ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

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

M.Sc. THESIS

Atakan ARAL

Department of Computer Engineering

Computer Engineering Programme

JUNE 2012

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

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

M.Sc. THESIS

Atakan ARAL
(504091503)

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Asst. Prof. Dr. Marco BRAMBILLA

JUNE 2012

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ALANA ÖZEL MODELLEME DİLLERİ İLE YAPILANDIRILMIŞ AĞ VERİSİ ÜZERİNDE MOBİL ARAMA UYGULAMALARI GELİŞTİRMESİ

YÜKSEK LİSANS TEZİ

Atakan ARAL
(504091503)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Asst. Prof. Dr. Marco BRAMBILLA

HAZİRAN 2012

Atakan ARAL, a M.Sc. student of ITU Institute of Science and Technology 504091503 successfully defended the thesis entitled **"DEVELOPMENT OF MOBILE SEARCH APPLICATIONS OVER STRUCTURED WEB DATA THROUGH DOMAIN-SPECIFIC MODELING LANGUAGES"**, which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Asst. Prof. Dr. Marco BRAMBILLA** Politecnico di Milano	.....
Co-advisor :	**Assoc. Prof. Dr. Zehra ÇATALTEPE** İstanbul Technical University	.....
Jury Members :	**Asst. Prof. Dr. Feza BUZLUCA** İstanbul Technical University	.....
	Asst. Prof. Dr. Tolga OVATMAN İstanbul Technical University	.....
	Instr. Dr. Suzan ÜSKÜDARLI Boğaziçi University	.....

Date of Submission : **04 May 2012**
Date of Defense : **15 June 2012**

Babama,

FOREWORD

According to the requirements of the Top Industrial Managers for Europe (T.I.M.E) double degree program and the bilateral agreement between the institutions, this thesis has been completed at PoliMI and then represented with some minor improvements at ITU.

I would like to thank my mother, my father and my brother for their enormous support to my education, for their faith in me and for the inspiration.

This thesis depends on work that was carried out by me and M.Sc. İlker Zafer AKIN. I would like to send my sincere thanks to him for all his support and hard work.

Last but not least, I would like to thank my advisors from both institutions: Asst. Prof. Dr. Marco BRAMBILLA from Politecnico di Milano (PoliMI) and Assoc. Prof. Dr. Zehra ÇATALTEPE from Istanbul Technical University (ITU) for their unfailing guidance, as well as to all other jury members for their precious advises that helped me to improve the thesis.

June 2012

Atakan ARAL
(Computer Engineer)

TABLE OF CONTENTS

Page

ABBREVIATIONS

AJAX : Asynchronous JavaScript and XML
API : Application Programming Interface
CSS : Cascading Style Sheets
GPS : Global Positioning System
GUI : Graphical User Interface
HTML : HyperText Markup Language
HTTP : Hypertext Transfer Protocol
iOS : iPhone Operating System
JSON : JavaScript Object Notation
PHP : PHP: Hypertext Preprocessor
ReST : Representational State Transfer
SeCo : Search Computing
SQL : Structured Query Language
SSL : Secure Socket Layer
UML : Unified Modeling Language
VPN : Virtual Private Network
XML : Extensible Markup Language

LIST OF TABLES

Page

LIST OF FIGURES

Page

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

SUMMARY

Searching on mobile devices gained significance as smartphones became widely available and the quality of mobile networks increased. Established web search interfaces and methodologies do not show the same usability and efficacy when they are directly migrated to mobile environments. This is due to their difference from personal computers in areas such as form of interaction and screen size as well as user goals and expectations.

Our objective is to design a novel web search methodology that is optimized for mobile devices. We aim to quicken and ease the search process in smartphones while maintaining at least the same accuracy. Our approach includes notions of multi-domain search and exploratory search and pays specific attention to the user interfaces which take advantage of these notions in mobile devices.

Information from different semantic fields of interest can make sense by means of multi-domain search. For instance, a multi-domain search application should perform three searches to answer sample query "good computer graphics conference October 2012 Istanbul reasonable 5-star hotel", since there are three semantic fields (conference, city and hotel). These searches may be carried out on topic dedicated search engines and should be somehow related to each other. Building a list of computer graphics conferences in Istanbul in October 2012 along with nearby 5-star hotels and ordering the list according to conference rating, hotel price and proximity is one of the solutions for this specific search.

On the other hand, proponents of exploratory search argue that user may not always be an expert in the field of search so they propose that he/she should be aided in formulating his/her interest, in exploring most relevant and credited information sources and in correlating elements of these sources. This process of assistance can be accomplished by asking user to choose a topic then to specialize on that topic and related information sources step by step, and finally asking for input data specific to that sub-topic. During or after the search process, searches in related topics may be offered to user in order to let him/her enhance the query.

Requirement for more advanced ways of presentation arises as a natural outcome of multi-domain and exploratory search approaches. It is necessary to treat data from different semantic fields differently and to present them with the aid of different interface elements such as geographical maps, item lists or data tables. In addition, further specifying the query as well as filtering and sorting results according to various criteria should be possible and straightforward for user via certain interface cues.

We analyze applicable solutions for recent and innovative ideas on mobile web search including multi-domain search and exploratory search. We also demonstrate how these

solutions can be cooperated in order to enhance and ease search process for complex queries on mobile devices, as well as, what kind of user interface elements should be used to fully support them. Additionally, a practical application of the discussed solutions is presented to clarify technical issues. The application is designed and implemented by conforming software engineering and user interface design principals.

Experiments demonstrate that the users who do not use the proposed search paradigm have to interact more with the computer and enter longer textual inputs. Obtained results confirm that the objectives of the thesis are met.

ALANA ÖZEL MODELLEME DİLLERİ İLE YAPILANDIRILMIŞ AĞ VERİSİ ÜZERİNDE MOBİL ARAMA UYGULAMALARI GELİŞTİRMESİ

ÖZET

Akıllı telefonlar ve tablet bilgisayarların yaygınlaşması, kablosuz ağlar ve telefon şebekesi veri iletişim kalitelerinin yükselmesiyle mobil cihazlar üzerinden internet erişimi hızla yaygınlaşmıştır. Bu durum masaüstü ve dizüstü bilgisayarlar için geliştirilmiş bir çok ağ uygulaması gibi, internet arama motorlarının da mobil cihazlarda kullanımını artırmış, bu uygulamaların mobil versiyonlarına olan ihtiyacı ortaya çıkarmıştır.

Ancak yerleşmiş internet arama yöntemleri ve arayüzleri doğrudan mobil ortamlara taşındığında aynı kullanılabilirlik ve etkinliği gösterememektedirler. Bu durumun nedeni, bu yöntem ve arayüzler geliştirilirken hedeflenen çalışma ortamı ile mobil cihazların farklılıklarıdır. Bunların başlıcalarını; insan-cihaz etkileşim yöntemleri, ekran boyutları ile kullanıcıların amaç ve beklentilerinin aynı olmaması olarak sıralayabiliriz. Yine akıllı telefonların tipik özellikleri arasında kişisel bilgisayarlara göre daha düşük hesaplama yeteneği, aynı anda tek bir ekran ile etkileşebilme gibi kısıtları sayabiliriz. Belirtilen zorluklar aşılabilirse, mobil cihazların aramayı kolaylaştırıcı yetenekleri de tam olarak değerlendirilmiş olacaktır. Örneğin, çoğu akıllı telefon ve tablet bilgisayarda bulunan GPS alıcısı, aramada girdi olarak kullanılabilecek çok değerli konum verisi sağlamaktadır.

Bu tez çalışmasında amacımız, mobil aygıtlar için en uygun şekle getirilmiş yeni bir internet arama metodolojisi tasarlamaktır. Özellikle karmaşık sorgular için arama süreci hızlandırılıp kolaylaştırılırken, en azından aynı kesinliğin korunması hedeflenmektedir. Yaklaşımımız çok-alanlı arama ve keşifçi arama gibi kavramları içermekle birlikte ve bu kavramlardan en iyi şekilde yararlanmayı sağlayacak kullanıcı arayüzlerine de özellikle önem vermektedir. Tasarlanan yöntem kullanıcının en az sayıda etkileşim yapmasını amaçlamaktadır. Özellikle, dokunmatik cihazlarda zaman alıcı ve yorucu bir işlem olan klavyeden metin girme işleminden mümkün olduğunca kaçınılmaya çalışılmıştır. Ayrıca, genel amaçlı ve mobil uygulamalara özel kullanıcı arayüzü tasarım prensipleri incelenmiş, bunların internette aramaya uygun olanları uygulanmıştır.

Çok-alanlı aramada amaç, farklı anlamsal alanlardan gelen arama sonuçlarını kullanabilmektir. Örneğin kullanıcı genel amaçlı bir arama motoruna "iyi fizik konferansı ekim 2012 milano uygun fiyatlı 5-yıldızlı otel" şeklinde bir sorgu girdiğinde aslında 3 farklı anlamsal alandan sonuçlar beklemektedir. Bu alanlar konferans, şehir ve otel alanlarıdır. Tecrübeli bir kullanıcı, alana özel arama motorlarını kullanarak bu 3 alanda arama yapıp el yordamıyla elindeki sonuçları ilişkilendirebilir. Bir çok-alanlı arama uygulaması ise bu işlemleri otomatik olarak yapabilmelidir. Verilen örnek için uygulanabilecek bir çözüm Ekim ayı içinde ve Milano'da düzenlenecek olan Fizik

konferanslarının ve yakınlarındaki 5 yıldızlı otellerin bir listesini oluşturmak ve bu listeyi konferans etki faktörü, otel fiyatı ve yakınlık gibi etmenlere göre sıralamak olabilir. Genel olarak, yapılacak birden çok arama işlemi ve bunların sonuçlarının verilen kıstaslara göre ilişkilendirilmesi otomatik olarak yapılmalıdır.

Keşifçi arama ise kullanıcının arama sürecinin her adımında desteklenmesi gerektiğini önerir. Bu adımlar, kullanıcının ilgilendiği aramayı formüle etmek, en ilgili ve en güvenilir bilgi kaynaklarını bulmasını sağlamak ve bu kaynakların elemanlarını ilişkilendirmek olarak verilmiştir. Bu yardımın nedeni, aramayı yapacak olan kullanıcının, arama yapacağı alanda ve kullanacağı arama teknolojileri konusunda uzman olmayabileceği gerçeğidir. Keşifçi aramayı gerçekleştirebilmek için uygulanabilecek bir yöntem, kullanıcının genel bir konu belirlemesini istemek daha sonra ise adım adım bu konuda özelleşerek bilgi kaynağına ulaşmasını sağlamak olabilir. Daha sonra ulaşılan bilgi kaynağına özel olan girdiler kullanıcıdan alınır ve sonuçlar kullanıcıya sunulur. Arama süreci sırasında ya da sonrasında, kullanıcıya ilişkili alanlardan sonuçlar önermek de sorguyu geliştirmeye yardımcı olacaktır.

Çok-alanlı ve keşifçi aramanın doğal bir sonucu olarak daha gelişmiş gösterim yöntemlerine ihtiyaç duyulmaktadır. Farklı alanlardan gelen sonuç kümeleri farklı şekillerde ele alınmalı ve farklı arayüz öğeleri aracılığıyla sunulmalıdır. Örneğin konum bilgisi içeren sonuçlar harita üzerinde gösterilebilirken, daha metinsel sonuçların çizelge ya da liste olarak sunulması uygun olacaktır. Ayrıca kullanıcı arayüzü öğeleri, sonuçları farklı kıstaslara göre yeniden sıralamayı, elemeyi ya da sorguyu geliştirmeyi kolaylaştırmalıdır. Tüm bunlar yapılırken, akıllı telefonlar ve tablet bilgisayarların kısıtları ve güçlü yanları göz önünde bulundurulmalıdır.

Yukarıda belirtilen yaklaşımlar göz önünde bulundurularak, mobil cihazlar için yeni bir internet arama yöntemi önerilmiştir. Önerilen yöntemi kısaca özetlemek gerekirse, kullanıcı ilk adımda kendisine sunulan anlamsal alan listesinden aramasına başlayacağı konuyu seçmektedir. Daha sonra, seçilen alanda yapılabilecek farklı arama tiplerinden birine karar vererek sorgusunu özelleştirmeye başlamaktadır. Bir sonraki adım ise seçilen alan ve arama tipine uygun, alana özel bilgi kaynaklarının listelenmesi ve kullanıcının seçimini yapmasıdır. Yapılan seçimlere göre oluşturulan ve veri tipine göre özelleşmiş olan girdi alanları kullanıcıya sunulur. Girdilerin sağlanmasıyla bir adet alana özel arama tamamlanmış olur. Sonuçlar harita, liste ve çizelge gibi farklı gösterim şekilleriyle sunularak, kullanıcının kararını vermesi beklenir. Seçilen sonuç arama geçmişine kaydedilir ve arama geçmişinin mevcut durumu gösterilir.

Arama geçmişi aynı zamanda çok-alanlı aramaya geçiş noktası olarak görev yapmaktadır. Buradaki sonuçlardan hareketle yeni ilişkili aramalar yapılabilmektedir. Her alana özel olan ilişkili aramalar listesinden biri seçilir ise ikinci alana özel arama süreci başlayacaktır. İlk aramaya benzer şekilde kullanıcının bir bilgi kaynağı seçmesi ve girdileri sağlaması beklenecektir. Ancak bu kez farklı olarak alanlar arasındaki ikili ilişkiler göz önünde bulundurulacak, sonuçlar listelenirken bu bağlantıya göre eleme ya da sıralama yapılacaktır. Örneğin, ilk aramanın alanı konferans ise ve seçilen konferanstan sonra ilişkili bir alan olan otele geçiş yapılırsa, sadece konferansın düzenleneceği şehirdeki oteller listelenecektir. Ayrıca otellerin sıralamasında konferans konumuna olan uzaklık da göz önünde bulundurulacaktır.

İkinci alana özel aramanın da tamamlanmasından ve bir sonuç seçilmesinden sonra bu sonuç da arama geçmişine eklenecektir. Bu noktadan sonra kullanıcı, geçmişteki

iki sonuçtan biriyle ilişkili olan istediği kadar alanı aramasına ekleyebilir ve aramasını geliştirebilir. Bu sırada önceki aramalarda seçilen tüm sonuçlar saklanmaya devam edecek, liste veya harita şeklinde incelenebilecek, istenmeyen sonuçlar listeden çıkartılabilecektir. Arama süreci, kullanıcı başlangıçtaki karmaşık sorgusuna cevap olan tüm alanları eklediğinde ve ulaştığı sonuçtan tatmin olduğunda sonlanacaktır.

Önerilen yöntemin gerçeklenmesini uygulamalı olarak göstermek ve teknik konulara açıklık getirmek amacıyla web tabanlı bir mobil uygulama geliştirilmiştir. "Model-view-controller" mimari modeline uygun olarak, bir alana özel dil olan mobl'da geliştirilen uygulama, tüm modern tarayıcılar ve cihazlarda herhangi bir eklentiye gereksinim duymaksızın çalışabilmektedir. Uygulamanın veri modeli, iş mantığı ve kullanıcı arayüzü ile ilgili teknik detaylar ayrıntılı olarak tezde verilmiştir.

Yapılan deneylerde, örnek iki arama görevini önerilen uygulamayı kullanmadan gerçekleştiren kullanıcıların hem daha çok etkileşimde bulundukları hem de daha çok metin girişi yaptıkları görülmüştür. Deney sonuçları önerilen arama yönteminin hedeflenen iyileştirmeleri sağladığını göstermektedir.

Sonuç olarak, mobil internet arama uygulamaları alanındaki boşluğu doldurmaya katkıda bulunacak bir yöntem önerilmiş ve gerçeklenmiştir. Yöntem, farklı anlamsal alanlardan bilgi gerektiren karmaşık sorguların cevaplanması işini hızlandırmakta ve güvenilirliğini artırmaktadır. Hız artışı kullanıcı-cihaz etkileşiminin en az seviyede tutulmasıyla, güvenilirlik artışı ise her alana ait itibarlı bilgi kaynaklarının ön tanımlı olarak sunulmasıyla sağlanmıştır.

1. INTRODUCTION

Users of smartphones, tablets and other mobile devices are able to search for information on the move thanks to the developments and proliferation in the wireless internet connectivity. Performing search tasks on mobile devices is quickly becoming an expectancy or even a requirement for many people.

1.1 Problem Definition

As a result of increase in the demand for mobile web search, search applications that have optimized interfaces for mobile devices are required. However, simply resizing personal computer versions of search engines for mobile devices is not efficient due to the unique characteristics of these devices such as small screen size, touch-sensing input and difficulty in inputting long text. Mobile versions of the general purpose search engines usually take too much time when it comes to answering complex queries or even fail to provide useful results especially when used by nonexpert users.

1.2 Purpose of Thesis

We propose a novel method to handle query formulation and elaboration, as well as result exploration and visualization on mobile devices. Our aim is to let user build up complex queries and explore results using minimum number of interactions, especially textual input. Easing the process of combining results from different semantic fields and gradually extending the scope of the search is another goal.

The method makes use of multi-domain search, in which result set consists of correlated results from separate domains, and exploratory search, in which result set is expanded by adding one connected domain at a time. User interface elements and visualization methods support these strategies.

A web-based mobile application is designed and developed in order to demonstrate practical issues about the proposed method. A domain specific language for mobile

browsers called mobl [5] is chosen for the implementation, while data operations are carried out through a ReST API using JSON data interchange.

1.3 Author's Contribution

Work done for this thesis is abundantly integrated with the other parts of the Search Computing project. In order to avoid any misperception of the actual contribution, areas of contribution by the author are listed in this section.

- Existing theoretical ideas for search systems are materialized for the case of mobile search. Methods that let them apply to real life scenarios are suggested.
- A novel search paradigm which aims to reduce number of interaction steps between the user and the mobile device and handles complex queries is suggested.
- Three data visualization methods are suggested for different kinds of data in an attempt to optimally summarize high-dimensional search results.
- Design principles for mobile graphical user interfaces are studied and applicable ones for the search paradigm are identified. A user interface that is most user-friendly for the case of mobile multi-domain search is suggested.
- A data layer which makes use of existing infrastructure of the Search Computing project is designed and developed. It retrieves necessary data for the search paradigm through a ReST API using JSON data interchange
- Suggested search paradigm, visualization methods and user interface are implemented as a web based mobile application.
- An evaluation is carried out to demonstrate that suggested search paradigm let users carry out their complex searches easier and faster than traditional search tools.

1.4 Structure of the Document

An overview of the technologies that this work benefits from partially or as a whole, and especially the Search Computing project are given in the next two chapters. Chapters 4 and 5, describe the proposed solution in detail and explain implementation

issues, respectively. Results and discussion of the evaluation are provided in chapter 6. Finally, the last chapter concludes the thesis and provides future work ideas.

2. THEORETICAL BACKGROUND

2.1 Web Search

2.1.1 Multi-domain search

In recent years, digital data has increased rapidly forcing applications to deal with huge repositories. Helping users find informationIt is now a requirement. As a consequence of this, search engines quickly started to have an important role in any type of information system. As user requirements change depending on the amount and kind of information, search engines also need to adapt themselves [6].

There are two different trends for searching for information. The first trend is to look for information and objects of interest instead of Web pages which illuminate such objects. Hence, user expects to be satisfied for his/her search need using the search engine directly. For instance, when the temperature in New York City is asked to the search engine, user expects to see directly this information instead of a list of pages carrying it. The other trend is related to user experience. When users get confident about using search engines, queries become more complex. The meaning of the query may be more than what is possible to express using a few keywords. Therefore, answers must be more structured than a list of Web pages [7].

In order to meet 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. Furthermore, result sets may contain different types, such as images or videos for each domain of interest. Domain, content type and layout of the result set should meet the user expectations [1].

Three different query examples are given below [1]:

- **Query 1:** "Washington D.C." is a mono-domain query which concerns only one semantic domain, city. The only issue is visualization of the specified domain.

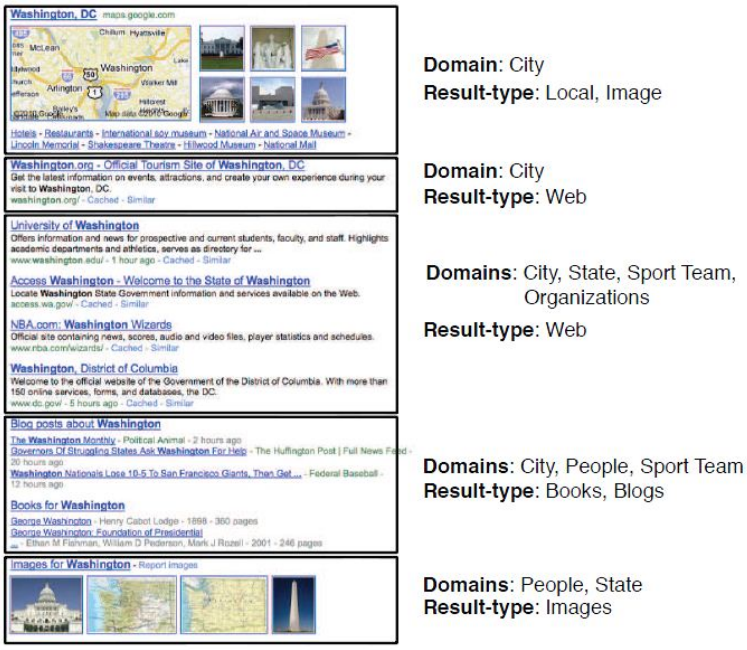


Figure 2.1: An example of ambiguous domain result list layout for Query 2 [1].

- **Query 2:** "Washington" in Figure 2.1 is an ambiguous query which can include many semantic domains. It may be the capital of the United States, the first president of the United States or the state. In order to show results related to user intent, the query should be disambiguated. If this is not possible, the result set should grant coverage of the most expected intentions.

- **Query 3:** "rock concert Washington July 2010 good restaurant" is a multi-domain query which is related with several domains. The query refers to cities, restaurants and concerts. The main issue here is to build a result set connecting different concepts together. A solution for this issue which splits the results in three lists is shown in Figure 2.2.

These queries can produce different types of information. For example, result of the US capital can be a map, news article, image, etc. Generally, different result types come from different search engines and need to be aggregated. This operation should be done carefully but unfortunately not much attention has been devoted to the display of search results [1].

Visualization of search results is an important issue for search engines in order to make user perceive quality [8]. Web search engines such as Google, Yahoo! and Bing, also called general-purpose web search engines, nowadays contain domain-specific

Figure 2.2: An example of multi-domain result list layout for Query 3 [1].

functionalities such as image or video search 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, results coming from different domain-specific search engines need to be aggregated by the engine in order to be displayed on a map. For Query 2, a result diversification task need to be performed by a general-purpose engine, splitting the result list in order to include information related to all identified domains [1].

Currently most search systems do not manage multi-domain queries correctly. Search Computing (SeCo) is an approach which takes this issue into account. SeCo proposes some methods and tools for handling multi-domain queries by automatically uniting results from diverse search services to create an extensive answer, combining relevant pieces of information from different domains. It aggregates results at system level and produces a single result list including the combinations of results from different domains. "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" [1]. More details on SeCo's approach to multi-domain search is given in chapter 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

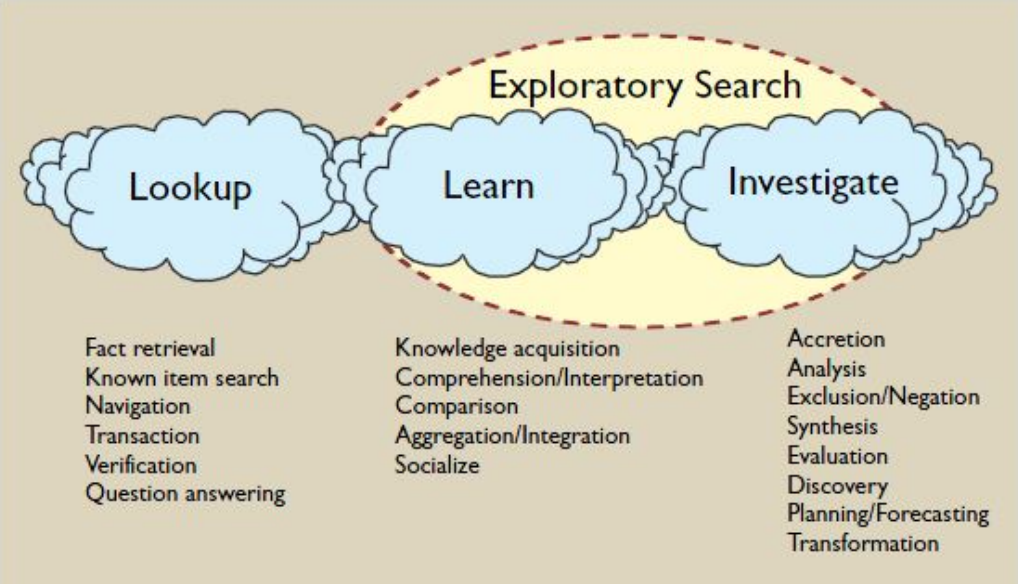


Figure 2.3: Types of the search activities and the role of exploratory search [2].

throughout the life cycle 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 [9].

Exploratory search provides information exploration for following types of users: [10]

- 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

Search activities can be categorized into three types: lookup, learn and investigate. First kind of search activity, which is also called fact retrieval is already one of the most successful applications of computers. In lookup search, exploratory search is not particularly effective, however it can make significant contributions to the other two categories of search activities. In learning search, user iterates over materials that require cognitive processing. Investigation search also requires iterations but takes longer time and the results are more critically evaluated. It should be also noted that those categories are not independent sets as most users perform those three kinds of searches in parallel or nested as shown in Figure 2.3 [2].

2.2 Design Principles for Mobile Graphical User Interfaces

User interface is one of the most important factors when an application is assessed by its user since it is the part where the user mostly interacts with. Even if the application is capable of performing expected tasks, 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 [11].

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.

2.2.1 General design principles

Following design principles are applicable to any computer system that contains user interaction including desktop programs, mobile applications, web applications and web sites [12].

- **Accessibility:** It should be designed to be usable by as many people as possible, without modification. This includes elderly and handicapped users
- **Aesthetical Beauty:** Its elements should be aesthetically in contrast, aligned, grouped and colored.
- **Availability:** All of its public objects should always be available for its users.
- **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 to 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:** People's requirements should always take precedence over technical ones when they conflict.
- **Visibility:** Its status and usage should be clearly visible.

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

- Less computational capacity
- Smaller screen size and resolution [11]
- Ability to be used in different orientations like landscape and portrait [11]
- Responds to hand gestures instead of clicks [11]
- User can interact with a single screen of a single application at a given time [11]
- Different application goals and user expectations [13]
- Different kind of usage environment [13]

While most of the previously given principals are perfectly valid for mobile devices, additional mobile characteristics and limitations impact their significance. 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 [13].

2.2.2 User interfaces for searching

Searching interfaces are typically designed extremely simple, sometimes as simple as only a field to enter the query [13]. This tradition is a legitimate one supported by the following reasons [8]:

- Searching process and its interface is not the goal for the user. He/she is interested in 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 leads to favoring simpler interfaces.

In addition to simplicity, following design principles are proposed for search interfaces [8]. Although they usually coincide with general design principals, they are presented again with examples for searching.

- Efficient and informative feedback should be provided for the user. Some examples can be: relevance indicators, term suggestion or correction, and 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 to speed up common tasks.
- User errors should be minimized by spelling correction, term expansion and supporting synonymies.
- Pages should be designed in a consistent style and they should not be distractive.
- User should be able to reverse his/her actions, such as cancelling a query and returning to the previous result set.

Surveys show that average query length is smaller and variation is narrower for mobile devices than their desktop counterparts [14]. 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 of words), anticipating common queries and voice-entered query terms can be used efficiently to decrease the impact of this limitation and improve convenience of query entry [8].

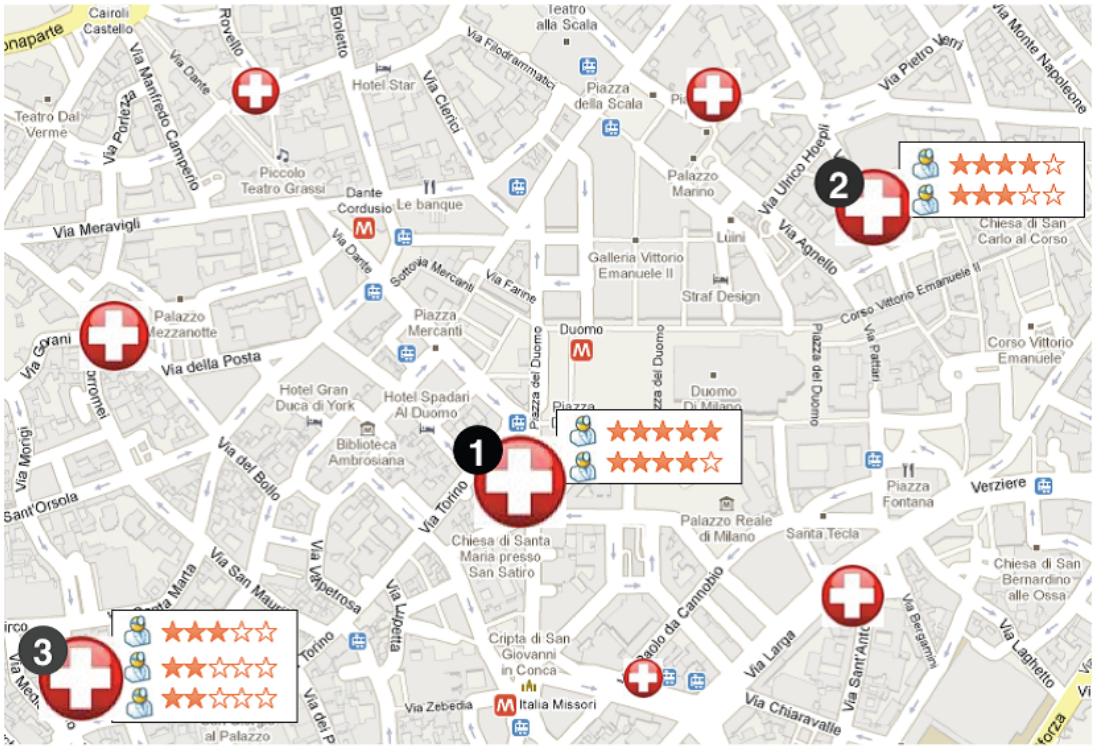


Figure 2.4: Map Visualization of results for hospital search [3].

2.2.3 Visualization of information

Although result interfaces for mobile searches do not differ significantly from desktop counterparts [15], 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 [8].

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 as shown in Figure 2.4 [3].

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 [3].

Another kind of visualization appropriate for particular result types is timeline visualization. Timelines are useful for displaying results with time or date information.

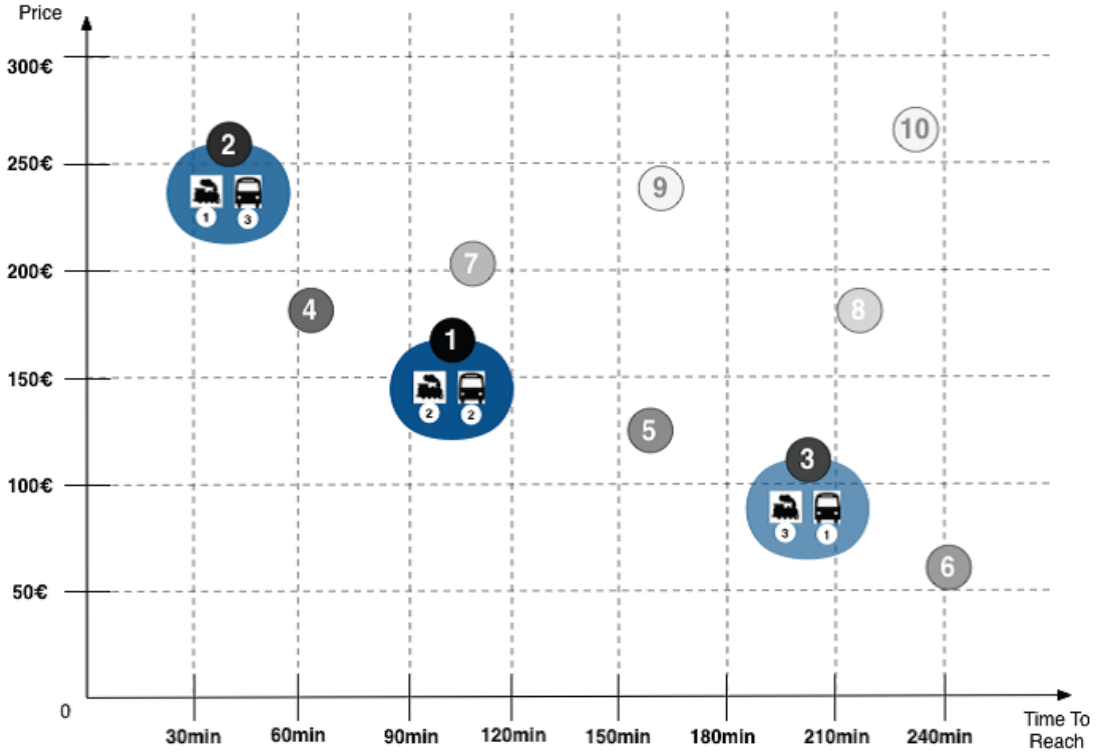


Figure 2.5: Timeline Visualization of results for transportation search [3].

,

Similar to map visualization, ranking and details of results can be displayed by using different symbols or numbers [3]. A sample timeline can be seen in Figure 2.5 that is for balancing the trade-off between duration and cost of transportation solutions. Ranks of the solutions are displayed along with the number and type of vehicles.

3. BACKGROUND ON SEARCH COMPUTING

3.1 What is Search Computing (SeCo)?

Search computing is a multi-disciplinary science which provides the abstractions, foundations, methods, and tools required to answer multi-domain queries over heterogeneous data sources. It is the outcome of a project started on November 1st, 2008 and will last 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 [16].

"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" [4].

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 enough for this [17].

Given an extended version of the example above, "Where can I attend a scientific conference in a city within a hot region served by luxury hotels and reachable with cheap flights?", an expert user would make 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 whether 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 final step, a hotel is searched for from another system. Instead of these steps, SeCo aims to provide a system for supporting another type of search process explained below [16]. By this type of search, it is easier to find the exact solution.

- Several solutions which integrate all dimensions are built.

- A global rank function is employed for ranking solutions to order them and/or filter out low ranked ones.
- Browsing the result and typing query are supposed to be more user-friendly and less time consuming than traditional search.
- New domains can be added incrementally as the search proceeds in order to broaden the scope of the query.
- 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" [17]. On the contrary, domain-specific systems normally depend on knowledge of field and user expertise. In addition to discovering comprehensive and credited information sources for each field, the user should also figure out how to combine the results of such systems into a meaningful answer for his/her complex query.

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, which is called multi-domain search, ranking and ordering has to be done manually or by an automated system. However, it is not supported by most other data integration platforms. Search Computing supplies a platform in order to imply requests over various search services, where the weighted rankings of individual search results are taken into account for the results of the integrated requests [18].

The vision of Search Computing is to develop technologies and architectures mainly for two kinds of users [17]:

- Content providers who would like to provide their domain-specific search results to an integration system
- and application developers who would like to offer novel services built by composing multiple domain-specific content sources as an alternative to the existing general-purpose search engines.

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 are provided in order to express multi-domain queries and their answers. For instance, "Which drugs treat diseases those are likely to be associated with a given genetic mutation?" can be expressed using a multi-domain query. Query is broken down into sub queries (like "Which drugs treat which diseases?", "Which diseases are likely to be associated with a given genetic mutation?") and each domain-expert server registered in the system takes care of related sub-query (in this case, calls to previously registered servers named "Drug4Disease", "GeneticMutation2Disease"). After decomposition phase, query is translated into an internal format and an efficient plan is applied for query execution supported by an execution engine that sends calls to services via a service invocation framework. Query results are built by combining 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 [18].

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 employed to produce these results. These sources must be registered as services in the Search Computing Framework. Relationship between Service Mart and operation to be invoked on the service are defined for this purpose [18].

Service Mart Framework, shown in Figure 3.1, provides structure for wrapping and registering data sources, while Service Invocation Framework controls the technical

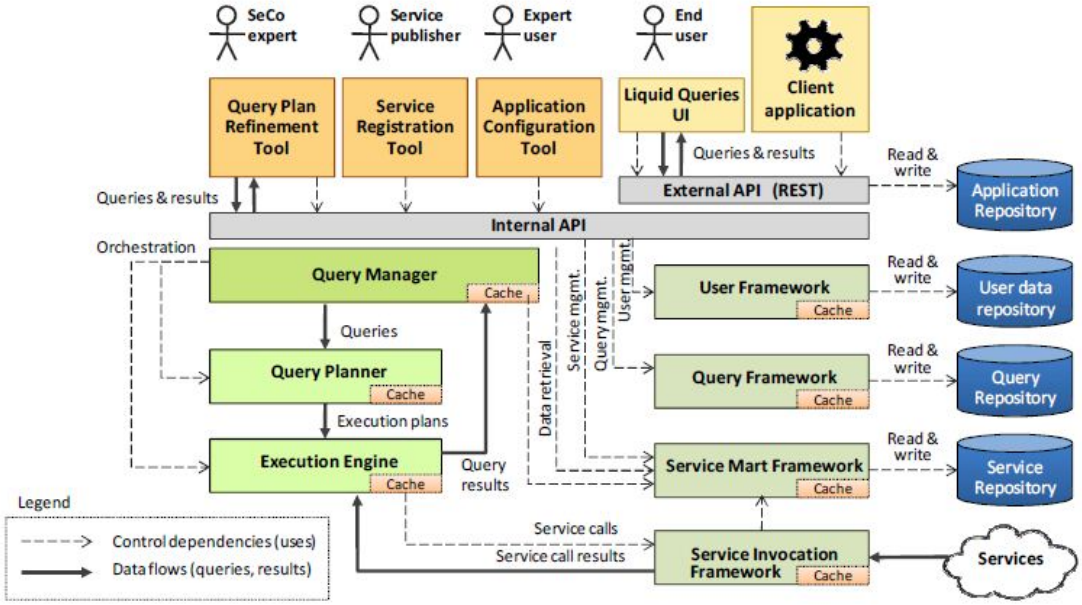


Figure 3.1: Overview of the Search Computing Framework [4].

issues concerning interaction with Service Mart such as Web service protocol and data caching.

User Framework supplies storage for registering users who 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" [4].

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 3.1 that an external

API is used to access end-user applications and interfaces in order to call them from any client environment [18].

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 such as ranked access, are supported by Service Marts. 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 [4].

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" that exists on the web, so there are Service Marts for "hotels", "flights", "doctors" and so on. Furthermore, some Service Marts are connected to each other to support their association. Service Marts and their connections form a network which may be used as a high-level interface for queries [4].

Mapping to several data sources, which may be APIs, web services or materialized data collection, is required for implementing a Service Mart. Therefore, the Service Mart concept provides a regular view of the world [4]. In the following sub-sections, a top-down view is given from the conceptual level to the physical level.

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" [4]. For instance, Service Mart for movie has both single-value attributes (Title, director, score, year, language) and repeating groups (Genres, openings, actors), each has sub-attributes. One level of parentheses is used for repeating groups, like in the following [4]:

$$Movie(Title, Director, Score, Year, Genres(Genre), Language,$$
$$Openings(Country, Date), Actors(Name))$$

$$Cinema(Name, Address, City, Country, Phone, Movies(Title, StartTimes))$$

$$Restaurant(Name, Address, City, Country, Phone, Url, Rating, Category(Name))$$

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, top-down process is not used; 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 [4].

3.2.2.2 Logical level

In this level, each Service Mart is correlated with one or more specific access patterns that express the route to access that Service Mart. An access pattern contains the characterization of each attribute or sub-attribute that can be input (I) or output (O). If the results are produced in an order, an output attribute called ranked (R) is used. Values of ranked attributes are normalized within the interval [0...1]. Below is an example of access pattern for the Movie Service Mart [4]:

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

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

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" [4].

In some cases, Service Mart may have less attributes than access patterns. For instance, if a cinema or restaurant is considered, address is an important object for them. However, 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 for user's location (U) and another for cinema/restaurant's (T) [4].

$$Cinema_1(Name^O, Address_U^I, City_U^I, Country_U^I, Address_T^O, City_T^O, Country_T^O,$$
$$Phone^O, Distance^R, Movie.Title^O, Movie.StartTimes^O)$$

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 [4]:

$$Shows(Movie, Cinema) : [(Title = Title)]$$

In this case, if the title of movie is equal to the title of any movie shown in the cinema, then the condition is satisfied. Below is another connection between cinemas and restaurants which is a directed pattern. The direction is from the first mart to the second

one which means the system first searches for cinemas and then for close restaurants. The address of the cinema will be the input location of a restaurant service, after finding a cinema close to the user's address [4].

$$DinnerPlace(Cinema, Restaurant) : [(Address_T = Address_U), (City_T = City_U), (Country_T = Country_U)]$$

"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 should 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 [4].

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 [4]:

- **Ranking Descriptor:** It classifies the service interface as a search service which produces ranked results or an exact service which produces objects without ranking. In exact services, there is the selectivity which is a positive number showimg 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 marked R, then it is an explicit ranking which can be either ascending or descending; if not, it is an opaque ranking. Search services do not necessarily present a result with ranked attributes.

- **Chunk Descriptor:** "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."

- **Cache Descriptor:** 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."

- **Cost Descriptor:** 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.

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 [4].

3.2.3 Web service registration and adaptation

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. Various tools are designed in Search Computing in order to build search service adapters. There are three different scenarios [4]:

- A Web service can be used to query data or results from different Web Services.
- Wrappers must be used in order to extract the data which is available on the Web.
- Data is not directly usable, thus it must be materialized first.

Results are in an interchange format in JSON which is not very easy to read and generate by humans but easy to parse and generate by computers. All instances of a Service Mart use the same interchange format, without regard to the service interface which creates them. Below is a sample JSON data for a movie instance containing both simple and collection objects [4]:

```
{
    "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 may return too many results, chunker module needs to change the chunk size [4].

3.2.3.2 Web pages

HTML pages are desired 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 3.2, the relation between data an extracted on the Web can be seen [4].

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

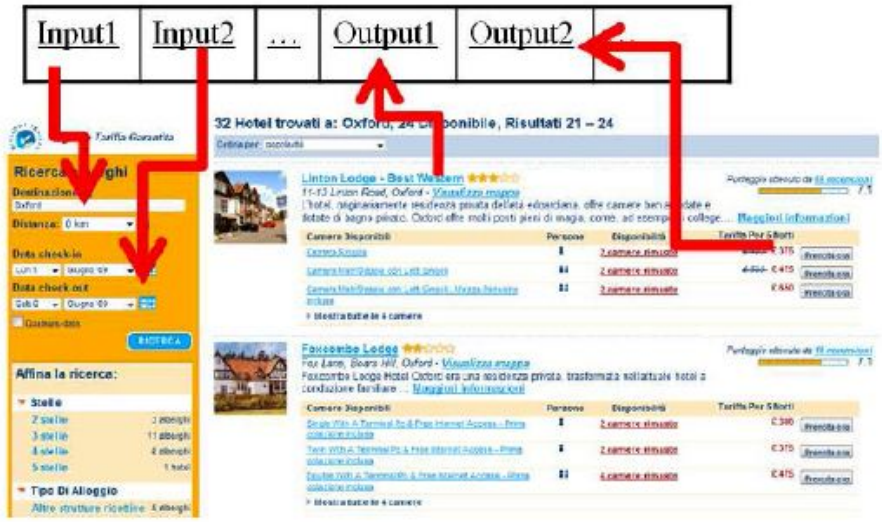


Figure 3.2: Data extraction from query results in Search Computing [4].

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 [4].

A materializer uses the modules mentioned for web services 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 3.3 [4].

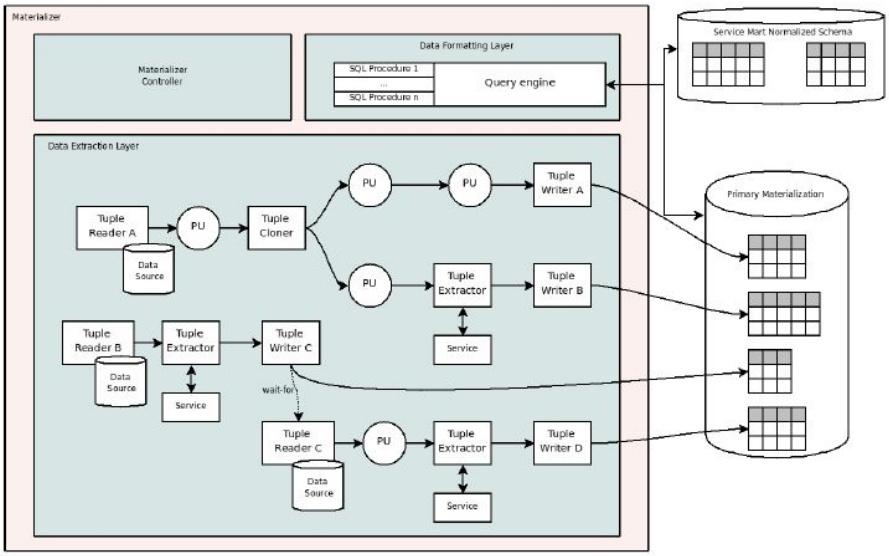


Figure 3.3: Materialization process in Search Computing [4].

4. ANALYSIS AND DESIGN

Purpose of this chapter is to explain how a real-world search application can benefit from previously explained approaches. Here, we focus on application of ideas rather than technical details which are provided in the following chapter.

4.1 Requirements

Inclusion of multi-domain and exploratory search approaches to the search process is the initial requirement. Although, we explained these two terms thoroughly in the previous chapters, it is useful to remind them with a combined summary of their main idea: "the application should be able to search for combinational results from multiple semantic fields in an incremental fashion and should guide the user during query development" [19].

Application should also allow its user to visualize result sets in multiple and customized perspectives or views. Eligible views may depend on the semantic type of domain where the user is searching for. Visualizing results from different domains should also be possible in order to present previously chosen results that are linked to one another.

Mobility is another need we want to satisfy. Thus, application should function properly on mobile computers such as smart phones and tablet computers. These devices may have different hardware and software capabilities so application should be compatible for most kinds of devices and browsers [19].

Last but not the least, user interface should be as self-explanatory as possible since target group is end users and target environment is mobile devices. The reason why user should use the application is either he/she is not experienced enough to carry out a complex web search or he/she avoids it because of interaction difficulty / time limitation with mobile devices. Tutorial should not be required even for first time users.

4.2 Overview of the Solution

The aim of our approach is to prove a mobile version of Search Computing and demonstrate proposed solution with a practical approach. This would allow the user to get answer for their complex queries without the restriction of sitting in front of a computer. They may get answer for their search tasks on the move, easily and quickly. Furthermore, valuable location data on mobile devices is used to enrich searching experience.

"Overall picture of the application can be considered as a loop of connected domain specific searches. User searches for a result from a single domain at a loop-step, while previously chosen results from other domains may also affect the result set" [19].

First, user decides which domain his/her initial search belongs to. For example, these can be Real Estate, Car Rental, Restaurant, Hotel, Information about movie, etc. After choosing the topic, he/she chooses the way to search for, i.e. by which input he/she will search. For instance, after choosing the Restaurant topic; he/she may simply search for restaurants by Address or by its coordinates. After this step, he/she chooses the service provider, in other words, information source. In our example, after choosing Restaurants by coordinates, search may be conducted via Yelp or YQL search providers. When these steps are completed, the user is asked to complete the input 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 for the results: accordion, map view, and comparison.

Accordion view is the default visualization tool in our 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 the Map view which can be used if the results include location information. This visualization method is very useful since address information can be easily understood and make sense on a map. Location of the user is also displayed with an approximation to cellular base, even if the device does not have a GPS module. By this way, user may see the distance of results from his/her location. Pins representing the results can be clicked to display details or to choose results.

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 result, history page is automatically displayed with selected item added. In history page, all the items which are previously selected can be seen with accordion feature. Moreover, map view is also valid in this screen in order to let the user see the locations of the selected items from different domains. 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 user can jump to choose a hotel which is connected to the restaurant by location or by theme. By this way, various combinations can be created.

Same operations should be carried out at each step until the user includes all the domains to the query and is satisfied with the chosen results. As a matter of course, result set of the domain specific searches except the first one also depend on what user have selected in previous searches and in what ways the domains of these selections are connected to the current domain. In between each step, user is presented with an overview of the ongoing multi-domain search in history view. This overview is stored for later reference or modification until user discards it in order to launch a new search.

For more details; UML Use Case, Component, Statechart and Activity diagrams are presented in Appendix A.

4.3 Design of the Mobile Interface

Interface of the web application is designed following the design principles given previously. 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.

Figure 4.1: Sample tab sets from amazon.com.

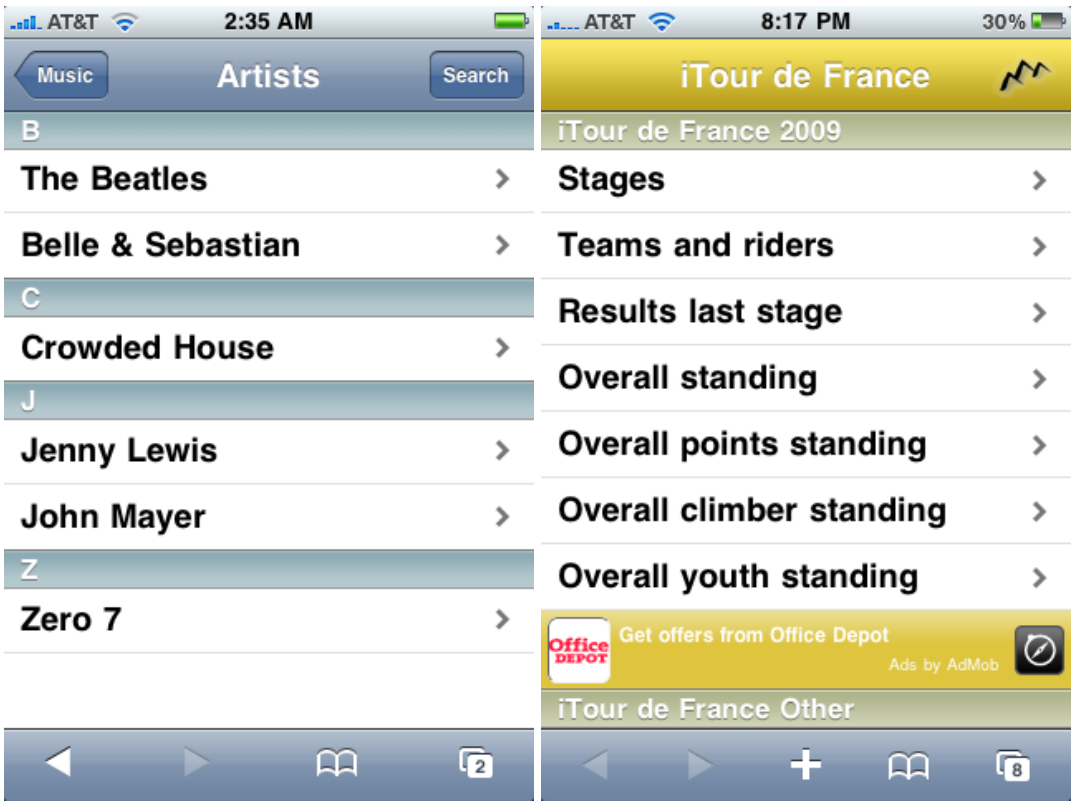
4.3.1 Tab set

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:

- It is guaranteed that the tab set will fit to user screen even in smaller mobile devices.



(a) iPhone music player screen reproduction (b) iTour de France web application

Figure 4.2: Two sample item lists from iPhone web applications

- Complexity of the user interface is decreased while the ease of use is increased.

4.3.2 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 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, for instance, while choosing a service mart for the search.

4.3.3 Accordion

Accordion is a vertical stack of elements with expandable and shrinkable bodies. Clicking, tapping or pulling 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 4.3: A sample accordion from a camp survey.

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, 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 so that only one section is open at a time.

4.3.4 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.

Figure 4.4: A sample Google Map view.

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.

4.3.5 Table

Table is an arrangement of date 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.

Show 10 entries Search:

Rendering engine	Browser	Platform(s)	Engine version	CSS grade
Gecko	Firefox 1.0	Win 98+ / OSX.2+	1.7	A
Gecko	Firefox 1.5	Win 98+ / OSX.2+	1.8	A
Gecko	Firefox 2.0	Win 98+ / OSX.2+	1.8	A
Gecko	Firefox 3.0	Win 2k+ / OSX.3+	1.9	A
Gecko	Camino 1.0	OSX.2+	1.8	A
Gecko	Camino 1.5	OSX.3+	1.8	A
Gecko	Netscape 7.2	Win 95+ / Mac OS 8.6-9.2	1.7	A
Gecko	Netscape Browser 8	Win 98SE+	1.7	A
Gecko	Netscape Navigator 9	Win 98+ / OSX.2+	1.8	A
Gecko	Mozilla 1.0	Win 95+ / OSX.1+	1	A
Rendering engine	Browser	Platform(s)	Engine version	CSS grade

Showing 1 to 10 of 57 entries

Figure 4.5: A sample table with advanced features for browser data.

(a) Date picker (b) Time picker

Figure 4.6: Some temporal form elements for iPhone.

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 numeric overload and a table that cannot fit to the mobile device screen.

4.3.6 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.

Form elements in the web application are dynamically chosen according to the data type of the input field. For example, a date input is represented by a date picker element while a regular string input is represented with a text field. Decisions for form element are explained in the implementation details.

4.3.7 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.
- Reset button is always on the right side of the header bar.

- 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.

5. IMPLEMENTATION

Web search, as the name suggests, is conventionally carried out in web pages through a browser rather than standalone applications. This allows users to visit linked result pages in the same context without switching to the browser. We follow the convention to ensure that the user searches in a natural way on their mobile browsers.

Our application is developed as a web application optimized for mobile devices and browsers instead of multiple device-specific native applications. This choice is also motivated by the new opportunities provided by HTML5 and client-side technologies such as JavaScript, AJAX, and CSS. These technologies allow web-based applications to make use of most features of mobile devices like GPS adapter or Camera. Moreover, web-based mobile applications are cross-device which redeems developers from the complexity of developing native applications for each device.

A domain specific language for developing mobile web applications called mobl [5] is chosen to speed up building the application. Mobl is a statically typed language that integrates all aspects of the application: data modeling, user interfaces, application logic, styling and web services. Mobl projects generate static, cross-device and cross-device HTML5, JavaScript, and CSS files as output.

Implementation of the web application is explained in this chapter. 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.

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. This choice resulted in:

- Testing and debugging is easier because separate layers can be tested and debugged individually.
- Reusing should be 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 APIs 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. First task of the web service 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.

5.1.2 Client-side technologies

mobl is capable of accessing ReSTful web services. Since Mart Repository and Query Processor APIs 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 [5]. 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.

```
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 [5]. Below are two example usages of the "createExecution" resource:

```
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 known, 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 [5]. A sample code snipped demonstrating the usage of "whenLoaded" control is given below.

```
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.

(a) Search tab. (b) History tab.

Figure 5.1: Tab set in the root screen of the application.

For each screen, we provide sample screen captures followed by textual explanations of interaction elements as well as programming aspects supported by code snippets.

5.2.1 Root screen

Root is the screen that must be present in all mobl web applications. It is the screen which is first generated 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:

- Title of the tab that is visible to the user
- URL to an icon to be displayed near the title
- 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 [5]. 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 5.1. 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 5.1(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 root screen.

- Group control is a container for item controls which can only exist inside a group.
- 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.

(a) Access patterns. (b) Interfaces.

Figure 5.2: Access patterns and Interfaces screens of the application.

- Item control contains an "onclick" event trigger which calls the "accessPatterns" screen and redirects the current page there. 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 is used to display the verbal description of the mart inside the body of an item.

5.2.3 Access pattern screen

After choosing the service mart, a similar list is provided to the user which is demonstrated in Figure 5.2(a). This time, group is filled with descriptions of access patterns belonging to the previously selected service mart. An array of all access patterns are received and then related ones are filtered programmatically. This is due to the limitation of Mart Repository ReST API for accessing access patterns. Because there is no direct way of getting the information about the access patterns that belong to a specific service mart.

```
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 when construct. When 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 [5]. Clicking an item calls the interfaces screen and redirects the page there with some input arguments stating the previous choices of the user, namely service mart and access pattern.

5.2.4 Interfaces screen

The user should make one last decision before accessing the search inputs 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 5.2(b)) and the source code of the screen (as it can be seen below) are very similar to the previous two screens.

Again, due to the limitation of the API, it is not possible to access information about the interfaces only if they belong to the specified access pattern from the API. Thus, they need to be sorted out using the when construct evaluating the identifiers of access patterns.

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.

(a) (b)

Figure 5.3: Form elements in the form screen of the application.

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

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 5.3, 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.

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. Table 5.1 summarizes which data types are represented with which input element under which special treatment.

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 5.4.

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:

- A string to display user as the title of the section
- A control to render as the body of the section

Second argument of the accordion control is a title to represent currently selected tab [5]. 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.

```
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 depending on the data received from web service. Each section title contains the main

Table 5.1: Mapping between data types and input elements.

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

(a) Closed accordion. (b) Open accordion.

Figure 5.4: Accordion view of the application for a news search about Turkey.

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.

As distinct from other data objects, main output was not provided by the Search Computing API's. However, it was necessary to detect it for the header of accordion rows. As a result, a temporary solution is implemented: a JSON file is generated in order to provide a main output for each domain.

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.

(a) Comparison selection screen. (b) Compare screen.

Figure 5.5: Comparison view of the application for a movie search for Batman.

```
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 screen. 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 5.5(b).

Source code given below demonstrates the table, row and column constructs of the mobl [5]. 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.

```
Table {
    list(element in comparedProperties){
        row {
            col { label(element._1) }
            col { label(element._2) }
            col { when(...) {
                button("Choose",onclick={
                    chooseResult(...);
                    currentTab="History"; history1();
                })
            }}
        }
    }
}
```

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 below, mobl can natively render a Google Map. Coordinates of the center of the map, A collection containing MapMarker objects, height, width and level of zoom are some of the arguments. In the project, markers are received by the show map screen as the input argument. Then, location of the first

(a) Show map screen. (b) Marker details screen.

Figure 5.6: Map view of the application for a real estate search by location.

(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.

```
googleMap(  coords,
            markers=markers,
            height=window.innerHeight,
            zoom=12
)
```

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.

5.2.11 History section

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

- Eplicitly anytime when the user taps the history tab.
- Implicitly and programmatically when the user taps the choose button from any of the three possible result perspectives: accordion view, map view or comparison view.

An example of history screen is shown in Figure 5.1(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.

```
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

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 [5]. Scripting code is used in three different circumstances in the project:

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

```
button("Choose",onclick={   chooseResult(...);
                            currentTab="History";
                            history1();}
)
```

- Inside a function body.

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

- Inside the script blog in a screen.

```
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".

Second script blog is checking whether two special conditions are met or not, and perform actions if they are met.

- 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.

- 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.

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.

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.

5.3.4 Result 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.

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.
- 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 functionality by means of the static methods

of the Reflector type. Main output of the interface is determined with the "getMainOutput" function.

- Pushing name of the sections and their body controls to an array called "resultAccordion". Later on, this array is used to generate the accordion control in history screen.

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.

5.3.6 Compare screen script blogs

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:

- Value of the main output, this value is taken from the "resultTupleProperties" array and used in the first column of the comparison view.
- 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.
- 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.

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.

- Iterating over the properties of the selected result and pushing them to the "historyTupleProperties" array.
- 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.
- 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.

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 that has the same title as the given mart.
- 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.

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.

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 properties of all output fields of the source interface to an array called "searchOutput".
- Find the mart id, description and access pattern id associated with target interface.
- Set search tab as the active tab and call interfaces screen inside that tab.

6. EVALUATION AND DISCUSSION

An experiment is carried out in order to comprehend whether proposed search paradigm actually ease and quicken the search process. In few words, number of interaction steps and number of characters entered are compared in two cases: using traditional web search, and using the proposed paradigm. Following sections explains the details of experiment method, results and their interpretations.

6.1 Method

Method used for the experiments contains a prior survey and two search tasks.

6.1.1 Participants

Three male and three female graduate students from Istanbul Technical University Graduate School of Science Engineering and Technology participated in the evaluation part of this thesis. Ages of the participants range from 23 to 25 (first to third year students).

6.1.2 Survey

A short, anonymous survey is conducted in order to record demographics (age and gender) and web search preferences of the participants. They are asked, via a printed form page, whether they own a smartphone or a tablet computer, how frequently they carry out a web search on a mobile device and which types of search they carry out most.

While, ownership of mobile devices are yes-no questions, other two questions have multiple options. Choices for the frequency of mobile web search are "daily", "weekly", "monthly" and "never". In the question about the type of search, participants are asked to sort three types of search from most frequently they conduct to least frequent one. These options are lookup search, learn search and investigate search

as suggested in [2]. For clarification, informal definitions of these types and some sample queries are provided as well.

6.1.3 Experimental design

Participants are asked to carry out two complex search tasks: "A real-estate for rent in the neighborhood you are in right now and a nearby job position in the IT sector" and "A good hotel and a Chinese restaurant near to Champs-Élysés, Paris". They are allowed to use any search engine and spend as much as time as they wish. Video captures of their searches are recorded for later analysis.

Selection of the tasks is not arbitrary. Both tasks require associated results from two domains: real-estate and job position for task 1 and hotel and restaurant for task 2. Multiple constraints such as neighborhood of real-estate, quality of the hotel or the cuisine of the restaurant are also included to complicate the query. Location information is required for both for association and filtering of results. First task allows the user to make use of their personal experience and local search engines, second one on the other hand, requires a more impersonal approach and global search engines.

6.1.4 Data collection

Recorded search videos are watched and analyzed multiple times in order to calculate five counts about the interactions between the user and the computer. Separate counts for each user and each task are logged. These counts are interaction steps, characters of textual inputs, domain specific search engines, general purpose search engines, and the duration of search.

Number of interaction steps is basically how many times user clicks/taps on a button/link or chooses an option. It can be considered as number of clicks/taps except accidental ones that are not considered. Interactions are counted only until the first result is found, exploring for more results or comparing them are not included as the counts can be quite subjective.

Number of characters inputted to the search systems is also counted. Only the keywords entered to the input fields are considered while the characters entered to navigate to the websites are excluded. When the user misspells and retypes something,

Table 6.1: Survey results.

ID	SP	TPC	Freq.	Lookup	Learn	Investigate
1	N	N	Never	2	1	3
2	Y	N	Daily	1	2	3
3	Y	N	Daily	1	2	3
4	N	N	Never	1	2	3
5	Y	Y	Daily	1	3	2
6	Y	N	Daily	3	2	1

only the characters of the final (possibly correct) entry are counted. Characters that are auto-completed, auto-corrected or suggested by the search system are not considered. However, if the user interacts with the system to use the suggestion or correction, this is counted as an interaction step.

Participants make use of the services of either domain specific search engines or general purpose search engines. We count how many such engines they use for their query. Sometimes, one result of a general purpose engine may be the result list of a domain specific engine. In such cases, if the user merely browse the result list and do not interact with the domain specific engine, we count only the general purpose search engine. On the other hand, if the same search engine is employed multiple times by the user for the same task, only the first use is considered.

Durations of the search tasks are also logged. However, this data is not used for formal comparison since it depends heavily on the user and connection speed.

6.2 Results

This section contains results of the survey and experiment tasks without comments.

6.2.1 Survey

Survey results are shown in Table 6.1. An anonymous identification number (ID) is assigned to each participant to allow association between survey and experiment results. SP and TPC stand for smart-phone and tablet PC ownerships, respectively. Frequency is for the usage of mobile web search. Finally, last three columns are the orders of importance and frequency of three search activities. 1 means, related participant carries out that type of search most, 3 means least.

Table 6.2: Collected data for task 1.

ID	Interactions	Characters	DSSE	GPSE	Duration
1	6	56	0	1	60
2	19	37	1	1	160
3	20	26	2	0	180
4	25	99	2	1	600
5	20	5	1	1	230
6	13	30	2	1	300
Average	17.17	42.17	1.33	0.83	255.00
Baseline	**14**	**2**	-	-	-

Table 6.3: Collected data for task 2.

ID	Interactions	Characters	DSSE	GPSE	Duration
1	4	46	0	1	50
2	8	13	1	0	110
3	10	16	1	1	300
4	9	38	0	1	300
5	11	50	1	1	150
6	9	51	0	1	150
Average	**8.50**	35.67	0.50	0.83	176.67
Baseline	13	**20**	-	-	-

6.2.2 Experiment

Data collected from task 1 and task 2 of the experiment phase are given in Table 6.2 and Table 6.3, respectively. In addition to number of interaction steps and number of input characters, DSSE stands for the number of domain specific search engines and GPSE stands for the number of general purpose search engines employed by the user. Unit for duration is seconds.

Average values for each count is also presented in the final rows of both tables. Baseline values of the tasks are explained in the next sub-section.

6.2.3 Baseline

In order to compare the result of traditional search activities, we also need same data from our application. Since we do not count values coming from errors or browsing the results, data counts are fixed while using the suggested search paradigm. All users follow the same interaction steps and enter the same input to the forms. That's why;

data for the suggested search paradigm is not included to the experiments. Instead, we provide and explain baseline data here.

One should note that, user may follow different paths by changing the order of domains to search. However, this kind of change results in the same number of interactions and character entries. Although durations of the tasks are not measured formally, they typically take less than 60 seconds.

6.2.3.1 Task 1

It takes 14 interactions and 2 character entries ("IT") to get a result for task 1. Real-estate type "for rent" is not a character entry since it is selected from a dropdown list. These steps are provided below. Each item corresponds to an interaction and textual inputs are marked *italic*.

1. Choose "Real-estate" service mart
2. Choose "Real-estate by coordinates" connection pattern
3. Choose any interface
4. Focus "Type" field
5. Choose "for rent"
6. Tap "Submit" (Coordinates are automatically filled with current location)
7. Choose a result to see details
8. Tap "Choose" button
9. Choose "Job near home" from connected searches
10. Choose any interface
11. Focus "Keyword" field and enter *"IT"*
12. Tap "Submit" (Coordinates are automatically filled with real-estate's coordinates)
13. Choose a result to see details
14. Tap "Choose" button

6.2.3.2 Task 2

Steps to get a result for task 2 are very similar to ones in task 1. Only change is in input screen. There is one step for focusing "Address" field in hotel search, and another for focussing "Cuisine" field in restaurant search instead of steps 4, 5 and 11 in task 1. So, the number of interactions is one less than task 1 (13), and the number of character entries is 20 ("Champs-Élysés" and "Chinese").

6.3 Discussion

Survey data show that owners of mobile devices use them for web search very frequently. Most popular search type is the lookup search, followed by learn search. Investigate search is the least frequent one. Since our search paradigm is not very effective for lookup search and more useful for more complex queries, search for learning can be an area where it may show its true potential and be most beneficial.

Table 6.2 demonstrate that, number of interaction steps are just slightly less than the average while number of input characters are significantly smaller. Similarly, for task 2 in Table 6.3 interactions are slightly outnumbered by the average but again, amount of textual input difference is decisive.

On a mobile device, a character entry takes nearly as much time as any other interaction due to the lack of physical keyboard or even if keyboard exists due to its small size. As a result, we can say that: although suggested search application yields more or less the same number of interactions, its smaller textual input requirement causes considerably less effort for mobile web search.

Another interesting result is about the number of search engines participants employed. When the question is familiar or local (like task 1), participants tend to use more domain specific search engines, however when it is distant (like task 2), they prefer general purpose search more. Number of unique general purpose search engines is the same because they always use the same one, namely Google. They use Google more in the second task.

This tendency to general purpose engines should be due to their unawareness about credited and useful domain specific search engines in the area of search. Our search

paradigm provides these domain specific search engines seamlessly even if the user is oblivious about the target area.

To sum up, survey and experiment results show that when the users search for learning new information and have complex questions to answer, suggested search paradigm can contribute to their search by reducing the effort to enter their query, shortening the search duration and providing goal directed search services.

7. CONCLUSIONS AND FUTURE RESEARCH

In a world where everything is heading to mobility, 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. A web-based mobile application that makes use of recent notions in web search is developed in order to suggest a course of action about how these notions can be practically applied for mobile devices.

Resulting application lets user develop a complex query which may typically contain multiple semantic fields. It guides user to customize query according to his/her needs step by step. Since user may not be aware of credited sources of information in the area of search, it lets him/her to get results from them with ease and possibly associate results with one another. It increases the usability of search in mobile devices and exploits such devices' strengths by channeling them to search process.

Experimental results agree with the objectives of the thesis. They demonstrate that users need to enter more text and follow more interaction steps with their traditional search habits in comparison to our search paradigm.

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.

- Altough we have compared the interaction and textual input counts with traditional search, no formal evaluation was carried out regarding the usability of the user interface.

- Despite the practicability and the generality of the mobile web applications, some users may prefer native applications for their mobile device. A future work may be devoted to the feasibility and realization of device specific native application versions of the web application, for example for iOS and Android.

- When the Mart Repository and Query Processor ReST APIs become accessible publicly and host location of the application is moved to the same server, data access of the application can be made though 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 cleanness.

- 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.

REFERENCES

[1] **Bozzon, A.**, **Brambilla, M. and Comai, S.**, 2010. A Characterization of the Layout Definition Problem for Web Search Results, 1st International Workshop on DATA Visualization and Integration in Enterprises and on the Web, Crete, Greece.

[2] **Marchionini, G.**, 2006. Exploratory Search: From Finding to Understanding, *Communications of the ACM*, **49(4)**, 41–46.

[3] **Bozzon, A.**, **Brambilla, M.**, **Catarci, T.**, **Ceri, S.**, **Fraternali, P. and Matera, M.**, 2011. Visualization of Multi-domain Ranked Data, **S. Ceri and M. Brambilla**, editors, Search Computing - Trends and Developments, Springer LNCS, pp.53–69.

[4] **Campi, A.**, **Ceri, S.**, **Maesani, A. and Ronchi, S.**, 2010. Designing Service Marts for Engineering Search Computing Applications, 10th International Conference on Web Engineering, Vienna, Austria.

[5] **Hemel, Z.**, mobl, `http://www.mobl-lang.org`, date received: 02.05.2011.

[6] **Kuhlthau, C.C.**, 1991. Inside the search process: Information seeking from the user's perspective, *Journal of the American Society for Information Science*, **42(5)**, 361–371.

[7] **Bozzon, A.**, **Brambilla, M.**, **Ceri, S. and Fraternali, P.**, 2010. Liquid Query: Multi-Domain Exploratory Search on the Web, 19th International World Wide Web Conference, Raleigh, North Carolina, USA.

[8] **Hearst, M.A.**, 2009. Search User Interfaces, Cambridge University Press, 1st edition.

[9] **Bozzon, A.**, **Brambilla, M.**, **Ceri, S. and Quarteroni, S.**, 2011. A Framework for Integrating, Exploring, and Searching Location-Based Web Data, *IEEE Internet Computing*, **15(6)**, 24–31.

[10] **Fu, W.T.**, **Kannampalill, T.G. and Kang, R.**, 2010. Facilitating exploratory search by model-based navigational cues, 15th International conference on Intelligent User Interfaces, Hong Kong, China.

[11] **iOS Reference Library**, 2011. iOS Human Interface Guidelines, Technical Report, Apple Inc.

[12] **Galitz, W.O.**, 2007. The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques, Wiley, 3rd edition.

[13] **Bonardi, S.**, 2010, Data Visualization Techniques for Multi-Domain Result Sets.

[14] **Kamvar, M. and Baluja, S.**, 2006. A large scale study of wireless search behavior: Google mobile search, SIGCHI conference on Human Factors in computing systems, Montreal, Quebec, Canada.

[15] **Church, K.**, **Smyth, B. and Keane, M.**, 2006. Evaluating interfaces for intelligent mobile search, International Cross-disciplinary Workshop on Web Accessibility, Edinburgh, Scotland, UK.

[16] The Search Computing Project, Politecnico di Milano, `http://www.search-computing.com`, date received: 29.04.2011.

[17] **Brambilla, M. and Ceri, S.**, 2009. Engineering Search Computing Applications: Vision and Challenges, The 7th joint meeting of the European Software Engineering Conference, Amsterdam, The Netherlands.

[18] **Masseroli, M.**, **Paton, N. and Ghisalberti, G.**, 2010. Search Computing: Integrating Ranked Data in the Life Sciences, 7th International Conference of Data Integration in the Life Sciences, Gothenburg, Sweden.

[19] **Aral, A.**, **Akin, I.Z. and Brambilla, M.**, 2012. Mobile Multi-domain Search over Structured Web Data, **S. Ceri and M. Brambilla**, editors, Search Computing - Broadening Web Search, LNCS vol. 7538, Springer, (In Print).

APPENDICES

APPENDIX A.1 : Use Case Diagram
APPENDIX A.2 : Component Diagram
APPENDIX A.3 : Statechart Diagram
APPENDIX A.4 : Activity Diagram

APPENDIX A.1

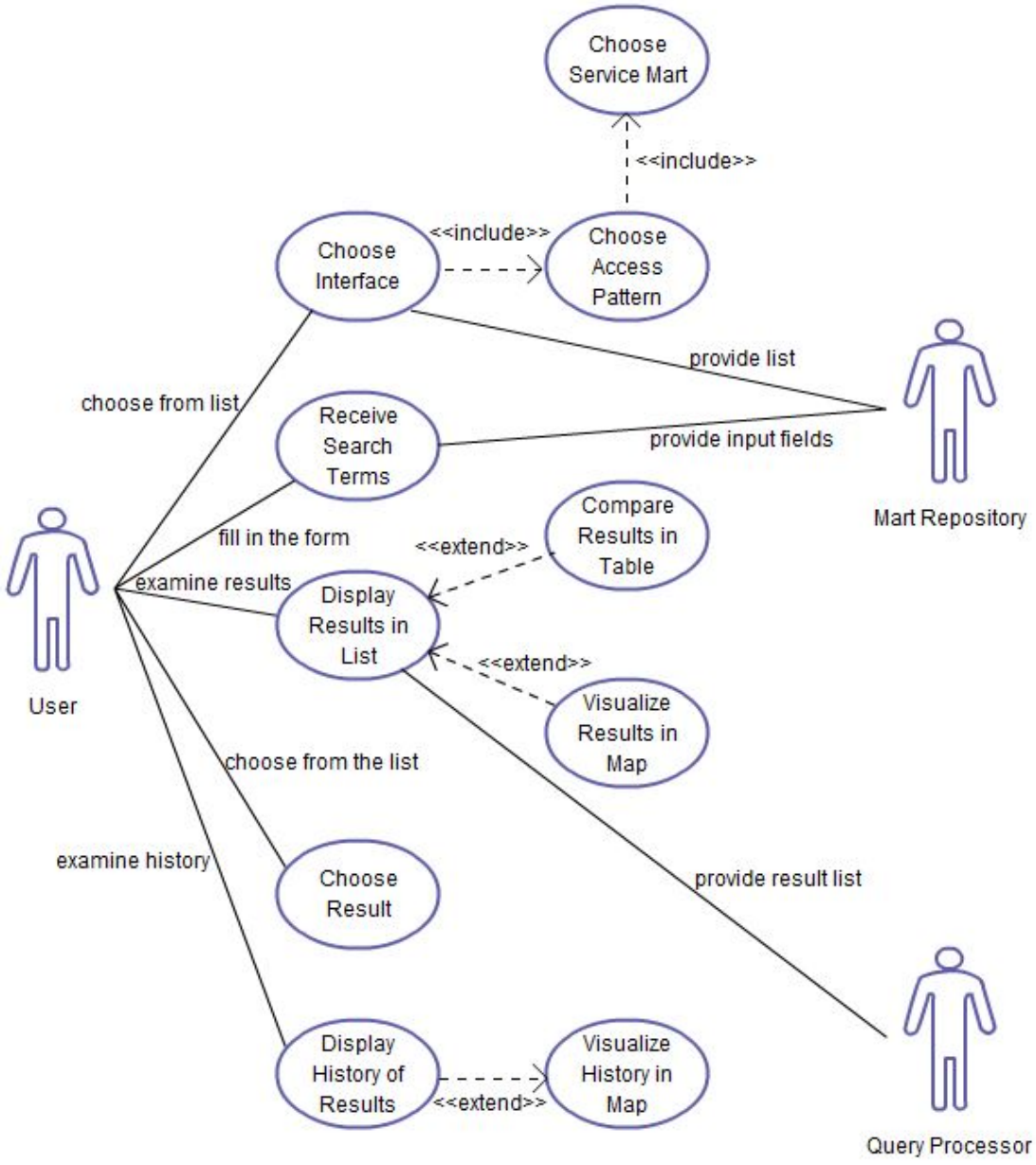


Figure A.1: Use Case Diagram of the application

APPENDIX A.2

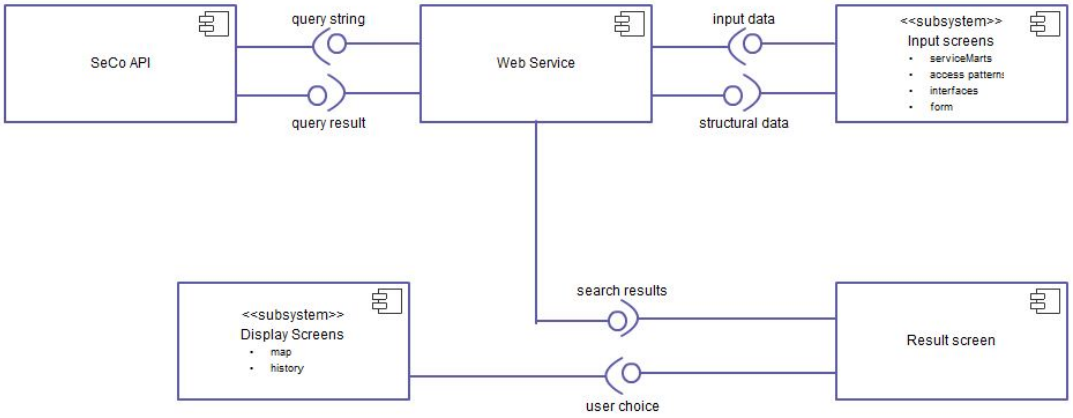


Figure A.2: Component Diagram of the application

APPENDIX A.3

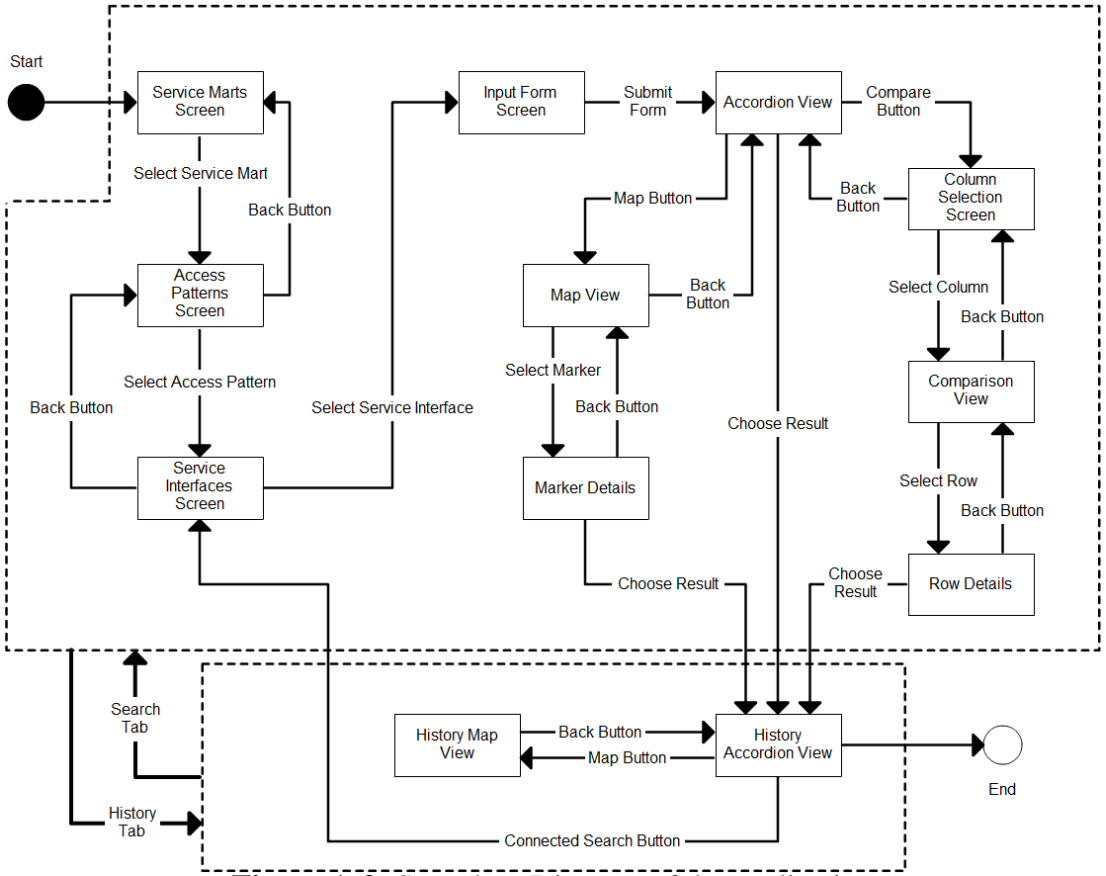


Figure A.3: Statechart Diagram of the application

APPENDIX A.4

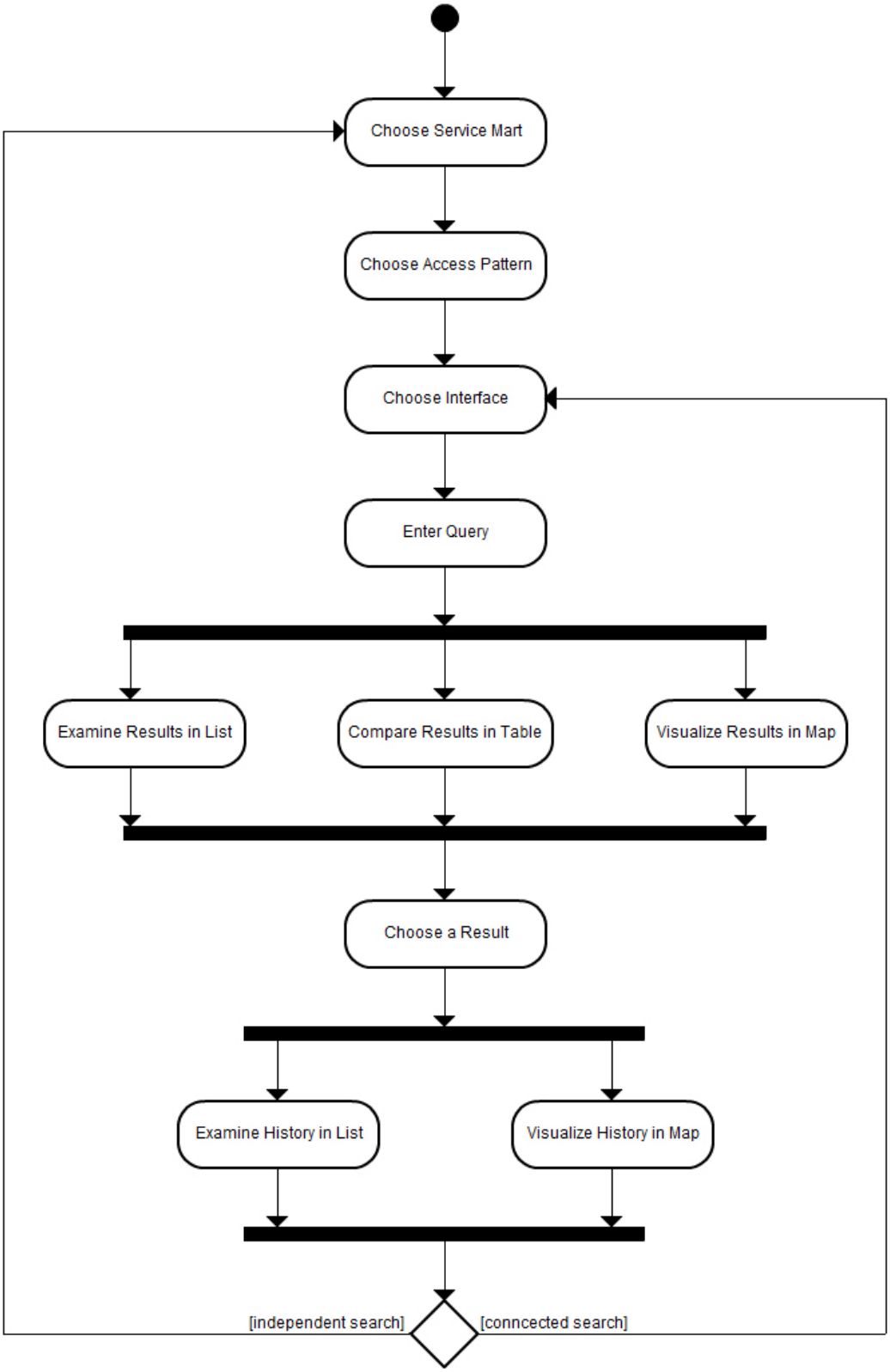


Figure A.4: Activity Diagram of the application

CURRICULUM VITAE

Name Surname: Atakan Aral

Place and Date of Birth: Istanbul, 02.04.1986

Address: Istanbul Technical University, Faculty of Computer and Informatics, Maslak, Istanbul, Turkey, 34469

E-Mail: aralat@itu.edu.tr

B.Sc.: Istanbul Technical University, Computer Engineering

M.Sc.: Politecnico di Milano, Computer Engineering

Professional Experience and Rewards:

- Scholarship of Turkish Premiership (2005-2009)
- İTÜ ARI Awards (2005-2009)
- "Diritto allo Studio Universitario" Grant by Italian Government (2010-2011)

List of Publications and Patents:

- Senliol, B., **Aral, A.** and Cataltepe, Z., 2009. Feature Selection for Collective Classification, 24th International Symposium on Computer and Information Sciences (ISCIS), Northern Cyprus.
- **Aral, A.** and Cataltepe, Z., 2012. Learning Styles for K-12 Mathematics e-Learning, 4th International Conference on Computer Supported Education (CSEDU), Porto, Portugal.

PUBLICATIONS/PRESENTATIONS ON THE THESIS

- **Aral, A.** and Akin, I. Z. and Brambilla, M., 2012. Mobile Multi-domain Search over Structured Web Data, S. Ceri and M. Brambilla, editors, Search Computing - Broadening Web Search, LNCS vol. 7538, Springer (In Print).