sub-section 5.2.6.

5.2.11 History Screen

History screen is the initial screen of the history tab and it can be accessed by two different ways:

- Anytime when the user taps on the history tab
- Programmatically when the user taps the choose button from any of the three possible result views: accordion, map or comparison view

An example of history screen is shown in Figure 21-b. This screen is similar to the results screen because it also consists of an accordion control. However, history screen lists all the results from different searches that are previously chosen by the user instead of all results of a single search.

Moreover, bodies of the sections contain a button to delete that result from the history and indefinite number of items for connected access patterns for that search. Those items can be tapped to make a quick search related to the previous search. For example after finding a hotel depending his/her criteria, user can find a nearby restaurant easily by tapping the associated item. When he/she does so, interfaces screen is called to let user choose on which interface to make the search.

After the selection, unsurprisingly form screen is displayed but with one difference: input attributes that are connected to the output attributes of the source result are automatically filled with output values. To continue with the same example: in the input form of the restaurant search, latitude and longitude fields would be filled with the location of the hotel...
that the user has chosen before. Those connections between access patterns and input – output attributes are dynamically retrieved from the web service for the sake of generalness.

Source code below shows which controls and constructs the body of a result in the history accordion consists of. Title of a section is the name of the service mart that the result belongs to while its body consists of a button to delete the result from history, output name – value pairs of the result and a group control with indefinite number of item controls to go to the associated interface screen when tapped.

```javascript
Button("Delete", onclick={delete(...);})

list(property in historyTupleProperties){
    when(...){
        label(property._1 + ": ",style=boldLabelStyle)
        label(property._2)
        nl()
    }
}

group{
    list(connectionButton in historyConnectionButtons){
        when(connectionButton._4 == tupleJSON.tupleId){
            item(onclick={goToInterface(...);}) {
                label(connectionButton._1)
            }
        }
    }
}
```

### 5.3 Application Logic
Critical parts of the source code that is taking care of application logic are explained in this section. Application logic is encoded using mobl’s scripting language which is syntactically similar to JavaScript but as a big difference, it is a typed language [11]. Scripting code is used in three different circumstances in the project:

1. When a callback is defined inline, by enclosing within curly braces

```javascript
button("Choose",onclick={chooseResult(...); currentTab="History"; history1();})
```
2. Inside a function body

```javascript
function stringShorten(input : String, length : Num) : String{
  if(input.length > length){
    return input.substr(0, length) + "...";
  }
  return input;
}
```

3. Inside the script blog in a screen

```javascript
script{
  var first = true;
  foreach(prop1 in resultTupleProperties){
    ...
  }
}
```

Inline scripts are usually contain few expressions and are easy to understand; moreover most of them are already explained in the previous section. Here the focus is mostly on complex functions and script blogs. Scripting codes are discussed in the order they would be executed during a typical usage scenario.

### 5.3.1 Form Screen Script Blogs

First script blog in the form screen is responsible for pushing the significant properties of the input fields coming from web service into different global arrays depending on the data types of them. There are five different arrays for strings, decimals, integers, dates and booleans. Arrays contain tuples with four elements: name, value, connection id and caption of the input field. Example code for the boolean array is given below. Field name and connection id are directly pulled from the data while the caption is determined by another function called “getCaption”.

```javascript
var fieldsBoolean = Array<(String, Bool, String, String)>();

foreach(field in formResults.inputAttributes){
  ...
  else if (field.dataType == "BOOLEAN") {
    fieldsBoolean.push((field.name, false,
```
Second script blog is checking whether two special conditions are met or not, and perform actions if they are met.

1. If an input field is in latitude or longitude type, its initial value is set to the current location of the user detected from the GPS receiver of the mobile device.
2. If the user have already chosen some results and he/she is revisiting the form page in order to conduct a connected search, then the input fields are filled with the output data of the connected field. This condition check can overwrite the first one and it is done with the help of the function called fieldInConstraints.

```javascript
if(field._4 == "Longitude"){
    var coords = getPosition(true);
    field._2 = coords.longitude.toString();
}

if(fieldInConstraints(outputField._3, inputField._3, previousAPid, apId)){
    inputField._2 = outputField._2;
}
```

### 5.3.2 Get Caption Function

This function takes interface id, input field id and a JSON document containing the metadata information as input argument. It iterates over the JSON document to find the caption of that field and returns it. Caption is an alternate name for the input fields that is displayed to the user to increase the clarity of what that field is.

```javascript
function getCaption(intId:String, fieldId:String, metadata:JSON) : String{
    var serviceData = metadata.data;
    foreach(service in serviceData.services){
        if(service.serviceID == intId){
            foreach(input in service.inputSignature){
                if(input.__id__ == fieldId){
                    return input.caption;
                }
            }
        }
    }
}
```


5.3.3 Fields in Constraints Function
This function checks whether an input field matches with an output field from a connected search. To make that check, it takes connection id of the output, connection id of the input, access pattern id of the previous search and the access pattern id of the current search as input arguments. Then it iterates over an array of constraints and returns true if a tuple containing the input arguments exists in that array.

```javascript
function fieldInConstraints(ell : String, ... , el4: String) : Bool{
    foreach(element in constraints){
        if(element._1 == el1 && ... & element._4 == el4){
            return true;
        }
    }
    return false;
}
```

5.3.4 Results Screen Script Blogs
First script blog inside the result screen generates a JSON string containing the service mart id, interface id and all the input parameters of the user search query. This string is then sent to the web service in order to launch a search.

```javascript
JSONData = "{"sourceId":"" + martId + ",""accessMethodId":"" + intId + ",""inputTuple":{";
foreach(field in fieldsString){
    JSONData = JSONData + "," + field._1 + ":" + field._2 + ",";
}
JSONData = JSONData.slice(0, JSONData.length-1);
JSONData = JSONData + "}";
```

Second script blog is responsible for the following tasks:
- Re-initializing the result markers collection with the location marker of the user. This collection is sent to the show map screen as input argument
• Adding results that have location information to the result markers collection

```javascript
var lon = Reflector.get(tuple,"longitude");
var lat = Reflector.get(tuple,"latitude");
resultMarkers.add(MapMarker(coords=Coordinates(latitude=lat, longitude=lon), title=name,
onclick={ markerDetails(...) }));
```

• Iterating on every output property of every search result and pushing them to an array called “resultTupleProperties”. Since the names and the number of the properties are indefinite, they have to be accessed using object reflection in the runtime. mobl supports this process by means of the static methods of the Reflector type. Main output of the interface is determined with the “getMainOutput” function.

```javascript
var mainOutputName = getMainOutput(intId, metadata);
foreach(tNumber in Reflector.getProperties(executionResults.tuples)){
    var tuple = Reflector.get(executionResults.tuples, tNumber);
    ...
    foreach (propertyName in Reflector.getProperties(tuple)){
        var propertyValue = Reflector.get(tuple,propertyName);
        var isMain = false;
        if(propertyName==mainOutputName){
            isMain = true;
        }
        resultTupleProperties.push((propertyName, propertyValue, tNumber, isMain));
    }
}
```

• Pushing name of the sections and their body controls the array called “resultAccordion”. This array is then used to generate the accordion control. Body control is not given here as it is already explained in the sub-section 5.2.6.
resultAccordion.push((name, control () {...}));

5.3.5 Get Main Output Function
This function takes interface id and a JSON document containing the metadata information as input arguments. Then, it iterates over the JSON document to determine and return the name of the main output of the specified interface. If no matching main output is found, “tupleId” is returned as the main output.

```javascript
function getMainOutput(intId : String, metadata : JSON) : String{
    var serviceData = metadata.data;
    foreach (service in serviceData.services){
        if (service.serviceID == intId){
            return service.mainOutput;
        }
    }
    return "tupleId";
}
```

5.3.6 Compare Screen Script Blog
Purpose of the script blog in the compare script is to fill the compared properties array with the tuples of following three elements for each result:

1. Value of the main output, this value is taken from the “resultTupleProperties” array and used in the first column of the comparison view.
2. Value of the output that is specified by the user, this value is also taken from the same array and used in the second column of the comparison view.
3. A JSON string that contains all the information about that specific result, this value is taken from the input arguments of the screen and it is used as an input to the choose result function.

```javascript
var tuple = Reflector.get(tupleJSON, prop2._3);
comparedProperties.push((prop1._2, prop2._2, tuple));
```

5.3.7 Choose Result Function
Choose result function is the most complex scripting code in the project with many different tasks such as:
- Adding new elements for the selected result to the history connection buttons and constraints arrays with the values received from the web server through connection patterns resource

```javascript
historyConnectionButtons.push({connectionPattern.name, apId,
connectionPattern.targetAccessPatternId,
tupleJSON.tupleId});

foreach(constraint in connectionPattern.constraints){
    constraints.push({constraint.sourceAttributeId,
        constraint.targetAttributeId,
        connectionPattern.sourceAccessPatternId,
        connectionPattern.targetAccessPatternId});
}
```

- Iterating over the properties of the selected result and pushing them to the “historyTupleProperties” array

```javascript
foreach(propertyName in Reflector.getProperties(tupleJSON)){
    var propertyValue = Reflector.get(tupleJSON,propertyName);
    historyTupleProperties.push({propertyName, propertyValue,
tupleJSON.tupleId});
    ...
}
```

- If the item contains location information, determining the longitude, latitude and the main output values, adding a new marker to the history markers collection using those values in the location and the info html of the marker.

```javascript
if(latlong){
    var lon = Reflector.get(tupleJSON,"longitude");
    var lat = Reflector.get(tupleJSON,"latitude");
    var metadata = repository.metadataResource();
    mainOutput = Reflector.get(tupleJSON, getMainOutput(intId, metadata));

    historyMarkers.add(MapMarker(coords=Coordinates(latitude=lat,
        longitude=lon), infoHtml= "<b>" +
        martDesc + "</b>:<br/>" + mainOutput));
}
```
Adding a new section to the history accordion by pushing an element to its array with the title of first 25 characters of mart description and the body control. Body control is not explained here as it is explained in detail in section 5.2.11.

```javascript
historyAccordion.push((stringShorten(martDesc,25), control () {...}));
```

### 5.3.8 Delete Function
Delete function takes mart description, id and the main output of the results as input arguments. It removes following values from associated arrays:

- Section from the history accordion which has the title equal to given mart description
- All properties from history tuple properties which has the given id
- All connection buttons from history connection buttons which has the given id
- Marker from the history markers which has the title equal to given main output

Only one of those four deletion operations is shown in the below code snippet as they all are very similar.

```javascript
foreach(marker in historyMarkers){
    if(marker.title == mainOutput){
        historyMarkers.remove(marker);
    }
}
```

### 5.3.9 Reset Function
Reset function takes no input arguments and it clears all the elements in the history accordion array, re-initializes the history markers collection with the location of the user and refreshes the history screen.

```javascript
function reset(){
    historyAccordion.splice(0,historyAccordion.length);
    historyMarkers.destroyAll();
    historyMarkers.add(MapMarker(coords=getPosition(true),
                                   infoHtml="You are here"));
    history1();
}
```
5.3.10 Go To Interface Function

This function carries out the operations that are needed when the user clicks on a connection button inside the history accordion. Those operations are:

- Add required the properties of all the output fields of the source interface to an array called “searchOutput”

```javascript
var int = repository.serviceInterfaceSpecifiedResource(oldIntId);

foreach(outputField in int.outputSignature)
    var fieldValue = Reflector.get(tupleJSON, outputField.name);
    searchOutput.push((outputField.name, fieldValue, outputField.patternSignatureComponentId, apId));
```

- Find out the mart id, mart description and access pattern id associated with the target interface

```javascript
var ap = repository.accessPatternSpecifiedResource(apId);
var martId = ap.serviceMartId;
var mart = repository.martSpecifiedResource(martId);
var martDesc = mart.description;
```

- Set search tab as the active tab and call interfaces screen inside that tab using the values found out.

```javascript
currentTab = "Search";
interfaces(martId, martDesc, apId);
```
6 Conclusion and Future Work

In a world where everything is heading to being mobile, one cannot expect that searching will not follow the trend. Especially, multi-domain and exploratory search is quite appropriate for moving to mobile because such search applications provides possibility to obtain results that are possible with regular searching only in multiple iterations. That characteristic of multi-domain and exploratory search applications saves time and decreases the number of user interactions required which are two critical sources in mobile devices.

Moreover, mobile devices that are equipped with GPS receivers can also provide valuable user location information for the search application that can be related to search results and used for exploration.

This project emerged in an attempt to exploit this convenience of mobile devices for multi-domain and exploratory search. The application will hopefully fill the gap in the area of mobile search applications. Some future work may be devoted to improvements of the web applications in the following areas.

- Despite the practicability and the generality of the mobile web applications, some users may prefer native applications for their mobile device. A future research may be devoted to the feasibility and realization of device specific native application version of the web application, for example for iOS and Android.
- When the Mart Repository and Query Processor ReST APIs become accessible publicly and the host location of the application is moved to the same server, data access of the application can be made through mobl features such as HTTP Access. This would result in a more generic data model for the application.
- Since both mobl language and the mobile devices are relatively new technologies and are still developing, possible features in the future may be implemented inside the application in order to increase usability, feature set or code cleanliness.
- Map view may be enriched with features such as different symbols for different result types (in history map), direction drawing or fetching updated results when the user moves to another location on the map.
- During the project focus was on the usability and functionality rather than look, feel and catchiness. Those aspects of the application may be improved using graphics and styling features of the language.
7 Bibliography


[22] A. Bozzon, M. Brambilla, S. Ceri and S. Quarteroni, “Location-based Web Data Integration Search and Exploration”.

79