UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

WHAT'SUP: A MOBILE APPLICATION FOR SEARCHING ONGOING CULTURAL EVENTS

João Manuel Amaro Silva

PROJECTO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2012

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

WHAT'SUP: A MOBILE APPLICATION FOR SEARCHING ONGOING CULTURAL EVENTS

João Manuel Amaro Silva

PROJECTO

Trabalho orientado pelo Prof. Doutor Paulo Jorge Cunha Vaz Dias Urbano
e co-orientado pelo Prof. Doutor João Carlos Balsa da Silva

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2012

Acknowledgments

In first place, I must thank my Mom and Dad, Noémia Silva and Luís Amaro, for supporting me during my entire academic route, without whom I would never be able to develop this Master's Thesis and obtain my Master's degree. They always let me take my own academic decisions and gave me full support for studying and finishing my course. Then, I thank to my brother, who says he wants to become a Computer Engineer in the future, and for who I represent an example, for good things and bad things. I really love my family and I get strength and motivation from them. They always motivated me, since kid, to do my homework, study and to always want to learn more. I also give my thanks to FCT (Fundação para a Ciência e a Tecnologia) for approving this thesis project and supporting it. To Professor Paulo Urbano and João Balsa, who are my Thesis supervisor and co-supervisor, respectively. Their help was fundamental for me to proceed further in the developing of this project. They were always there when I needed help or had any doubt. They provided me scientific material to work on, such as articles, books and papers on the subject. They also coordinated all my work on this project, giving me suggestions and opinions. I thank them also for reviewing this document. I must thank specially to Professor Urbano, my Thesis Primary Supervisor, because he really is one of a kind teacher and person as well. He almost makes me forget about his status as a professor, due to his teaching methods, open and humble spirit and constant good mood. Professor João Balsa's contribution was also very significant. He helped me a lot with technical issues or theoretical questions on the artificial intelligence field and is also a very dedicated professional and excellent teacher. I also thank to my colleagues at LabMAg. Special thanks go to Edgar Montez, who was my nearest colleague on the laboratory and consequently, my "neighbour". Also, Edgar has become, above everything, a friend that I will keep for as long as possible. Then, I thank Nuno Henriques for helping me with his grat experience on the Computer Engineering field; and also to Christian Marques, João Lobo, Davide Nunes, Jorge Gomes, Fernando Silva, Rui Flores and Phil Lopes for working, discussing, socializing and laughing with me daily. They provided a good environment to work and also to relax (at the proper moments). I also thank to Marcelo Vicente and João Marques, who were my roommates for the last five years and to my university former colleagues, Paulo Ribeiro, Ivo Rodrigues and João Antunes. I will not forget of all my friends outside my academic environment, like Diogo Fortes, Ana Filipa, Daniel Silva, Nilda Duarte and a never-ending list of people that like me. Thanks also to all whose names were not mentioned but that supported me.

To dear Mom and to the world

Resumo

Hoje em dia, não existem muitas aplicações móveis de turismo em Portugal, orientadas para a organização e calendarização de eventos culturais. Para além disso, a área do turismo doméstico (turismo feito dentro do próprio país por residentes desse país) em Portugal tem muito potencial e, face à actual situação do país, pode representar uma alternativa fiável e financeiramente mais agradável. De qualquer forma, a aplicação que se pretende desenvolver não tem como alvo o turismo tradicional (visita de museus, monumentos e espaços verdes) mais orientado ao local, em que se assiste a eventos que são previsíveis em termos de calendarização e que acontecem regularmente ao longo do tempo, sempre nesses locais. Em vez disso, apostamos na descoberta de eventos dentro do ambito doo turismo urbano, de curta duração e num contexto mais ad-hoc (sem um planeamento prévio muito aprofundado ou sem um conhecimento prévio do local onde nos encontramos), tendo maioritariamente como objectivo o entretenimento, como por exemplo a participação em festivais, concertos, eventos de cinema, festas, exposições, entre outros. O nosso foco é, portanto, um turismo mais orientado aos eventos, mais dinâmico. O nosso utilizador-alvo procura idealmente eventos que estejam a decorrer nesse momento ou que decorram no próprio dia. Também é importante salientar que decidimos utilizar ao máximo as tecnologias da Web Semântica que têm emergido nos últimos anos. Este factor mostra que a nossa aplicação é baseada numa abordagem inovadora e que tem grandes potencialidades dentro da área.

Deste modo, este projecto, chamado **What'sUp** tem como objectivo o desenvolvimento de uma aplicação móvel, destinada a funcionar em dispositivos Android, que possa indicar ao utilizador eventos culturais existentes no local onde este se encontra (e nas redondezas), que decorram num dado momento ou intervalo de tempo. Estes eventos são apresentados ao utilizador através da análise da linguagem natural que é introduzida pelo mesmo, quando este procura por eventos na sua área circundante. O utilizador pode colocar perguntas à aplicação (através de introdução de texto) do tipo ”O que há de teatro agora?”, ”Que concertos vai haver hoje à noite?”ou mesmo ”Que exposições posso visitar às 10 horas”. A aplicação analisa a pergunta do utilizador, através de um sistema de palavras-chave e tranforma-a numa *query* na linguagem de pesquisa SPARQL, que é executada sobre uma ontologia, que serve como base de dados da nossa aplicação. Esta ontologia de eventos culturais, escrita na linguagem OWL 2 (baseada em RDF), permite-

nos representar a informação dos eventos com um detalhe muito elevado. O resultado da *query* é a lista de eventos culturais, correspondente à pergunta do utilizador. No entanto, para que este processo se concretize, há uma série de tarefas que são executadas antes disso. Quando a informação sobre os eventos é armazenada na ontologia, já sofreu um conjunto de alterações e passou por várias fases: primeiramente, os eventos são extraídos de um conjunto predefinido de sítios Web apropriados (com informação sobre eventos culturais em Lisboa e arredores) existentes na Internet, utilizando web scrapers para tal. Estes web scrapers possuem uma grande flexibilidade, já que os web sites de onde a informação é extraída estão estruturados de formas diferentes e mostram informações diferentes sobre os eventos culturais. Por isso mesmo, é necessário adaptar os web scrapers para que se enquadrem com a estrutura de navegação de cada site, com o objectivo de extrair a informação correctamente em cada caso. É extraído conteúdo digital (em forma de texto) relevante sobre cada evento publicado nos documentos HTML das agendas on-line, previamente seleccionadas. Essa informação é guardada num ficheiro do tipo CSV (Comma-Separated Values). O ficheiro é lido por uma aplicação Java, que faz ligeiras alterações às expressões (sejam elas sobre a data, categoria ou preço do evento) e executa sobre cada expressão extraída, a função apropriada da gramática construída em Prolog, baseada no conceito de gramática livre de contexto, também chamada de gramática de cláusulas definidas. Esta gramática é constituída por um conjunto de regras que permite avaliar expressões de linguagem natural com certas caracterídticas. Assim, são extraídas as características relevantes das expressões. As propriedades do evento são definidas através do tipo de retorno que estas funções devolvem. De seguida são criadas instâncias desses eventos na ontologia, com as suas respectivas características, como o nome do evento, a sua categoria, data, preço e local. Isto é concretizado através do uso da framework para Java, Jena API, que nos permite editar a ontologia directamente (sem aceder a uma ferramenta de edição de ontologias) e ter controlo sobre o seu conteúdo. Todos estes elementos formam uma base de dados de eventos devidamente catalogados e organizados, que pode ser pesquisada, devolvendo os resultados esperados, depois do utilizador fazer a sua pesquisa, através de um *input* de texto na aplicação. A localização do utilizador é sempre tida em conta, através dos dados da sua geolocalização, retirados do dispositivo móvel (por exemplo, do sistema de Global Positioning System, conhecido por GPS). Com estes dados, a aplicação consegue calcular quais os eventos que decorrem em locais que se encontram mais perto do utilizador e apresentar essa informação.

Assim, a aplicação devolve um resultado ou conjunto de resultados que correspondam à pesquisa do utilizador e permite ao utilizador aceder a toda a informação sobre cada evento, incluindo a sua geolocalização no mapa da aplicação e a distância a cada evento.

O utilizador faz as suas pesquisas através de linguagem natural, o que é vantajoso para o próprio. Esta abordagem permite ao utilizador evitar uma pesquisa por parâmetros e demasiado complicada, que exigiria da sua parte uma maior carga cognitiva, em termos

de utilização da aplicação e também em termos de conhecimento dos principais locais e atractivos turísticos da zona onde se encontra. Deste modo, o utilizador não necessita de saber o nome do evento nem o local onde este acontece para o encontrar. Apenas tem que fazer uma pergunta que o leve a obter os resultados para descobrir que eventos estão a decorrer no momento.

Para que a informação sobre eventos seja renovada, os web scrapers são executados periodicamente para actualizar a base de dados de eventos. Este processo é, portanto, automático e invisível para o utilizador.

A arquitectura da aplicação está dividida em vários módulos, tendo cada um a sua função e sendo fundamental para o funcionamento de todo o processo. Procurámos ter uma arquitectura modular, em que cada módulo é relativamente independente e, consequentemente, pode ser estendido individualmente quando for necessário. A dissecação da arquitectura é feita na secção do trabalho realizado.

A interface final da aplicação tem um estilo simples e minimalista, onde, num primeiro nível, o utilizador faz um *input* de texto e depois explora os resultados da sua pesquisa. O utilizador pode ainda alterar algumas definições na sua pesquisa, a partir da aplicação, como por exemplo, o raio geográfico da procura (500 metros, 2000 metros, etc). Desta forma, este projecto visa oferecer alternativas para os turistas ocasionais, que queiram descobrir a cultura da cidade. Para isso, esta aplicação fornece toda a informação de que estes necessitam para se orientarem nas suas aventuras turísticas e culturais.

Palavras-chave: web semântica, linguagem natural, ontologia, evento cultural, gramática

Abstract

What'sUp is a project which focuses on the development of a tourism and culture oriented context-based mobile application that helps the user to explore local cultural offers. The goal is to inform and to lead the user about the ongoing and upcoming cultural events in the user geographical area, without the need to specifically state his geographic location or the exact date/time of the intended event. We built a Web ontology in OWL 2, which contains four types of information about the cultural events: *When*, *What*, *Where* and *How Much*. This event data is scattered across on-line agendas and e-magazines on the Web. We periodically extract the most important content about the events, to keep our event information updated. This information is then refined and inserted in the web ontology, using web scraping methods and a Definite Clause Grammar. Each user question is made in natural language (through text input) and is after transformed in a SPARQL query that runs in the OWL 2 database. The result of that query is a list of cultural events, that is then presented to the user. What'sUp can greatly improve the access to context-based information in digital cities, through the use of natural language interaction and Semantic Web technologies.

Keywords: semantic web, mobile application, grammar, ontology, natural language

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We are witnessing a widespread interest in mobile recommender systems for tourists and city inhabitants, that tailor their content based on the user's current location, personal profile, activity, weather conditions or other contextual factors [1, 2, 3]. Mobile context-aware recommendation systems are the natural outcome of the advances in positioning technologies and the increased sensing capabilities of smartphones. A central issue in mobile recommendation systems is the exclusive exposition to filtered content that matches user profiles, the filter bubble [4], preventing the access to novel and serendipitous content [5]. The support of serendipitous discovery of interesting events in an urban environment will certainly help urbanites that emphasize spontaneity.

Semantic Web is a promising technology for organizing, sharing, integrating and searching distributed information about city cultural events, enabling machine processing of the massive amount of event information available on the Web. Web Ontologies will certainly play a fundamental role, structuring web documents with a semantic vocabulary (understandable by machines) that may be shared by different cultural event providers [6]. Currently, most information about city events is scattered across event providers websites and on-line cultural agendas. Traditional web information is targeted towards humans, but must be extracted and converted to the ontologies vocabularies, in order to be manageable by machines. Ontologies and ontology-based information retrieval have the potential to significantly improve the process of searching information on the Web [7]. The scenario of an universal and unambiguous adoption of Semantic Web technologies is still unreal. In wich concerns the e-Tourism area, that goal will only be fulfilled if cultural event providers adopt Semantic Web standards, so that information can be distributed and gathered without relying on web scraping technologies.

What'sUp is a mobile recommendation system for supporting serendipitous discovery of events in an urban context. It is a Semantic Web mobile application that guides the user throughout the city activities, making use of contextual data, location and time. We built a web ontology in OWL 2.0 [8], for representing cultural events in a city. The content is periodically extracted, refined and updated from Lecool website, a free weekly e-magazine, featuring a selection of cultural events and leisure activities in Lisbon, as well as MyGuide, a website about cultural events occurring in Portugal. In order to extract events information from LeCool e-magazine and Myguide website, we rely on web scraping and natural language processing with Definite Clause Grammars (DCGs) to deal with the events' temporal aspects. Users can discover what is happening "here and now" by making a question on the mobile application interface, using natural language. For example, the users can type in the smartphone "What is up now?" or "What is up today at noon?" and the natural language queries are transformed into corresponding SPARQL queries [9], using a keyword matching process. The resulting events are sorted by proximity to the mobile device's GPS position. This means that the results on the top of the list are the nearest ones to the user. The user can restrict the search by choosing the geographical search radius around him. The event information is enriched using the Google Maps service, providing the event location on the map and suggesting an itinerary.

In the Semantic Web environment, there are several languages and standards that make different approaches to provide a way of including semantic data in a website or in a document including knowledge. One of these approaches is Schema.org, which is analysed in Section 3.2.2. These approaches represent alternatives to the user and, despite having the same common objective, they have different syntaxes and substantially different features that could lead the user to choose the best approach for his case. Only when one of these main standards is universally (at least at the touristic level) and unambiguously used, together with precise indications of the vocabularies that are used (using unique namespaces), and having an external agreement on the meaning of annotations and concepts, we will have all the conditions for a real touristic Semantic Web scenario.

Mobile devices are increasingly equipped with more and more powerful software, hardware and data connectivity. Most smartphones have built-in GPS receivers, available for providing position data to applications. There are obvious advantages in the applications that make use of the GPS system. Tourism has, therefore benefited from this fact, because travellers often carry their phones during vacation. On the other hand, the richness of information available on-line, in great amounts, has empowered tourists to exploit the Internet for searching travel and culture-related information. In the last decade, we witnessed a significant increase in the performance of mobile devices. In parallel, the growth of on-line data and services available on the Web make it become more and more emerging as a gold mine for the tourist guide applications.

In this chapter, we will present the motivation, objectives and context of the project. Finally, we present the document structure.

1.1 Motivation

As a touristic and cultural target, Portugal is characterized as a small territorial space. However, it possesses a very diverse historical, cultural, landscaping, climate and human content which enriches it [10].

As it is referred in Salvador Lima's article [15] , the appearance of the World Wide Web (WWW) made available massive amounts of data, although that data is prepared and linked in a way that is not suitable for automation, integration, interoperability or even context-oriented search. That is why the Semantic Web exists, to promote global information integration and semantic interoperability, through the use of meta-data, ontologies and inference mechanisms.

The motivation for this project comes with the idea of making an application for mobile devices with the capability of providing useful information about the ongoing and upcoming events without much cognitive effort from the user side. We found very attractive the idea of making it simpler to the user (possibly a tourist) to explore the cultural panorama of a certain place, at a certain time. The user does not need to manually search for these events, as the information is regularly (daily or weekly) extracted and updated, in a transparent way to the user. Nevertheless, the challenge to aggregate information from multiple web sources (that are dispersed in the WWW) in a single database is considerably motivating, due to the complex unification process of the data. As it is shown in Martin Hepp, Katharina Siorpaes and Daniel Bachlechner work [11], most tourism portals, agendas and websites contain insufficient information in various categories. Also, the information is disposed in a different way for each website and to navigate through the information is a task that requires some cognitive charge about the structure of each website. Plus, there is few work done in this subject in the Portuguese language, starting by Portuguese language ready ontologies, Portuguese natural language evaluation and Portuguese language web scraping on the tourism area. Besides, there is not an application that aggregates such elements (scraping, analysis, labelling, storage and retrieval of cultural events information) that is oriented to the Portuguese community. Even semantic applications for the tourism area are rare at the moment in Portugal. All these factors motivate us to work on this subject and to contribute to its development.

1.2 Objectives

The objectives for this project are based on the development of an application for mobile devices, where the user can have access to all the currently showing events in the surrounding zone. The application is oriented to the Portuguese language and it considers the Lisbon area and its nearest surroundings. The domain of this project is e-tourism. However, there is a more restricted domain in which the application focuses more significantly: not the e-tourism where the user handles his accommodation issues or chooses an hotel for staying at, or even registers for a flight. Instead, this application focuses on the immediate and on-the-fly tourism style, where the user is not previously prepared to a visit or participation in an event. Therefore, there are some premisses that we assume. Our project is more useful for tourists visiting a city for the first time. The tourist may not know the city cultural spots and may not be searching for any specific kind of event, just wanting to know which type of events are to occur, when and where they will occur. We expect the user to ask about events occurring on the same day of the search. Despite our application is ready for a search by event category only, this type of searching does not take advantages of all our searching features and capabilities. For that purpose, the application is able to extract automatically the useful information and provide it to the user, with efficiency. The main objectives of this project are listed below:

- To produce a set of web scrapers that automatically extract the event information from heterogeneous sources;
- To develop a grammar that manages to process and evaluate a relevant subset of natural language with certain characteristics (keywords, syntactic structure);
- To construct a homogeneous database of full event information that can be modified and queried;
- To bring all these features together in a smart and modular architecture;
- Design the application to run in mobile devices, as the e-tourism area requires;
- Ultimately, provide a simpler and useful way to know when and where the main cultural events will happen without having to know the city characteristics in first place.

1.3 Contribution

This work combines a set of technologies, from diverse areas of computing. The major contribution that our work has made is the proposal of a more practical way to search for cultural events on a certain time, being this natural language search. An important contribution is also the integration of Semantic Web technologies, like an ontology and a semantic query language, to store our event data, process it and return it to the user. This approach reveals our will to make a proof of concept for these emerging technologies, despite the obstacles that are yet to surpass. We built a web ontology as our application database and the feasibility of this solution is not taken in question as its performance is very satisfiable. Despite that a great number of mobile applications within the tourism area use other technologies for the database implementation, we found this solution very accountable. Therefore, we expect to encourage the use of Semantic Technologies in mobile applications. The fact that our application is event-oriented is also a strong contribution, in a way that most tourism applications are location-oriented, based on places and attractions that are not so dynamic, in concern to date. We give special attention to the date of the events and our application content is refreshed according to the present date. We also make an effort on being autonomous when it comes to the digital content extraction and treatment. Many applications have a strong dependency on the web providers and the content management is done by those providers, adding one more intervener on the process. We managed to do all our content management ourselves.

Furthermore, our work resulted in an article publication on the INFORUM 2012 conference [12], as it is referred in Section 7.1.

1.4 Document Structure

This document is organized in the following way: in the next chapter (Chapter 2), we explain some concepts that must be clarified to understand the scientific and computing environment in which we are working. These concepts are very important in this area of computer science. Following, on the Related Work chapter, we show what has been done lately in the area (in context-aware and Semantic Web systems). Each Related Work project was taken as an example or inspiration to our work. Some of these works are state-of-the-art. Next, we talk about the software development decisions and present the requirement analysis material. We also have the planning of the project and some general considerations about the completed work and explore its practical possibilities and viability. In the System Overview section, we describe the completed work: the extraction of data and its treatment, the ontology development and the grammar functioning details. Also on this chapter, we explain the mobile application interface and finally, the future work. The document closes with some considerations about our work. In this last chapter, we also present some conclusions that we made about all the development process, the results that we obtained and the project in general.

Chapter 2

Important Concepts

There are some important concept in this area of computer science that must be introduced and more deeply explained, as they have a fundamental role on the field and are a very important help to understand this project.

2.1 Smartphone

Our application is developed for smartphones, due to its dynamic context and “just-in-time” conceptualization. These devices, also named hand-held devices, are different from cellphones and PDA (short for Personal Digital Assistant) devices. Years ago, cellular phones, also called cellphones or mobile phones, served one purpose only: to send and receive voice communications and SMS communication. However, these devices evolved along the years and suffered multiple changes. Today, mobile phones are equipped with customized software, Internet access, digital cameras, portable music players, GPS functions and many more options. The mobile phones that still have very limited functionalities are currently called feature phones. Feature phones have proprietary operating system (OS) firmware. If they support third-party software, it is only via a relatively limited platform such as Java or BREW [1]. As of PDA devices , they are categorized as small hand-held devices that combine computing, telephone/fax, Internet and networking features. A typical PDA can function as a cellular phone, fax sender, Web browser and personal organizer. These devices are usually pen-based, which requires the use of a stylus rather than a keyboard for input. There is no industry standard for what defines a smartphone, so any mobile device that has more than basic cellphone capabilities can actually be filed under the smartphone category of devices. However, a smartphone may be considered as the combination of the traditional PDA and cellular phone, with a bigger focus on the cellular phone part. Most hand-held devices can also be equipped with WI-FI, Bluetooth and GPS capabilities that can allow connections to the Internet and other Bluetooth capable devices such as an automobile or a microphone headset. There are certain features

[1]http://www.brewmp.com/about

that smartphones usually have, unlike the previous two device types. The most obvious difference between smarthpones and other mobile devices is that smartphones have a Mobile Operating System, rather than an embedded system. The user interacts with the smartphone using a touchscreen. Lately, smartphones commonly have software related with social networks, like Facebook or Twitter, features that we do not see in PDAs and cellphones. Also, feature phones usually have a hardware keyboard, instead of a software keyboard. Mobile devices (and specifically smartphones) have limited interactivity due to the inconvenient keyboard and small screen size, compared to personal computers. This fact is mitigated by tablet computers, that have bigger screens and the same features of smartphones. Tablets have always a touchscreen interface. Our application is tablet compatible, too. All the icons and menus in the interface are auto adjustable to the screen size. Therefore, the goal of our application is to develop a smart system helping tourists find relevant cultural information with minimum effort.

2.2 Semantic Web

As we know, in the last years, in the World Wide Web, semantic content is accessible to humans but is not easily accessible by computers, due to the lack of mechanisms to understand and link semantic concepts existing in the Web infrastructures. In 2001, this concept was created by Tim Berners Lee, James Hendler and Ora Lassila [13], as an extension to the current web in which information is given a well defined meaning, better enabling computers and people to work in cooperation. According to Allemang [14], the main idea of the Semantic Web is to support a distributed Web at the level of the data rather than at the level of the presentation. Instead of having one web page pointing to another, like it happens in the HTML web infrastructure (the markup method consists in rendering information and having hyperlinks to related content, ignoring semantic connections), one single data item in a web page can point to another, using global references called URI. So, information about a single entity can be distributed over the Web. Semantic Web connects data semantically, not only syntactically. The Semantic Web architecture was defined also by Berners-Lee (as depicted on Figure 2.1).

The Semantic Web stack shows us the hierarchy of languages, where each layer exploits and uses capabilities of the layers below. It shows how technologies that are standardized for Semantic Web are organized to make the Semantic Web possible.

As we know, the Semantic Web development is increasing day-by-day. It will greatly improve the modus-operandi of the users, when searching or researching in the web. Also, like it is suggested in Lima's article [15], it permits to develop more accurate, useful and modular software tools, that can be based on the semantic interoperability, and hence, can communicate in a much suitable way, through metadata and ontologies. It adds web resources not only a meaning but also a more efficient manner of finding them, labelling

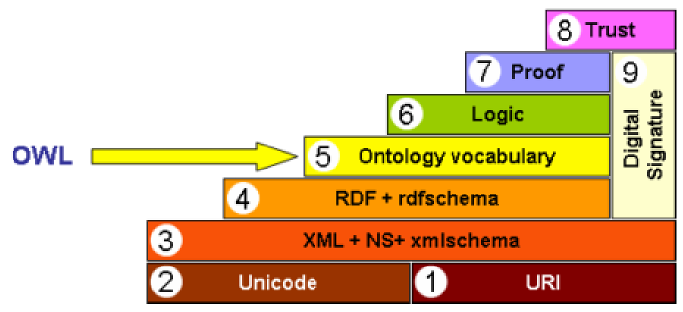


Figure 2.1: The Semantic Web layered architecture (Berners-Lee, Hendler et al. 2001)

and linking them in a new way. As referred in Dean Allemang and Jim Hendler's book [14], the RDF data model is the basis of the Semantic Web. As depicted on Figure 2.2, it permits to define triples, i.e., expressions in the form <subject><predicate><object>. The subject denotes the resource (by an URI), and the predicate (also represented by an URI) denotes traits or aspects of the resource and expresses a relationship between the subject and the object [16]. The object is either a literal or another resource represented by an URI.

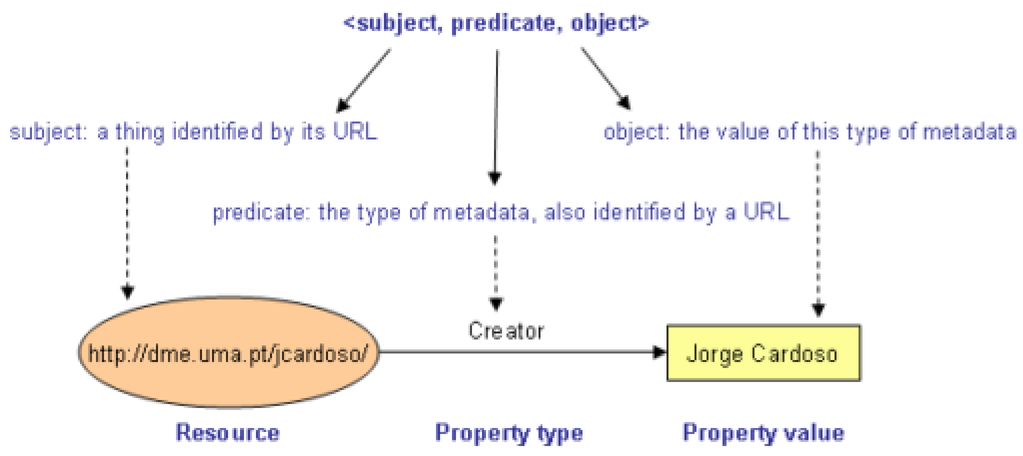

<subject, predicate, object>
subject: a thing identified by its URL
object: the value of this type of metadata
predicate: the type of metadata, also identified by a URL
http://dme.uma.pt/jcardoso/
Creator
Jorge Cardoso
Resource
Property type
Property value

Figure 2.2: An RDF statement (José Cardoso, 2006)

The object and subjects are defined as Classes, and the relationships between classes are called Properties. An element that is part of a Class is an Individual, also called an instance. Any of these three elements (Class, Property or Individual) in the Semantic Web environment, is called a Resource. In an RDF file, each of this resources can be uniquely identified using an URI, a formatted string that serves as a global identifier, from which the URL is a special case. This resolves the identity problem when two ontology classes have the same name, but do not have the same semantic context and the same attributes. Once more, the communication between the different software applications and web agents is privileged with this solution [16].

2.3 Web Scraper

Web scraping (also called web harvesting or web data extraction) is a computer software technique of extracting information from websites. Usually, such software programs simulate human exploration of the World Wide Web by either implementing low-level Hypertext Transfer Protocol (HTTP), or embedding a fully-fledged web browser, such as Internet Explorer or Mozilla Firefox.

Web scraping is closely related to web indexing, which indexes information on the web using a bot and is a universal technique adopted by most search engines. In contrast, web scraping focuses more on the transformation of unstructured data on the web, typically in HTML format, into structured data that can be stored and analysed in a central local database or spreadsheet (for instance, .CSV). Web scraping is also related to web automation, which simulates human browsing using computer software. The use cases of web scraping are, among others, online price comparison, weather data monitoring, website change detection, research, web mash-up and web data integration. In our project, web scraping will be used regularly to maintain and update the cultural event database, in other words, the OWL 2 ontology. There are many software tools available that can be used for customizing web-scraping solutions. These tools may attempt to automatically recognize the data structure of a page or provide a recording interface that removes the necessity to manually write web-scraping code, or some scripting functions that can be used for extract and transform content, and database interfaces that can store the scraped data in local databases.

2.4 Web Ontology

In order to understand what terms mean and what are the relationships between those terms, the computer has to have documents that describe all the terms and logic to make the necessary connections. A Web Ontology is a formal domain model that is linked to another in the web. The domain model of an ontology can be taken as a unifying structure for giving information in a common representation and semantics. An ontology comprises the classes of entities, relations between entities and the axioms , i.e., declarative statements, which apply to the entities of that domain [17]. They must be intelligible to both computers and humans. Ontologies are used in various domains such as knowledge discovery, natural language processing and cooperative information systems. Ontologies can be built using multiple ontology languages. An ontology language is a formal language used for encoding the ontologies. There is a large number of such languages for ontologies, both proprietary and standard-based such as common Algebraic specification language, common logic, CycL, DOGMA, Gellish, IDEF5, KIF, RIF, OWL and OWL 2. Moreover, ontology languages organize themselves in essentially three subgroups, according to its characteristics [18]:Logical Languages, Frame based Languages and Graph based Languages.

Ontology languages can be classified with more detail as follows:

- 1) Logical Languages
  - First order predicate logic
  - Rule based logic
  - Description logic
- 2) Frame based Languages
  - Similar to relational databases
- 3) Graph based Languages
  - Semantic network
  - Analogy with the web is rationale for the semantic web

In order to achieve its objective, an ontology must provide a shared and common understanding of a domain that can be communicated across people and applications.

In this project, we chose to use OWL 2 as our ontology language, in order to develop our ontology. The OWL 2 language is included in the Description Logic subgroup, being also a Logical Language. Ontologies can also be queried. We use SPARQL as our OWL ontology query language. SPARQL is the W3C recommendation as a query language for RDF [19].

2.5 Definite Clause Grammar and Context-free Grammar

The Prolog programming language offers a special notation for defining grammars, namely Definite Clause Grammars [2]. They are called Definite Clause Grammars because they represent a grammar as a set of definite clauses in first-order logic. With DCG, we are able to express Context-free Grammars and also Context-sensitive Grammars. In our case, we'll use DCG for expressing a Context-free Grammar. A Context-free grammar is a very powerful mechanism and can handle most syntactic aspects of natural languages (like English and Portuguese). Obviously, it cannot handle all the possible forms of natural language as a language is constantly evolving and changing, not only syntactically speaking but

[2]Definition on http://www.cs.sunysb.edu/ warren/xsbbook/node10.html

also semantically. This means that the syntax of natural language can be unpredictable and very diverse, if we consider all the new aspects and trends that slowly shape this language. The formalism of context-free grammars was developed in the mid-1950s by Noam Chomsky [20], and also their classification as a special type of formal grammar (which he called phrase-structure grammars).

It is a formal grammar in which every production rule is of the form

$$V \rightarrow w$$

where V is a single non-terminal symbol, and w is a string of terminals and/or non-terminals (w can be empty). As an example of a CFG, we have:

```
s → np vp
np → det n
vp → v np
vp → v
det → the
det → a
n → man
n → woman
v → shoots
```

The → symbol is used for defining the rules. The symbols `s`, `np`, `vp`, `det`, `n` and `v` are called the non-terminal symbols. The non-terminal symbols in this grammar have a traditional meaning in linguistics:

np:	noun phrase
vp:	verb phrase
det:	determiner
n:	noun
v:	verb
s:	sentence

The symbols that appear after the → symbol are the terminal symbols: the, a, man, woman, shoots.

This grammar contains nine context free rules. A context free rule consists of: A single non-terminal symbol followed by →, followed by a finite sequence of terminal or non-terminal symbols.

Translating this grammar in a Definite Clause Grammar (Prolog environment) would be like:

```
s --> np, vp.
```

```
np --> det, n.
vp --> v, np.
vp --> v.
det --> [the].
det --> [a].
n --> [man].
n --> [woman].
v --> [shoots].
```

Essentially, this breaking down of words helps a computer to process the different syntactic elements of a sentence and following simple syntax rules of the grammar that was created.

So if we consider the following sentence:

```
the woman shoots a man
```

we would have a parse tree, this is, a tree representing the syntactic structure of the sentence, as it is shown on Figure 2.3

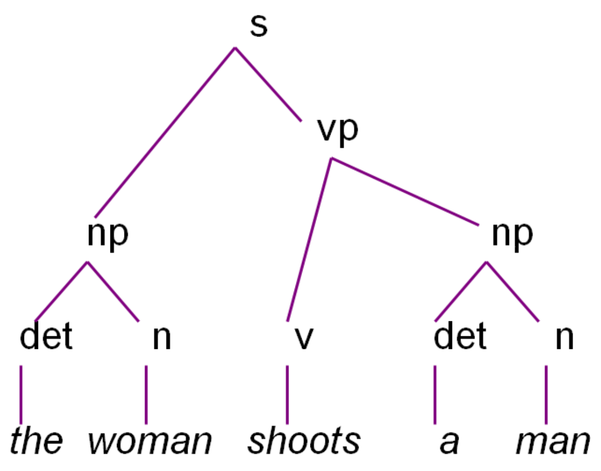


Figure 2.3: The sentence syntactic structure representation, using a parse tree (Patrick Blackburn et al)

Parse trees are very important. They give us information about the sentence and its structure. We are able to see that our example sentence is made by a set of different grammatical elements (two determinants, two nouns and one verb).

Chapter 3

Related work

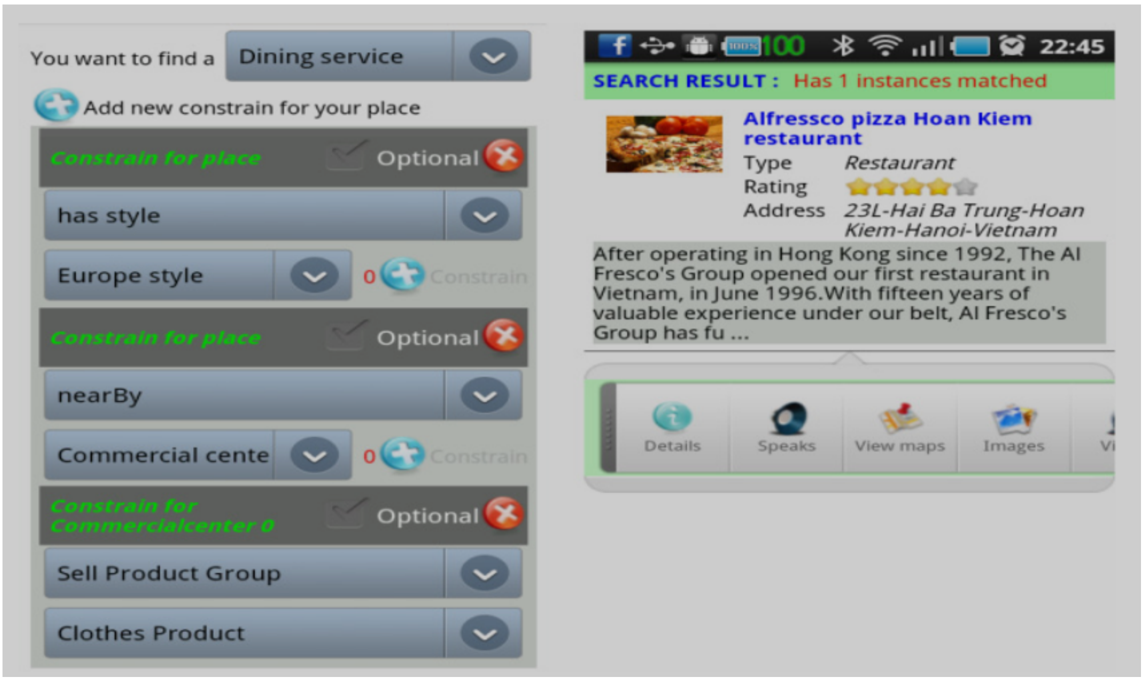
In this chapter, the state-of-the-art on this project domain will be presented, as well as the latest projects and ideas that have come up in the last years.

3.1 Comparable Systems

There are already some initiatives and projects related to the subject handled by this project. In our search, we found that some projects have modules that must be a reference and were used as inspiration to our developing process. Some ideas are fundamental for a coherent architecture, while others allow better efficiency to the application.

3.1.1 STAAR system

On Tuan-Dung Cao, Thanh-Hien Phan and Anh-Duc Nguyen work [21], a system called STAAR (Semantic Tourist Information Access and Recommending System) is presented. Helping a tourist who visits a travel destination for the first time is the main aim of STAAR, by giving advices about which places to take a look and even the best itinerary from a starting point. STAAR exploits the advantages of semantic web technology by providing tourist guidance through Web and smartphone devices. Ontologies play an important role in STAAR for representing knowledge about touristic resources, including also personal touristic interests (points of interest, activities, events and services). Ontologies can model (i) what a traveller can see and visit at a given destination; (ii) the location of interesting places and what their attracting characteristics are; and (iii) what are the relations between travel resources. STAAR helps tourist search information by providing a various semantic search feature in a mobile phone application. In addition, the team proposes an algorithm for recommending travel route relevant to both criterion: itinerary length and user interest. This system is an ontology based approach to provide the relevant information to tourists with different personal interests, what is still a challenging task for tourist guide information systems. The ontology was used for modelling all the background data and knowledge stored inside STAAR system, including



After operating in Hong Kong since 1992, The Al Fresco's Group opened our first restaurant in Vietnam, in June 1996.With fifteen years of valuable experience under our belt, Al Fresco's Group has fu ...

Figure 3.1: Search with constraints on multiple objects (Cao, T.-D. et al. 2011)

descriptions about tourism resources and services. It was built according to the method METHONTOLOGY [22], it began by exploring the field of the tourism and related fields, from which determine main coverage concepts of the ontology such as place, activity and topic.

To facilitate the integration of semantic data from multiple sources distributed on the Internet, their ontology is designed in standard language RDF++ (an extension version of RDF), which supports the OWL property `owl:sameAs`. This property allows referencing travel resources managed by the system to external resources in Linked Data repositories such as DBPedia, GeoNames. Besides displaying information of a semantic search, STAAR mobile application uses text to speech technology to help users get information conveniently with just a touch while they are moving. The team managed to implement semantic search feature in different levels of complexity. Figure 3.1 illustrates its advantages regarding to the keyword searching.

Search can be done at 3 different levels of complexity in a smartphone using an interface that is based on the ontology. Simple queries can be made browsing ontology concepts (called search by concept). Users can make searches like “Find a luxury restaurant” by specifying the concept of searching subject and get the results. Users can also make more complex questions expressing what they want. With Search with constraints on a particular object feature, they can make queries like, for instance, “Find a well known place related to architecture topic”. The most complex and powerful semantic search functionality is named Advanced Semantic Search. It helps to formulate a question, relying on the specification of complex property constraints over multiple objects. This makes possible queries like, for example, “Find dinning service place which has European-style, nearby a commercial center that sells clothes”. SPARQL queries are automatically generated and are sent to the semantic reasoning server. The results are sent back to mobile client. In addition, the team developed Quick Search with SVO model. SVO stands for Subject - Verb / adjective - Object. By expressing a query in form of these three basic elements in natural language, this model allows creating multi purpose quick search for travel information in a few steps. This system can understand different types of question such as “Find place nearby lake” as well as “Is Alfresco pizza restaurant near by Hoan Kiem lake?”, and even “Is Hanoi Opera House nearby Hoan Kiem Lake?”. Finally, they developed a Semantic-aware travel route planning algorithm, in which users can express their personal preferences on tourism resources (called Points-of-Interest, POI) and the level of interest for each preference. The system has a second algorithm, that, based on a set of candidate POIs, will generate an itinerary optimized both on the distance aspect as well as the level of satisfaction of these POIs to user interest topics. The main architecture of STAAR is divided into three main components:

- Semantic data integration and management: This component is responsible for integrating proprietary data of system with proper external data from Linked data repositories such as DBPedia, GeoNames. These data are then imported in form of triples. This component is in charge of executing semantic query in response of requests from web service component and web application component.
- Web service: To support multi-platform tourist guide applications in the role of client in a distributed environment, a set of web services is developed. These services hide the implementation detail of the operations that manipulates ontology and semantic data.
- Client: STAAR system is currently deployed for both the web and the Android phone. The web-based system that allows searching tourist resources in accordance with the preferences expressed in a semantic profile and suggested itineraries for visitors on the set of locations, while the Android phone-based system allows users to search quickly a system resource with a simple constraint as well as with complex constraints on one or more objects.

Finally, STAAR can enrich travel data extracting related information from from online data services such as Youtube.

One important feature of STAAR is that touristic recommendations rely heavily on the users personal profiles and on their positions through GPS navigation systems. Despite STAAR system helps the user with location-based problems in an effective way, it does not consider the date of any event or the scheduling of the tourist itineraries that are calculated. The temporal aspect of the events is one of the main aspects that our system takes in consideration.

3.1.2 Eiffel

The main goal of the Eiffel project is to provide users with an intelligent and multilingual semantic search engine dedicated to the tourism domain, allowing tourism operators and local territories to highlight their resources. The end users will be able to use the touristic search engine in order to organize their trips on the basis of contextualized, specialized, organized and filtered information [23]. By posting queries, depending on their profiles, they will have access to distributed and highly heterogeneous data in the tourism domain.

Eiffel application is driven by a tourism ontology based on conceptual graphs [24] and is able to automatically detect, select and classify tourism content distributed in the web, populating the ontology. In particular, it is capable to extract temporal and geographic information about touristic offers. Eiffel can also analyse touristic behaviours, extracting users models and profiles, in order to be able to give personalized recommendations, through the use of a reasoning engine.

Researchers from the Eiffel project adopted a symbolic approach relying on patterns and rules for the detection, extraction and annotation of temporal expressions in unstructured web pages. Their method is based on the use of transducers [25], being able to deal (i) with complex and imprecise temporal information, (ii) heterogeneous tourism web pages and (iii) the difficulty of linking the temporal and spatial extracted information to the proper tourism object.

What'sUp differs from the Eiffel approach mainly in terms of the techniques used for extraction of event information from web pages, being limited to structured web pages, where events temporal and spatial information can be easily tracked. It differs in the ontology language used: OWL 2, in our case. Note also that Eiffel was not designed as a mobile application.

3.1.3 IDUM project

The work of Giuliano Candreva, Gianfranco De Franco, Dino De Santo et al. introduces us to a system called IDUM [26]. IDUM exploits two technologies that are based on the state-of-the-art ASP system DLV: a system for ontology representation and reasoning, called OntoDLV [27] and HιLϵX [28], a semantic information extraction tool. The core of IDUM is an ontology which models the domain of the tourist offers. The ontology is automatically populated by extracting the information contained in the tourist leaflets produced by tour operators. More in detail, in the IDUM system, behind the web-based user interface (that can be used by both employees of the agency and customers), there is an "intelligent" core that exploits an OntoDLP ontology for both modelling the domain of discourse (i.e., geographic information, user preferences, and tourist offers) and storing the available data. Focusing on the HιLϵX system, it allows for automatically processing the received contents, and to populate the ontology with the data extracted from tourist leaflets. Regarding the unified document representation, the idea is that a document (unstructured or semi-structured) can be seen as a suitable arrangement of objects in a two-dimensional space. Each object has its own semantics, is characterized by some attributes and located in a two-dimensional area of the document called portion. A portion is defined as a rectangular area uniquely identified by four Cartesian coordinates of two opposite vertices. Each portion "contains" one or more objects and an object can be recognized in different portions. The language of HιLϵX is founded on the concept of ontology descriptor. A descriptor looks like a production rule in a formal attribute grammar, where syntactic items are replaced by ontology elements, and where extensions for managing two-dimensional objects are added. Each descriptor allows the description of: an ontology object in order to recognize it in a document or how to "generate" a new object that, in turn, may be added to the original ontology. The information regarding the tourist offers provided by tour operators is received by the system as a set of e-mails. Each e-mail might contain plain text and/or a set of leaflets, usually distributed as PDF or image files which store the details of the offer (e.g., place, accommodation, price). Leaflets are devised to be human-readable, might contain both text and images, and usually do not have the same structure. e-mails (and their content) are automatically processed by using the HιLϵX system, and the extracted data about tourist offers (illustrated on Figure 3.2) is used for populating an OntoDLP ontology that models the domain of discourse, called the "Tourism ontology".

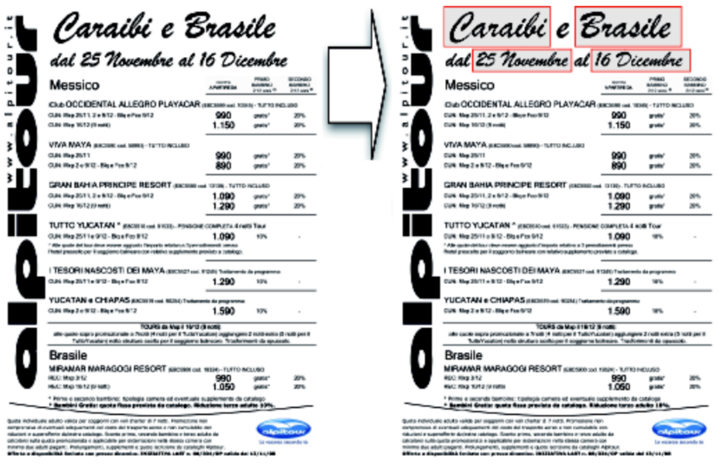


Figure 3.2: Extracting offer information (Candreva, G. et al., 2008)

This ontology contains a TouristicOffer class that has an instance for each available holiday package. Basically, after some pre-processing steps, in which e-mails are automatically read from the in-box, and their attachments are properly handled (e.g., image files are analysed by using OCR software), the input is sent to HιLϵX that is able to both extract the information contained in the e-mails and populate the TouristicOffer class. This has been obtained by encoding several ontology descriptors (actually, there were provided several descriptors for each kind of file received by the agency).

The system architecture explains how all the components cooperate with each other. The architecture of the IDUM system is made of four layers: Data Layer, Information Layer, Knowledge Layer, and Service Layer. In the Data Layer, the input sources are dealt with. In particular, the system is able to store and handle the most common kind of sources: e-mails, plain text, pdf, gif, jpeg, and HTML files. Regarding the three first layers, the Information Layer provides ETL (Extraction, Transformation and Loading) functionalities, in particular: in the loading step the documents to be processed are stored in an auxiliary database (that also manages the information about the state of the extraction activities); whereas, in the Transformation step, semi-structured or non-structured documents are manipulated. In first place, the document format is normalized; then, the bi-dimensional logical representation is generated (basically, the HιLϵX portions are identified); finally, HιLϵX descriptors are applied in the Semantic Extraction step and ontology instances are recognized within processed documents. The outcome of this process is a set of concept instances, that are recognized by matching semantic patterns, and stored in the core knowledge base of the system where the tourism ontology resides (Knowledge Layer). The IDUM system permits to extract information about tourist objects from non structured or semi-structured documents in an efficient way. In the case of our project, we need to extract information from structured documents, like HTML and RDF.

In 2004, the e-tourism working group, Katharina Siorpaes, Kathrin Prantner and Daniel Bachlechner [29] designed a Class Hierarchy for an e-tourism Ontology. This ontology holds many aspects that seem interesting to the building of our application ontology. Next, we will show the Class Overview of the built ontology.

- Accomodation
- Activity
- ContactData
- DateTime
  - OpeningHours
  - Period
    - DatePeriod
    - TimePeriod
  - Season
- Event
- Infrastructure
- Location

  - GPSCoordinates
  - PostalAddress

- Room
  - ConferenceRoom
  - Guestroom

- Ticket

This ontology has also documentation about each class properties. We show the class properties of the class Activity on Table 3.1.

Name	Type	Documentation
canBeDoneAt	Instance	This property links an individual representing the infrastructure where an activity can be done.
hasName	String	This property indicates the name of an individual.
Wednesday	String	This property indicates the type of an individual.

Table 3.1: Table describing the Activity class properties [29]

3.1.4 Siri

We may find certain characteristics of our application alike some functionalities offered by Apple's latest system, Siri, which allows a person visiting a new city to quickly discover the best places to have dinner or the best cultural spots within the city, e.g., monuments, views, historical locations. Nevertheless, our application is not place driven, but rather event-driven. We focus on events that occur within the space of a month or during all the year and consider always the temporal aspect (date and time). That aspect is not so deeply analysed in Siri, as it is driven by places and not cultural events. Therefore, Siri will not be so reliable and precise when returning results about ongoing or upcoming events in a city, with all the details that are stored in the Web. Another main difference between these two approaches is the possibility of producing the input using voice, faculty which is not offered in our system, since that we give priority to a more reliable and precise searching mode, using natural language input by text. Also, voice input analysis has many complex issues that we do not mean to address in this project.

3.1.5 SPARQL-DL Engine

Recently, Gerasimos Tzoganis, Dimitrios Koutsomitropoulos and Theodore S. Papatheodorou developed an approach to query OWL 2 ontologies, using SPARQL-DL [30], an extension of SPARQL. SPARQL-DL is known as a substantial subset of SPARQL with clear OWL-DL based semantics and claimed to be significantly more expressive than existing DL Query languages and still able to be implemented on top of existing OWL-DL reasoners like Pellet. SPARQL-DL's designers took advantage of the fact that SPARQL is designed so that its graph matching semantics are extensible to any arbitrary entailment regime and the fact that there is a well-defined entailment relation between valid OWL-DL ontologies. The applications permits to load an ontology and run SPARQL-DL queries on it. It then displays the results of the query. The interface of the application is very user-friendly. It has a minimalistic and simple design. The team makes some experiments, using an ontology describing the domain of a university. They used the OWL 2 Validator to make sure it is a valid OWL 2 ontology. The application was tested with two sample queries proposed in the article and both queries were answered fast, despite the second one containing a cycle through undistinguished variables. It is assumed that Pellet behaves good in both queries. Pellet proves to be quite effective in supporting this expressivity. Thus, the whole application performs knowledge retrieval effectively.

3.2 Other Related Work

3.2.1 Examples of Existing Tourism Ontologies

Starting with the article from Kathrin Prantner, Ying Ding, Michael Luger and Zhixian Yan [31], it states that many e-tourism oriented web ontologies were developed already. In this article, the team identifies several publicly available formal tourism ontologies which show the current status of the efforts and may serve as a basis for problem specific tourism ontologies.

As the first example of existing tourism ontologies, the team presents **Harmonise Ontology**, which supports tourism organizations with exchanging data and information without changing their local data structures and information systems. It maps different tourism ontologies by using a mediating ontology, that is represented in RDF and contains concepts and properties describing tourism concepts, mainly dealing with accommodation and events.

Mondeca Tourism Ontology includes important concepts of the tourism domain which are defined in the WTO thesaurus [1] managed by the WTO (World Tourism Organization). The WTO Thesaurus includes information and definitions of the tourism topic and leisure activities. The dimensions which are defined within the Mondeca On-

[1]www.world-tourism.org

tology are tourism object profiling, tourism and cultural objects, tourism packages and tourism multimedia content.

OnTour Ontology is especially for the tourism domain and was developed by DERI (Digital Enterprise Research Institute). In addition to normal tourism concepts (location, accommodation, among others) it also includes concepts that describe leisure activities and geographic data.

Besides these ontologies referenced above, there are more examples. Although, the great majority of them are oriented to deal with accommodation issues and transportation details. In our case, those details are not considered in our approach, as we do not want to concern about how the user can get to the city. Our interaction with the user begins after that, when he is already visiting the city. For this reason, the ongoing cultural events and its most important properties (scheduling, location, category and price) are a much more relevant subject to us. As we stated before, our application is tourism oriented and helps the tourist to explore the local tourism offers.

3.2.2 Including Semantic Information in a website

Much of the Semantic Web function and practicality are still in development, and there are some considerably big obstacles to overcome. Decentralization gives developers the freedom to create precisely the tags and ontologies that they need. But, it also means that different developers might use different tags to describe the same thing, which could make machine comparisons difficult [32]. Moreover, a browser does not automatically know what the tags mean. There are conflicting viewpoints, when it concerns to the common languages and schemas to represent semantic information. Before we can use a set of terms, we need a precise indication of what specific vocabularies are being used. For that it is necessary to define a namespace. A standard initial component of an ontology includes a set of XML namespace declarations enclosed in an opening `rdf:RDF` tag. These components provide a mean to unambiguously interpret identifiers and make the rest of the ontology presentation much more readable. The proper ontology file will have a namespace so the Classes and Properties that are defined in it can not be confused with any other resources on the web.

We must also look to other semantic technologies that are emerging nowadays, but do not use any ontology to function. Currently, millions of websites have markup content that was marked-up using RDF, Microformats, Microdata, Schema.org, RDFa and more. We will give an overview of these formats, to make clear what options are currently available to make a semantic content rich website. We will present the Rich Snippets, i.e., the formats that are built inside HTML code. The ontology methods will be described later.

Schema.org

Starting with Schema.org, it is a collaboration by multiple companies to create a set of extensible schemas that enables webmasters to embed structured data on their web pages for use by search engines. Search engines including Bing, Google, Yahoo! and Yandex rely on this markup to improve the display of search results, making it easier for people to find the right web pages and content. This approach has Classes - or Types - (e.g., LocalBusiness, Person, Organization, Event) and Properties - also called Attributes - (e.g., openingHours, productionCompany, streetAddress). It is basically a dictionary of terms used for annotating data within normal web pages. Schema.org initial focus is on supporting the Microdata format (a HTML5 specification), despite the fact that its long term goal is to support a wider range of formats, like RDFa. Developers argument they found that microdata strikes a balance between the extensibility of RDFa and the simplicity of microformats.

Here is an example of a short HTML block showing basic contact information for Ricardo Santos.

```
<div>
  My name is Ricardo Santos but people call me Ricky.
  Here is my home page:
  <a href="http://www.example.com">www.example.com</a>
  I live in Albuquerque, NM and work as an engineer at ACME Corp.
</div>
```

Here is the same HTML marked-up with microdata.

```
<div itemscope itemtype="http://data-vocabulary.org/Person">
  My name is <span itemprop="name">Ricardo Santos</span>
  but people call me <span itemprop="nickname">Ricky</span>.
  Here is my home page:
  <a href="http://www.example.com" itemprop="url">www.example.com</a>
  I live in Albuquerque, NM and work as an
  <span itemprop="title">engineer</span>
  at <span itemprop="affiliation">ACME Corp</span>.
</div>
```

Recently, RDFa 1.1 Lite was supported to be used with schema.org. This work opens up new possibilities for developers who intend to work with schema.org data using RDF-based tools and defines a simplified publisher-friendly 'Lite' view of RDFa. There are already some successful use cases as a long list of companies are implementing this idea on their websites, adding these additional tags to their HTML code. We can particularly refer to a use case that is related with our project theme: Eventful, a web service which aims to help users searching for, tracking, and sharing information about events worldwide. Users can search for events worldwide by time, location, performer, and descriptive keyword. We found some results when searching events in Lisbon and think that Eventful is a tempting target to obtain event information from, being that this web service already adopted a semantic content improving measure. Also related to our project, we found a type in Schema.org vocabulary (list of entities and their components) that can be useful to us, Event. This type is used for describing an event happening at a certain time at a certain location. By its turn, this type has a set of more specific types that can categorize a certain event in a better way. As examples of these subtypes there are SocialEvent, SportsEvent, DanceEvent, MusicEvent or TheaterEvent. It permits to identify a specific touristic offer type, rather than only state that it is a touristic offer. A definition of an event could be something like:

```
<html itemscope itemtype="http://schema.org/Event">
  <head>
    <meta itemprop="name" content="Gus Cage">
    <meta itemprop="description" content="Gus birthday party">
    <meta itemprop="image" content="http://guscage.com/profile.png">
    <meta itemprop="startDate" content="20130524">
    <meta itemprop="endDate" content="20130525">
  </head>
  <body>
  </body>
</html>
```

When one of these semantic content describing methodologies becomes adopted by the majority of touristic providers, it will be much more practical to obtain data about cultural events from the Web. The fact that a website using these markup techniques has a great chance to improve its Google search results (not the ranking but more details and extra tidbits of information, like star ratings, prices, stock information, and even breadcrumbs) is a strong motivation to make use of this technology. In fact, the more motivation a certain technology inspires on global users, the more chances it has to be globally used. However, in Schema.org case, it is still early to bet in this technology for an application.

Ontology techologies

Some areas of the World Wide Web have already incorporated Semantic Web components. These include RSS feeds, which use RDF, and the Friend-of-a-Friend (FOAF) project, which proposes to create machine-readable personal web pages. FOAF documents are RDF documents with the purpose of describe persons and the relationships between them or between them and other objects. Technically, FOAF is an OWL ontology used as a descriptive vocabulary. Each profile has a unique identifier (such as the person e-mail or a URI of the homepage or web-log of the person), which is used when defining the relationships. Next, we present an example of a FOAF document:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

<#JW>
    a foaf:Person;
    foaf:name "Jimmy Wales";
    foaf:mbox <mailto:jwales@bomis.com>;
    foaf:homepage <http://www.jimmywales.com/>;
    foaf:nick "Jimbo";
    foaf:depiction <http://www.jimmywales.com/aus_img.jpg>;
    foaf:interest <http://www.wikimedia.org>;
    foaf:knows [
        a foaf:Person ;
        foaf:name "Angela Beesley"
    ].

<http://www.wikimedia.org>
    rdfs:label "Wikipedia".
```

This document describes a person named Jimmy Wales, with an e-mail, homepage and even a nickname. This person has interest in Wikimedia and knows Angela Beesley, described as another person. FOAF is just an example taken out of multiple ontologies that exist on the web. We can give other examples, such as:

- Dublin Core – a metadata element standard for cross-domain information resource description which provides a simple and standardized set of conventions for describing things online in ways that make them easier to find;

- DOAP (Description Of A Project) ontology, an ontology to describe open-source projects;

- ResumeRDF to express a Resume or Curriculum Vitae (CV), including information such as work and academic experience or skills.

In addition, many ontologies are domain specific in fields such as technology, environmental science, chemistry and linguistics. These will apply to fewer websites than those listed above, however.

RDFa

The essence of RDFa is to provide a set of attributes that can be used for carrying metadata in an XML language (hence the 'a' in RDFa). These attributes include *about*, *rel* (to indicate a relationship between two entities), *vel*, *property*, *datatype*, among others. RDFa is used for embedding RDF subject-predicate-object expressions within XHTML documents. Once more, we present an example of a short HTML block showing basic contact information for Ricardo Santos.

```
<div>
 My name is Ricardo Santos but people call me Ricky.
 Here is my homepage:
 <a href="http://www.example.com">www.example.com</a>.
 I live in Albuquerque, NM and work as an engineer at ACME Corp.
</div>
```

Here is the same HTML marked up with RDFa:

```
<div xmlns:v="http://rdf.data-vocabulary.org/#" typeof="v:Person">
  My name is <span property="v:name">Ricardo Santos</span>,
  but people call me <span property="v:nickname">Ricky</span>.
  Here is my homepage:
  <a href="http://www.example.com" rel="v:url">www.example.com</a>.
  I live in Albuquerque, NM and work as an
  <span property="v:title">engineer</span>
  at <span property="v:affiliation">ACME Corp</span>.
</div>
```

Microformats

Microformats are another web-based approach to creating metadata and other attributes in web pages. It allows software to process information intended for end-users automatically, such as contact information (`hCards`), geographic coordinates (`geo`), calendar events (`hCalendar`) and so on. In general, microformats use the class attribute in HTML tags (often <`span`> or <`div`>) to assign brief and descriptive names to entities and their properties. `rel` is used for describing relationships. As we did for the previously described formats, here is an example of a short HTML block showing basic contact information.

```
<div>
   <img src="www.example.com/ricardosantos.jpg"/>
   <strong>Ricardo Santos</strong>
   Senior editor at ACME Reviews
   200 Main St
   Desertville, AZ 12345
</div>
```

And here is the same HTML marked-up with the hCard (Person) microformat.

```
<div class="vcard">
   <img class="photo" src="www.example.com/ricardosantos.jpg" />
   <strong class="fn">Ricardo Santos</strong>
   <span class="title">Senior editor</span> at
    <span class="org">ACME Reviews</span>
   <span class="adr">
      <span class="street-address">200 Main St</span>
      <span class="locality">Desertville</span>,
       <span class="region">AZ</span>
      <span class="postal-code">12345</span>
   </span>
</div>
```

The class `vcard` is a root class name that indicates the presence of an `hCard`. Concerning the case of `hCalendar`, and analogously to `hCard`, it is a microformat standard for displaying a semantic representation of the iCalendar format information about an event on a web page, using HTML classes and `rel` attributes. There are even tools to automatically create these structures, through a form filling process. Adding the structured markup to a web page is relatively simple, also depending on the format that we choose.

Whether we want to fully embrace the Semantic Web technologies in a website infrastructure, or just want to make existing digital content more useful and easily accessible, there are several approaches to add structure to existing content on a website. This is the domain of Microformats, RDFa and microdata. As a resume, the table below (Table 3.2) lists the more common information types that we can easily markup as structured data.

Opportunities for structured markup and automatic transformation	
Information type	Structured Markup
People and Organizations	hCard, RDF vCard
Calendars and Events	hCalendar, RDF Calendar
Opinions, Ratings and Reviews	VoteLinks, hReview
Social Networks	XFN, FOAF
Licenses	rel-license
Tags, Keywords, Categories	rel-tag
Lists and Outlines	XOXO

Table 3.2: The most appropriated semantic markup technologies for each case (Rob Crowther, 2008)

Chapter 4

Developing techniques

In this chapter we present some considerations about the project in general, like the software development model that we used, the planning of the project and some diagrams of the requirement analysis phase.

4.1 Software Development Model

During the initial phase of our project development, we had an idea about what would be our final product. That idea was discussed with deeper details until we had the tools to begin the development. We may consider that our client is represented by the persons who had the initial idea and concept of this project, who in this case, are this project coordinators. Although our project was not very well defined from the beginning, there were numerous discussions during the development of the project, occurring almost daily, which means there was a close control of the course that the project was taken. The software development model that we choose to follow was the scrum model. This sort of agile methodology is divided in Sprints, which may have 2 to 4 weeks of duration. We had longer discussions once a month to decide in which elements of the projects we must work next and then a new Sprint was started. An agile methodology is also the most adequate to follow when the developing team is small, which is the case. Our approach to the project development was clearly iterative and incremental, as some elements of the project, like the ontology structure or the DCG grammar, were consecutively being incremented, because new requirements were requested. Despite this classification, some characteristics of waterfall and spiral models are included in our methodology. The requirements were not very rigid and some changes were allowed to be done in our project, which made waterfall model very unlikely to be chosen by us. Waterfall fits when requirements are frozen, and no changes are allowed. There was also a great probability of adding new requirements during the project development, so an agile methodology was the right choice.

4.2 Project Planning

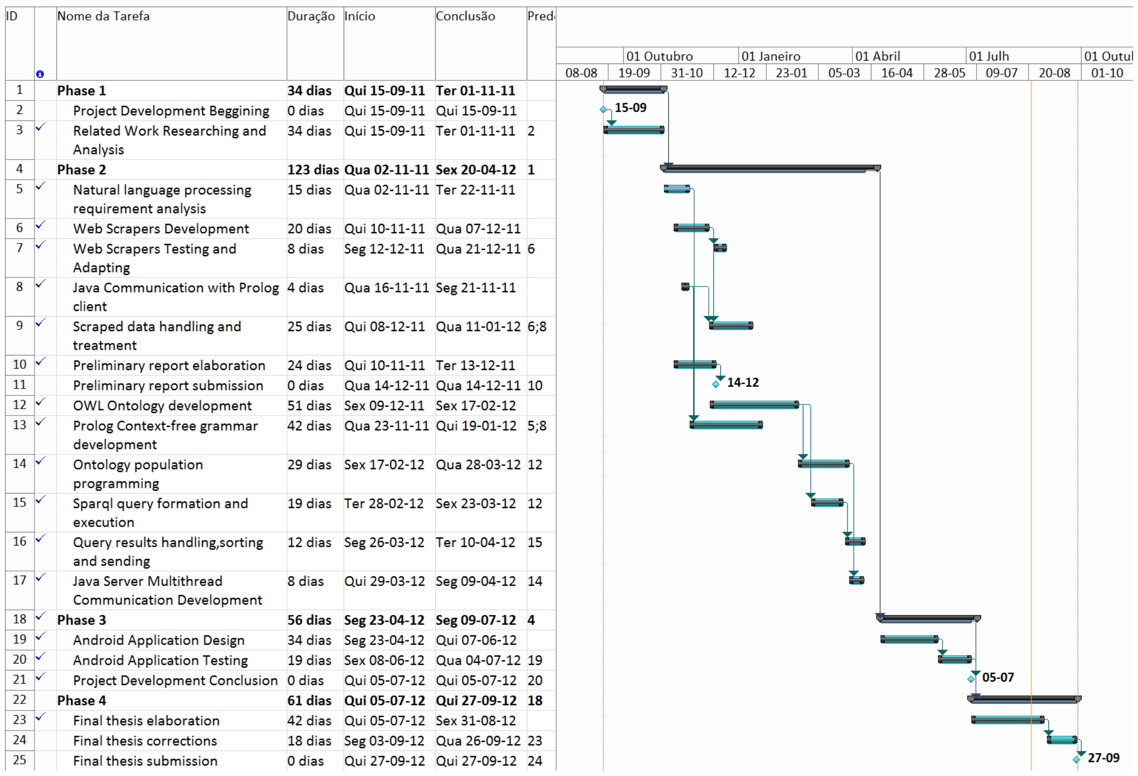
On the beginning of the project, we defined a simple planning based on the duration of each task. This planning was included on a document called specific agreement (*acordo específico* in Portuguese). The planning was the following:

- Related Work Research and Analysis – one month
- Web Scrapers development – two weeks
- Preliminary report elaboration – one month
- Ontology construction – three weeks
- Grammar construction for specific natural language processing – two months
- SPARQL Search method elaboration – one month
- Integration of the multiple modules on the project – one month
- Application interface development – one month
- Final Thesis elaboration – one month

As we can see, this planning has a duration for each task and does not specify any dates, only the project developing beginning date, which was September 14th 2011 and the final date, which was July 15th 2012.

Next, we present on Figure 4.1 a Gantt chart that illustrates what our real task accomplishing process was like, with date specifications included.

We divided our planning in four different phases. Phase one comprises the related work research and analysis and an initial overview of the existing tools and methods to conceive our application. Phase two is the longest phase and contains all the development work done on the server side and also the preliminary report, which was a mandatory document. Phase three includes the client side development, this is, the Android application construction and testing. The final phase represents the final thesis elaboration, including the future modifications that will be done after the first version delivery. Only the thesis correction task is not at 100% completion. As we can see, the final thesis report delivery date is set to the September 27th. This means that our maximum deadline to deliver the final version of this report and to submit it. However, the deal date was 15 July. This delay of one month is justified on Section 7.1.



ID		Nome da Tarefa	Duração	Início	Conclusão	Pred
1		**Phase 1**	**34 dias**	**Qui 15-09-11**	**Ter 01-11-11**	
2		Project Development Beggining	0 dias	Qui 15-09-11	Qui 15-09-11	
3	✓	Related Work Researching and Analysis	34 dias	Qui 15-09-11	Ter 01-11-11	2
4		**Phase 2**	**123 dias**	**Qua 02-11-11**	**Sex 20-04-12**	**1**
5	✓	Natural language processing requirement analysis	15 dias	Qua 02-11-11	Ter 22-11-11	
6	✓	Web Scrapers Development	20 dias	Qui 10-11-11	Qua 07-12-11	
7	✓	Web Scrapers Testing and Adapting	8 dias	Seg 12-12-11	Qua 21-12-11	6
8	✓	Java Communication with Prolog client	4 dias	Qua 16-11-11	Seg 21-11-11	
9	✓	Scraped data handling and treatment	25 dias	Qui 08-12-11	Qua 11-01-12	6;8
10	✓	Preliminary report elaboration	24 dias	Qui 10-11-11	Ter 13-12-11	
11		Preliminary report submission	0 dias	Qua 14-12-11	Qua 14-12-11	10
12	✓	OWL Ontology development	51 dias	Sex 09-12-11	Sex 17-02-12	
13	✓	Prolog Context-free grammar development	42 dias	Qua 23-11-11	Qui 19-01-12	5;8
14	✓	Ontology population programming	29 dias	Sex 17-02-12	Qua 28-03-12	12
15	✓	Sparql query formation and execution	19 dias	Ter 28-02-12	Sex 23-03-12	12
16	✓	Query results handling,sorting and sending	12 dias	Seg 26-03-12	Ter 10-04-12	15
17	✓	Java Server Multithread Communication Development	8 dias	Qui 29-03-12	Seg 09-04-12	14
18	✓	**Phase 3**	**56 dias**	**Seg 23-04-12**	**Seg 09-07-12**	**4**
19	✓	Android Application Design	34 dias	Seg 23-04-12	Qui 07-06-12	
20	✓	Android Application Testing	19 dias	Sex 08-06-12	Qua 04-07-12	19
21	✓	Project Development Conclusion	0 dias	Qui 05-07-12	Qui 05-07-12	20
22		**Phase 4**	**61 dias**	**Qui 05-07-12**	**Qui 27-09-12**	**18**
23	✓	Final thesis elaboration	42 dias	Qui 05-07-12	Sex 31-08-12	
24		Final thesis corrections	18 dias	Seg 03-09-12	Qua 26-09-12	23
25		Final thesis submission	0 dias	Qui 27-09-12	Qui 27-09-12	24

Figure 4.1: Gantt chart representing our project planning

4.3 System Requirements

As declared before, our system requirements were not very clear at the beginning of the development. Nonetheless, some guidelines were already defined from the conceptualization of the idea. Requirements analysis is critical to the success of a software project. Next, we will specify those requirements.

4.3.1 Functional Requirements

The application requirements that were defined initially are:

- The event extraction must be accomplished by using web scrapers
- The user must do his questions in natural language
- A grammar must be build and parse a sub-language of European Portuguese about event features
- An event must always have a location, a name and a date
- The results must be ordered by proximity to the user (the nearest first)
- The user must be able to see the location of an event on a map

- The ontology must have classes for When, Where and What, featuring the events
- Hour or day indications must be allowed on the natural language queries
- The system must update event information automatically

4.3.2 Non-functional requirements

- The information sources must be at least two
- The solution must have scalability to handle multiple clients at the same time (at least 20)
- The query response time must be reduced (up to 5 seconds for each query)
- The mobile application must have a high usability and user-friendly GUI interaction
- The project developing duration must not be more than 11 months

4.4 Use cases

We have several use cases that represent each interaction of the user with the system. We have use cases for making natural language questions, explore the results, see the details of an event and see the location of an event on the map. For each of these use cases, we present a use case diagram, according to the UML diagram design rules. These UML diagrams are behavioural, as they represent the system behaviour to the user interaction.

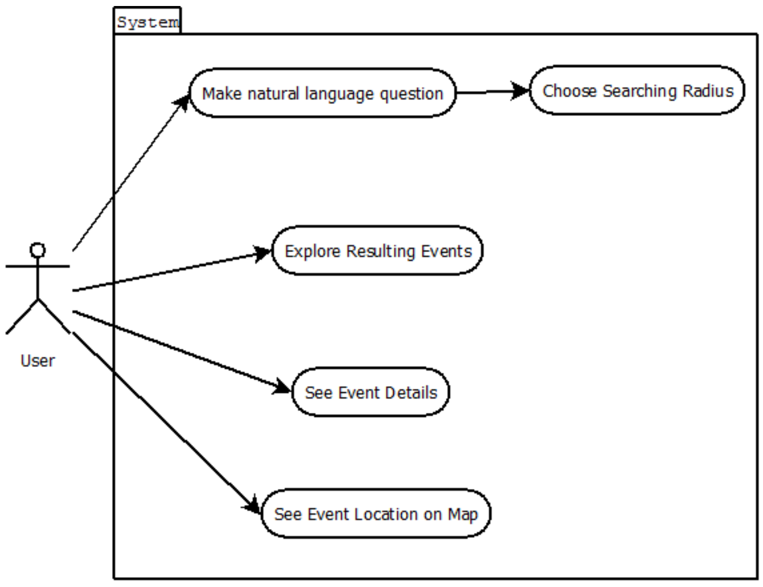


Figure 4.2: Use case diagram describing the user actions

4.5 System Sequence Diagrams

When the user makes a natural language question on the Android application, there is an exchange of data between the client and the server. This sequence of interactions between user and system could be represented by a System Sequence Diagram. Figure 4.3 depicts in more detail how this exchange occurs.

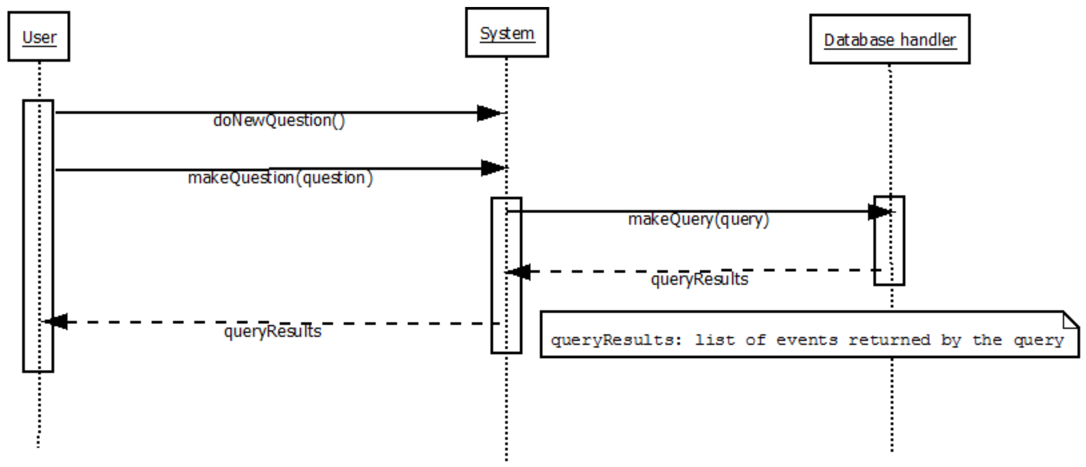


Figure 4.3: Sequence Diagram representing a search of events and results

This sequence ends when the user receives a list of results with events , resulting from his natural language question.

Another case that could be described on a SSD is when the user selects an event on the results list and sees the event details (date, location, name, category and so on). Figure 4.4 illustrates this sequence.

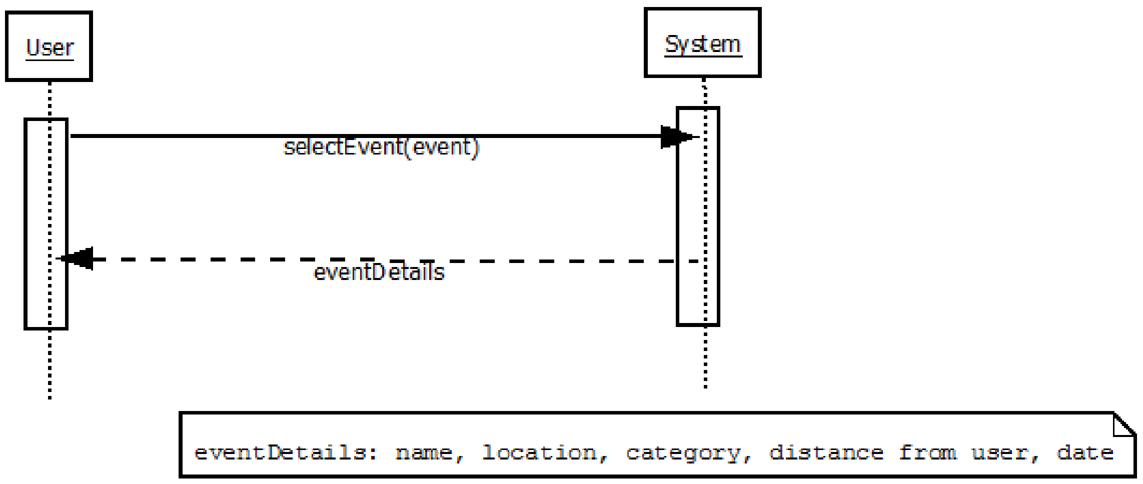

User
System
selectEvent(event)
eventDetails
eventDetails: name, location, category, distance from user, date

Figure 4.4: Sequence Diagram representing viewing the event details

4.6 UML Class Diagram

In this section, we present our UML class diagram, to provide a better overview of the functioning of our Java application. The Java application is responsible for a lot of functionalities on the server side of the **What'sUp** application. Most of these functionalities and methods are part of the back-end of the application, which means that the user does not interact directly with this side. On Figure 4.5, we can observe the main methods of our application and how the classes interact with each other.

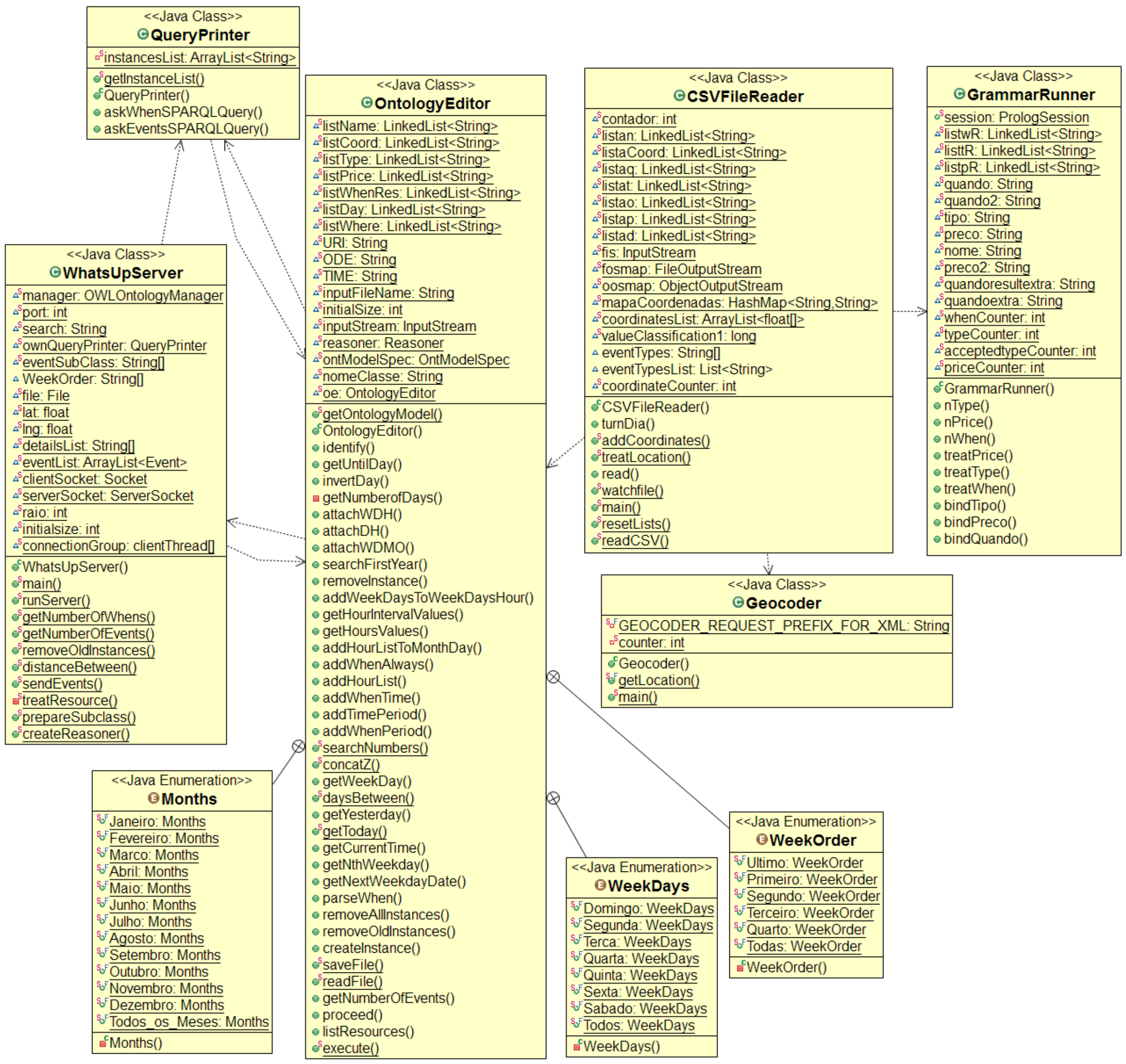


Figure 4.5: The server side class diagram of **What'sUp**

4.7 Development practices

All the Java and Android programming is documented and the most important methods are commented. There are many Exceptions on the code and they are handled in the best way. Because Object creation in Java is one of the most expensive operation in terms of memory utilization and performance impact, we make a great effort to create or initialize an object only when it is required in the code. We did some research on the existing libraries, avoiding having to “reinvent the wheel” and make the best use of these libraries. We tried to avoid throwing Exceptions on methods. Instead, we use, whenever that is possible, a try catch block. This way, we can handle the Exception and obtain more useful information out of it. We also try to use resources only during their useful life cycle, in order to reduce memory allocation.

Chapter 5

System Overview

As it was referenced before, most of these work is described on our article that was accepted for the INFORUM 2012 conference.

We may start with a quick overview of our system architecture. This way we can describe, in a simple way, how our system is designed. Figure 5.1 illustrates our system architecture.

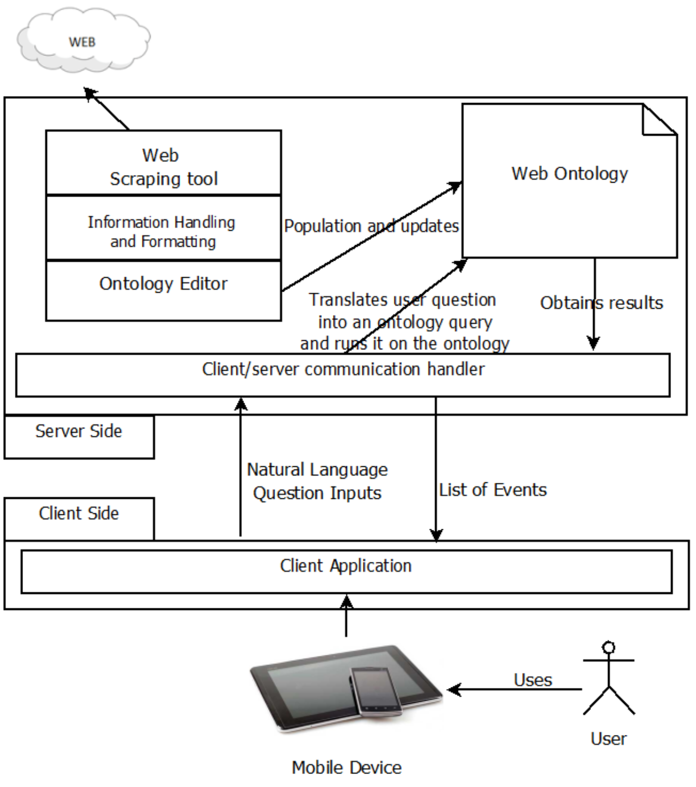

WEB
Web Scraping tool
Information Handling and Formatting
Ontology Editor
Web Ontology
Population and updates
Translates user question into an ontology query and runs it on the ontology
Obtains results
Client/server communication handler
Server Side
Natural Language Question Inputs
List of Events
Client Side
Client Application
Uses
User
Mobile Device

Figure 5.1: The **What'sUp** system basic architecture

We can verify the flux of information coming from the client side, in form of natural language questions. The server processes these questions and transforms them in ontology specific queries. These queries are run into the ontology and the returned results are sent back to the user and presented on the user's mobile device. The extraction and preparation of the digital contents about the cultural events is also done on the backend of the server. This system architecture is described with more detail on Section 6.11.

As we can see on the picture, the **What'sUp** project is divided into multiple modules, each one of them with a different purpose. Next, we will explain the functioning of each module and describe how they relate and communicate with each other, forming the complex workflow that exists in our project. This chapter also includes the algorithms that we had to develop, the description of the challenges that we were confronted with and our solutions and decisions during the development of the project.

5.1 Understanding temporal expression syntax variations

In order to do a efficient temporal expression processing, with the minimum failure rate possible, we tried to study the syntax of the temporal expressions that were scraped from the target websites. As our application is oriented to the Portuguese language, the temporal expressions that we captured are also in Portuguese. If in the future we want to adapt our application to extract and process expressions in any other language, we must adapt our grammar and the application interface. We built a table with some of the forms that we encountered and estimated some possible variations of those forms. We also registered which expressions we can process and which we cannot. The next table (Table 5.1) shows us some of the obtained examples, in order to give us a broader idea of the typology of the extracted temporal expressions.

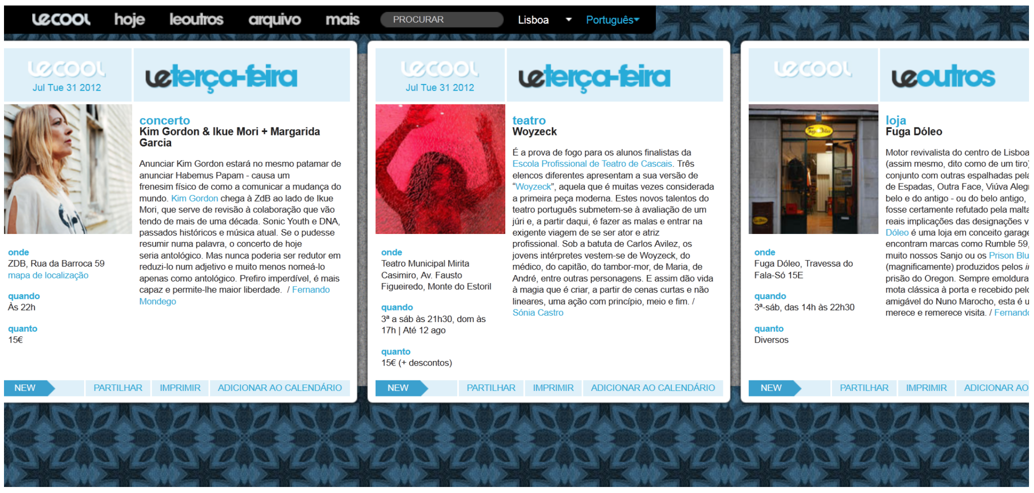
Temporal expressions	Handled
10 a 15 de Outubro or De 10 a 15 Setembro	Yes
Às 4h or A partir das 4h	Yes
4^{a} a dom às 21h — De 5 a 30 Out	Yes
Às 21h30 — Até 12 Out	Yes
Das 12h-23h or Das 12h às 23h	Yes
Das 10h às 15h — Todos os sábados	Yes
Almoços 2a-6a 12h30-14h30, Jantares 2^{a} a Dom 19h30-00h	No
Das 22h-2h Concertos, clubbing até 6h	No
Consulta calendário	Yes (Discarded)
17 e 18 Set a partir das 8h30	Yes
11 e 12 Nov, 17h-22h e 15h-22h	Yes
25, 26, 27 e 28, todo o dia	No
Às 21h30 seguido de debate pelas 22h	No

Table 5.1: Temporal expressions variations extracted from the Web

With this study we were able to understand better what were the requirements of our grammar, so a larger number of expressions could be successfully processed. Some expressions contain words that are completely unexpected like “jantar” (dinner) or “debate” (discussion). These words are not directly related to time, so we try to ignore these type of words, before the final expression is evaluated by the grammar predicates. In other cases, like “Consulta calendário” (consult calendar), we ignore the whole expression and no data is inserted on our ontology because no useful information about date or time is provided on the temporal expression.

5.2 Scraping information from the Web

We need to extract digital content about cultural events from the Web. For that, we use scraping techniques. The scraping work was one of the first tasks in our application, to be developed, considering its importance to the whole data workflow. It is in this phase that all the information cycle starts. This task was done using a plug-in for Mozilla Firefox web browser, named Outwit Hub (Professional Version). This scraping tool extracts information from websites, that is then stored in a Comma Separated Value (.CSV) document. We started by scraping event information from two websites (or on-line cultural agendas). The first is LeCool, a weekly magazine, with issues from the most cultural capitals in Europe, including Lisbon, which is our geographic target. The lecool main page is showed on Figure 5.2.



concerto
Kim Gordon & Ikue Mori + Margarida Garcia

Anunciar Kim Gordon estará no mesmo patamar de anunciar Habemus Papam - causa um frenesim físico de como a comunicar a mudança do mundo. Kim Gordon chega à ZdB ao lado de Ikue Mori, que serve de revisão à colaboração que vão tendo de mais de uma década. Sonic Youth e DNA, passados históricos e música atual. Se o pudesse resumir numa palavra, o concerto de hoje seria antológico. Mas nunca poderia ser redutor em reduzi-lo num adjetivo e muito menos nomeá-lo apenas como antológico. Prefiro imperdível, é mais capaz e permite-lhe maior liberdade. / Fernando Mondego

onde
ZDB, Rua da Barroca 59
mapa de localização

quando
Às 22h

quanto
15€

teatro
Woyzeck

É a prova de fogo para os alunos finalistas da Escola Profissional de Teatro de Cascais. Três elencos diferentes apresentam a sua versão de “Woyzeck”, aquela que é muitas vezes considerada a primeira peça moderna. Estes novos talentos do teatro português submetem-se à avaliação de um júri e, a partir daqui, é fazer as malas e entrar na exigente viagem de se ser ator e atriz profissional. Sob a batuta de Carlos Avilez, os jovens intérpretes vestem-se de Woyzeck, do médico, do capitão, do tambor-mor, de Maria, de André, entre outras personagens. E assim dão vida à magia que é criar, a partir de cenas curtas e não lineares, uma ação com princípio, meio e fim. / Sónia Castro

onde
Teatro Municipal Mirita Casimiro, Av. Fausto Figueiredo, Monte do Estoril

quando
3ª a sáb às 21h30, dom às 17h | Até 12 ago

quanto
15€ (+ descontos)

loja
Fuga Dóleo

Motor revivalista do centro de Lisboa
(assim mesmo, dito como de um tiro)
conjunto com outras espalhadas pela
de Espadas, Outra Face, Viúva Alegr
belo e do antigo - ou do belo antigo,
fosse certamente refutado pela malta
reais implicações das designações v
Dóleo é uma loja em conceito garage
encontram marcas como Rumble 59,
muito nossos Sanjo ou os Prison Blu
(magnificamente) produzidos pelos i
prisão do Oregon. Sempre emoldura
mota clássica à porta e recebido pelo
amigável do Nuno Marocho, esta é u
merece e remerece visita. / Fernando

onde
Fuga Dóleo, Travessa do Fala-Só 15E

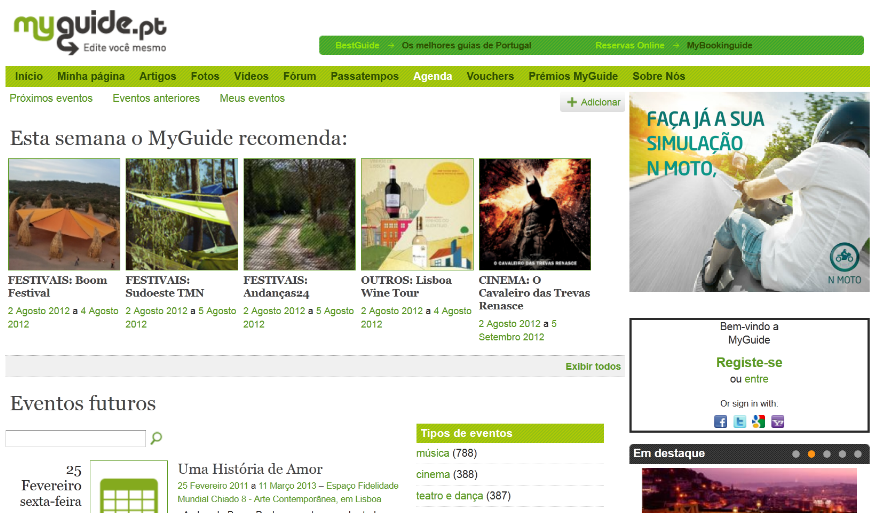
quando
3ª-sáb, das 14h às 22h30

quanto
Diversos

Figure 5.2: The Lisbon lecool website

The second is called MyGuide, a website about cultural events occurring in Portugal. Figure 5.3 shows the Myguide main page. Scraping the data from different websites is a major challenge, due to the lack of standards in the on-line tourism domain.

Each website shows information in a different structure and has different definitions of tourism concepts (for example, the event categories have different aggregations, depending on the provider). Some webmasters (or digital content providers) decided not to give relevance to some information about the cultural events. In most cases, this information


myguide.pt
Edite você mesmo

BestGuide → Os melhores guias de Portugal   Reservas Online → MyBookinguide

Início | Minha página | Artigos | Fotos | Vídeos | Fórum | Passatempos | Agenda | Vouchers | Prémios MyGuide | Sobre Nós

Próximos eventos | Eventos anteriores | Meus eventos   + Adicionar

Esta semana o MyGuide recomenda:

FESTIVAIS: Boom Festival
2 Agosto 2012 a 4 Agosto 2012

FESTIVAIS: Sudoeste TMN
2 Agosto 2012 a 5 Agosto 2012

FESTIVAIS: Andanças24
2 Agosto 2012 a 5 Agosto 2012

OUTROS: Lisboa Wine Tour
2 Agosto 2012 a 4 Agosto 2012

CINEMA: O Cavaleiro das Trevas Renasce
2 Agosto 2012 a 5 Setembro 2012

Exibir todos

Eventos futuros

25
Fevereiro
sexta-feira

Uma História de Amor
25 Fevereiro 2011 a 11 Março 2013 – Espaço Fidelidade Mundial Chiado 8 - Arte Contemporânea, em Lisboa

Tipos de eventos
música (788)
cinema (388)
teatro e dança (387)

FAÇA JÁ A SUA SIMULAÇÃO N MOTO,
N MOTO

Bem-vindo a MyGuide
Registe-se
ou entre
Or sign in with:

Em destaque

Figure 5.3: myguide website

was omitted from the website. On some other cases, the providers simply did not have that information from the original event provider available and, therefore, did not publish it on their website. This also depends on the type of cultural event. For instance, a set of information about a movie will have the different sessions scheduling and the list of theatres where the movie will be exhibited. Typically, a music concert will have a determined hour in a certain day and its location. On the other hand, a fair could have a period of days in a week as schedule.

For all these reasons, a different scraper must be design for extracting useful information from each website. We created two scrapers, one for each website. When defining the scraper details, we can manipulate the destination URL and have access to the HTML code, determining the exact location of the HTML text we want to extract from the HTML pages, by specifying the tags that exist before and after that text. All these specifications could be specified by the use of a regular expression, for more complex cases. Usually, these kinds of websites dispose their information using tables or lists, in a patterned way. This fact makes this way of scraping text very reliable. Given these steps, all the event information existing on the website is extracted to a file, for future handling. With one scraper we can define macros (combinations of scrapers or scrapers with advanced features) to apply a certain scraper to multiple web pages. This is useful to fetch multiple issues (or all of them) of a cultural journal, for instance. It is also possible to create jobs, that can execute macros in a regular period. We use this functionality to regularly execute the scrapers and update the information in the event database (OWL ontology). With these features, we are able to make the scraping process automatic, in which the information is regularly being renewed. We create an information cycle, that is regularly having a new iteration. The CSV file is constantly being changed, every time that a scheduled data scraping is executed. We had to find a way to treat all the new data and put it on the ontology. In order to do that, we created a listener on the file. This listener monitors file changes, and once the file is modified, it triggers the data modification and handling process, which we describe further ahead, starting on Section 5.3.1.

5.3 Definite Clause Grammar

The building of a Definite Clause Grammar, which will be necessary to recognize the natural language expressions, previously extracted from the cultural events agendas, was one of the main targets of development effort. Considering the event information processing and evaluating, we put more effort on the temporal aspect (in respect to "When"). We needed to develop a context-free grammar (CFG), defined in Prolog through a Definite Clause Grammar, mainly because it is a much more efficient and quick processing way. Therefore, to characterize and recognize temporal expressions we use the DCG formalism with Prolog. Although more powerful than Context-Free Grammars, we use DCGs (with context-free rules only) due to its simplicity and ease of use.

Grammar rules were developed taking into account a thorough analysis made on the scraping results (see Section 5.1).

A DCG grammar is a set of DCG rules. This grammar uses the `phrase()` predicate, a built-in predicate in Prolog. This is how it works:

```
?- phrase( < prologterm:P > , < list:S > ).
```

This predicate makes it easier to invoke goals involving predicates defined by grammar rules. It can be used for determining whether a list of words S representing a phrase or sentence conforms to the pattern specified by a particular grammar rule.

The first argument, P, should be a goal involving the predicate defined by the grammar rule, but should not contain extra arguments for the input and output strings (i.e., it should look similar to the head of the grammar rule). The second argument, S, should be the list of words to be worked on. It is assumed that the whole list of words must be recognized by the grammar rule.

All the predicates that we defined to evaluate this specific natural language expressions will be based on `phrase()`. The predicate that makes the temporal expressions evaluation is named `when()`. For example, if a `when()` predicate is defined by a rule like:

```
when(hour(X)) - -> hour(X).
```

then an appropriate call of `phrase()` would be:

```
?- phrase(when(hour(X)), ['as', 5, 'h']).
```

This would be completely equivalent to:

```
?- when(hour(X), ['as', 5, 'h'], []).
```

This predicate execution example would return the following value:

```
X = [hour(5)].
```

This result indicates that the `when()` predicate execution over the sentence, resulted in a successful recognition of the type of the temporal expression, as an hour reference (to 5 o'clock). In other words, we recognized the sentence "*às 5 h*" (at 5 a.m.) as a sentence of the type hour(5), which conforms to the pattern defined by our grammar rule.

Apart from these predicates based on `phrase()`, some auxiliary predicates were created, like predicates that manipulate lists (insertion and removal of elements from the list, recognition of lists of elements with the same syntax, and so on). Some of the grammar rules have conditions, like `day24()`, which evaluates whether or not a given term is an integer with the value between 1 and 31, which represents a day in a month. Similarly, it is possible to identify weekdays by defining literals for each day. For example, we define Monday as `weekDay('Segunda')` and Tuesday as `weekDay('Terça')`.

In order to evaluate which type of scheduling an event has assigned, different types of 'when' where created, defining rules for each of them. The scheduling information can be represented by an hour in the day, a period in a week or multiple days in a month. That is why the DCG grammar has to be flexible to all these possible cases. It would be easy to have a parse tree (as on Figures 5.4 and 5.5) representing the ramification of our DCG grammar, as it is based on the definition of simple rules. These simple rules are then used in the definition of more complex rules, augmenting our grammar evaluating capability. There exist special cases which force us to adapt the grammar specially for those cases. For instance, when an event (like a fair or market) has a scheduling like "Second Saturdays of each month" or "First Tuesday of the month during all the year" , we have to modify the grammar, by adding a new rule in the definition of the temporal predicate. This rule represents one more result that can be returned by the temporal predicate `when()`, after it evaluates a temporal expression. The rule will be defined by the grammatical characteristics of the temporal expression that is expected to be evaluated. We can understand that this type of temporal expression has an order of the weekday within the month (First, Second, Third or Last), followed by the weekday name (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday or Sunday). Then it has a set of words that indicates that it happens during the whole year (the same as 'of each month'). After collecting this information, we are ready to define the new grammar rule. So, considering that we already have auxiliary rules that help us to recognize the basic elements of the String, like:

- 'order()' to recognize the literals 'First', 'Second', 'Third' and 'Last';
- 'weekday()' for recognizing days of the week and

- ’alltheyear’ to recognize the last part of the String,

and assuming the rule is called ’dayofweekallyear()’ we build the rule the following way:

```
when(dayofweekallyear(WD)) --> order,weekday(WD),alltheyear.
```

When evaluating a String of this type, Prolog will use the grammar predicate relating to the temporality of the event and return a result of the type

```
when(dayofweekallyear(Tuesday))
```

This way, we are able to identify exactly the cultural events properties, to in the next step, insert them in the ontology.

5.3.1 Preparing expressions to the grammar evaluation

In our project, the path from the web page HTML code analysis to the ontology population is very complex. Data suffers multiple modifications along this path and is dynamically handled, modified and refined. Prior to be evaluated by the DCG grammar predicates, the temporal expressions that are extracted from the scraping process suffer a special treatment. This treatment is made by a piece of Java software that prepares the expressions, so they can have the right format to be evaluated by the predicates on our DCG grammar. We will pick a simple time expression as an example and will describe how a temporal expression is handled, in order to be evaluated by the DCG grammar predicate. We will consider the following time expression:

```
De 20 de Junho a 2 de Julho, às 21h30
```

(From 20th June to 2nd July, at 9.30 pm)

This expression denotes a time interval (measured in days) that lasts from 20th June to 2nd July, and has an hour indication, referring to 9.30 pm. These modifications include splitting the expression in words, each of them separated by commas. Also, non numerical words are surrounded by ” symbols. Some additional characters are concatenated to the final string for indicating that an hour or minute expression is present, ’h’ and ’m’, respectively). The result of this treatment, in our example, is the following:

```
'De',20,'junho','a',2,'julho','às',21,'h',30,'m'
```

(’From’,20,’june’,’to’,2,’july’,’at’,21,’h’,30,’m’)

This final format allows the DCG Prolog grammar to recognize these words as literals, and therefore, evaluate the whole string correctly and return a result for that same evaluation. So, considering our given example, the DCG rules responsible for evaluating this temporal expression are:

```
quando(intDiaMesHora(IDM,Hora)) --> intDiaMes(IDM),horas(Hora).
```

(when(dayOfMonthIntervalHour(DMI,Hour)) –> dayOfMonthInterval(DMI), hours(Hour).)

and:

```
quando(intDiaMesHora(IDM, Hora)) --> hora(Hora), intDiaMes(IDM).
```

(when(dayOfMonthIntervalHour(DMI,Hour)) –> hours(Hour), dayOfMonthInterval(DMI).)

The first rule evaluates the expressions in which the list of hours (identified by `Hour`) comes first than the day of month interval (identified by `DMI`) and the other evaluates the expressions that are on the opposite case. We can explain the structure of the first rule with the tree on Figure 5.4.

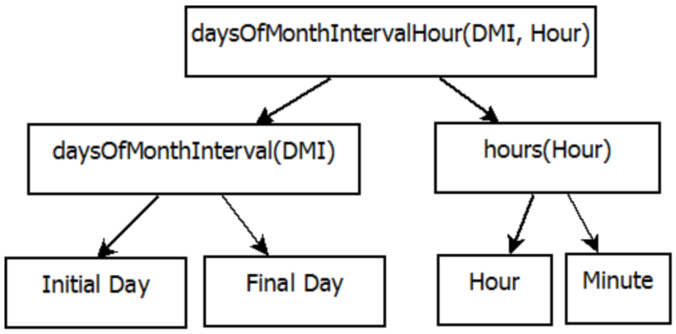


Figure 5.4: Tree demonstrating the first rule structure

In our example, the day of month interval comes first, so the rule that will be used is the first one. This rule returns the following result:

```
intDiaMesHora(diasMeses(diaMes(20,06,2012),diaMes(2,07,2012)),
[horaMinuto(21,30)])
```

(daysOfMonthIntervalHour(daysOfMonth(dayOfMonth(20,06,2012), dayOfMonth(2,07,2012)),[hourMinute(21,30)]))

This result respects the structure defined by the first rule (depicted in Figure 5.4). So, according to the rule structure, we can analogously illustrate the structure of this result, like we do on Figure 5.5.

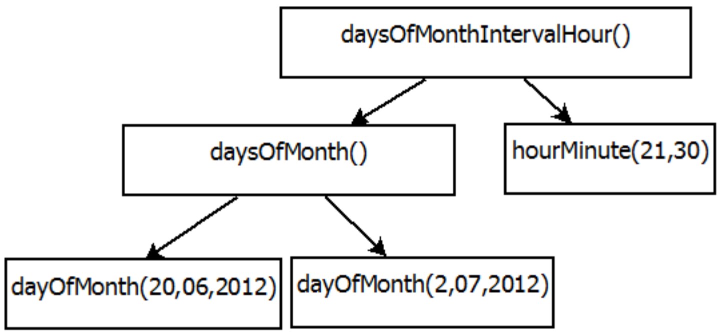


Figure 5.5: Tree demonstrating the result returned by the `when()` predicate

There are many predicates defined in the DCG, but we decided to focus on the predicate that handles temporal expressions (concerning "When"). The remaining existent predicates defined in the grammar concern, for example, one for handling "Where" expressions and one for handling the ticket price expression ("How Much") or detecting the cases it is free. The "Where" expression is handled in a different way (described on Section 5.7.1) and the "How Much" expressions are, in the majority of cases, considered simple strings given its evaluating complexity (see Section 5.3.3). Despite this fact, minimal treatment is still performed to these expressions, as they also must be introduced on the ontology.

5.3.2 Category evaluation predicate

In our DCG grammar we defined also a predicate to handle the event category. This predicate, called `category()`, is the simplest predicate on our DCG grammar, as it is a set of simple rules. We set this predicate to receive and analyse a category expression, which is usually one word, and depending on that expression, associate it with one of the possible event categories or not. In order to accomplish this task, we also made a study to make an estimation of the type of expressions that were extracted. We first looked at all possible category terms and tried to make associations or discover relations between them. For instance, the term "short film" is associated with Cinema. The same goes to the term "outdoor cinema". So, we made one rule that classifies all the "short film" events as Cinema events. This process is very important, as it reduces the number of overall categories that the user has to deal with. This process of aggregation is very useful as it also allows a category to have a larger number of events associated with it. This aggregation on the category system is also important when the user wants to make a more specific question. If the user wants to know where he can watch a short film on the city, he types, for instance:

"*Where can I watch short films today?*"

As these aggregation processes are expanded to the server side, whenever it translates a natural language question, it will return all the Cinema events. This behaviour is preferable than if the user would not have any result, due to the non-existence of aggregating processes.

There are also category expressions that are related with two of our pre-defined categories. For instance, the term "video and djset". An event with this category profile could be associated to Cinema but also to Concert. We tried to make clear separations between our categories, so when the user searches for a certain category, he is returned the right results. If a category expression is not recognized by our predicate or if it is not a category that we want to include in our system (for example, an event of the type "Restaurant", which is not properly a cultural event but a place), the event containing that category will not be inserted into the ontology.

After evaluating each String (representing an expression evaluation result) and parsing the result, the Java application creates an OWL instance of the class *Event* that represents the event containing this time expression and fills its properties, mainly the property related with its date, according to the parsed results. Each time a scraping schedule is executed, it triggers the whole data handling process and if the information is new (the scraping task may extract the same information out of the web cultural agendas, in case this information does not change), the ontology file will be automatically saved.

5.3.3 Price evaluation predicate

On table 5.2, we can see the syntax and structure of the price expressions that we extracted, in order to do our study.

Price expressions
20 € pass
10 € (with CD offer)
8 € - 30 €
10 € until à 1 a.m., 13 € after
With invitation
7,5 € (5 € for students)
Variable
8 € pre-sale, 10 € on the same day, 6 € clubbing
12 € for participants (Children until 12)
8 € and 9 €, snack + drink
3,5 € one film (see remaining prices)

Table 5.2: Price expression variations extracted from the web

Comparing to the temporal or location expressions, the price expression is a much more variable expression, according to our previous study. Regarding this fact, we decided to make an event price evaluating predicate with a more basic functioning. Despite not being so powerful as the temporal predicate (`when()`), this predicate, called `howMuch()` has a rule that identifies numbers on the expression and the Euro symbol (€). This way, we can detect when the price expression has a well defined value. This predicate has the capability of detecting keywords (like "*entrada livre*") that are usually utilized to express that the event has a free entrance. We ultimately decided not to give so much importance to the price relation of the cultural event. Our syntax study also allowed us to notice that the price expression has many times references to other details or does not even reference a price value. Sometimes, an event has multiple prices, or the price changes according to the date. There are also events with special price conditions, according to age, VIP states, associates, and so on). Faced up to these facts, we estimated that it would require a much bigger effort to have a successful evaluation rate equal to the rate on temporal expressions. Due to a large amount of words that are outsiders in the expression and this makes the predicate very inaccurate, as a large set of variations, with multiple meanings are possible. Another strong reason that justifies this decision is the fact of not always existing a value to the price. Many times this value is not specified on the web. The price expression is in fact the most heterogeneous and unpredictable expression that we faced.

5.4 The Ontology

The main element of this project architecture is our web ontology. The Web Ontology is written in OWL 2, the latest and current version of the standard ontology language. This ontology language is defined by a set of W3C recommendations. OWL 2 has a RDF/XML-based serialization for the Semantic Web, and that is the serialization type that we use in this project. To edit the ontology we used Protégé 4.2, one of the most used ontology editors. Protégé 4.2 works with the OWL sub-languages and with the OWL 2 Profiles. OWL DL is one of the three sub-languages of OWL, being the other two OWL Full and OWL Lite. This sub-language supports those users who want maximum expressiveness. OWL DL is more expressive but still ensures completeness and decidability, i.e., all the calculations will compute and terminate. OWL DL (DL for description logic) corresponds to a field of research concerning a particular fragment of decidable first order logic. OWL 2 provides three profiles: OWL 2 EL, which is designed for ontologies that contain a large number of classes or properties; OWL 2 QL, which is aimed primarily at query answering; and OWL 2 RL, which is aimed at application that require quite a bit of the expressivity provided by OWL DL but also require scalable reasoning. OWL 2 works with the Open World Assumption, this is, everything that is not stated, is unknown or undefined. Moreover, the reasoner that we'll be using is Pellet OWL Reasoner, an open-source Java based OWL 2 reasoner. A reasoner is a collection of software that allows understanding an ontology and inferring about it. Our OWL 2.0 ontology has been created to shape the event structure in terms of its temporality (*When*), spatiality (*Where*) and the type of event (*What*). All these events are categorized in types, such as *Cinema*, *Concert*, *Theatre*, *Festival*, among others. Each one of these types of event are OWL 2 classes and are subclasses of the Class *Event*, which represents an event. About the time relation, it is the most complex aspect of the ontology that has to be handled, due to its variety and heterogeneity. We also have a class that lists all the weekdays. This will provide an easier way to semantically define the different kinds of events, as the application will distinguish each of them by its properties.

The fundamental element in our project is the ontology, as it is the place where data is stored, queried and returned to the user. We decided to create our own ontology, named **What'sUp Ontology**, given the characteristics and objectives of our project. However, we took some guidelines in consideration [16]. This ontology was built in such a way that, in the future, it could be used on the Web. Moreover, it is extensible and reusable for other purposes.

There are already many ontologies on the Web that can be reused and extended in order to create our own ontologies. One of them is Time Ontology, which defines temporal relationships and uses multiple time definition levels (time intervals or time instants) as well as different time units (weeks, days, hours or minutes). We analysed this ontology to verify if it had potential to help us building our ontology, as its domain is closely related to what we need for describing our cultural events characteristics, namely the temporal aspect. We came to the conclusion that this ontology is too simple to cover so many cases of time relations that we had to handle. Despite that some of the time expressions that we deal with are capable of being described using this ontology, there is a significant set of expressions that could not be represented with the terms defined on the Time Ontology. However, we can still make a term mapping between Time Ontology and **What'sUp Ontology**, using the *owl:sameAs* built-in property, to state that two URI references refer to the same concept. For example, our ontology has an enumerated class named *WeekDay* that represents the same concept as the enumerated class *DayofWeek* in the Time Ontology. Thus, we could define that equality of concepts like this:

```
<rdf:Description rdf:about=
"http://www.co-ode.org/ontologies/ont.owl\#WeekDay">
<sameAs rdf:resource="http://www.w3.org/2006/time\#DayOfWeek"/>
</rdf:Description>
```

5.4.1 The ontology structure

In order to create our own ontology, we had to do ontology engineering work, defining the taxonomy, properties articulation and structural form. Our OWL 2.0 ontology has been created to shape the event structure in terms of its temporality (*When*), spatiality (*Where*) and the type of event *What*. We will focus on the most important ontology classes and properties. We will have a taxonomy of events with root class *Event* and each particular event will be an instance of the category in the taxonomy that best characterizes it. Feira da Ladra, a Flea Market in Lisbon, will be an instance of *Fair*, for example. We have a defined set of subclasses of *Event*, as illustrated on Figure 5.6.

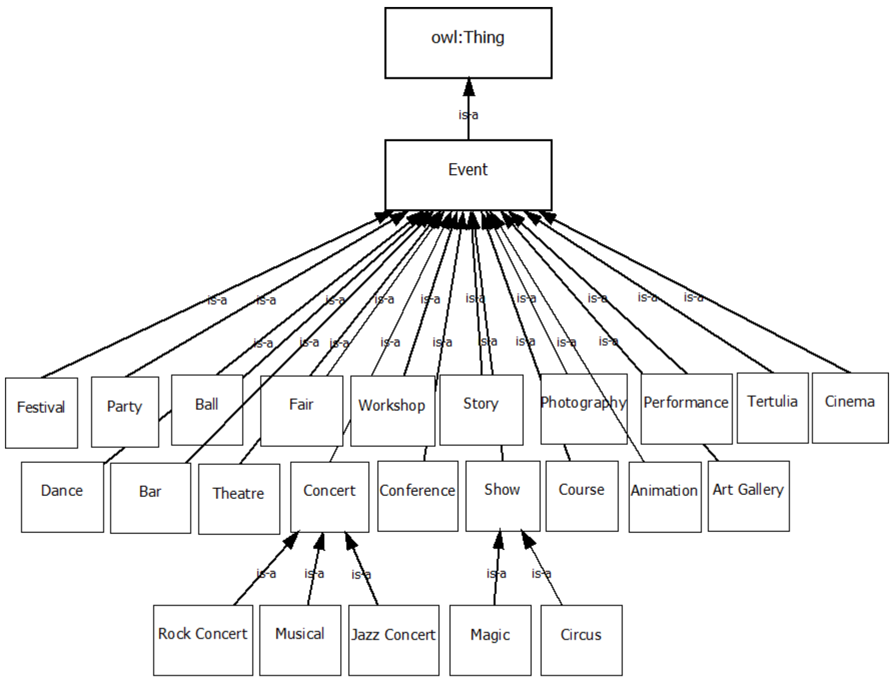


Figure 5.6: The event categories organization on **What'sUp Ontology**

All the classes on our ontology are subclasses of the main class *owl:Thing*. The same goes to our class *Event*. Also, all classes have the subclass (which is the same as saying that they are superclasses of) *owl:Nothing*. Like it is shown on the figure, on its turn, the class representing an abstract event has many subclasses, each one of them representing a different event typo. Some of those subclasses, like *Show* have a second level of ramification (corresponding to *Magic* and *Circus*) as it is shown.

As the *Event* class incorporates the event type, we will need further classes for the the remaining event aspects: *When*, *Where* and *How Much*. Three properties are defined to relate the *Event* class with those three aspects: *hasLocation* (domain *Event* and range *Location*), *hasWhen* (domain *Event* and range *When*) and *hasCost* (domain *Event* and range *xsd:String*, as illustrated in Figure 5.7. As said before, all these events are categorized in types, such as *Theatre*, *Concert*, *Party*, *Gallery Exposition*, among others (all of these are subclasses of the OWL class *Event*). Each *Location* individual has a property representing its geo-coordinates. We will focus on the time aspect, as it is the most complex aspect of our ontology, due to its variety and heterogeneity.

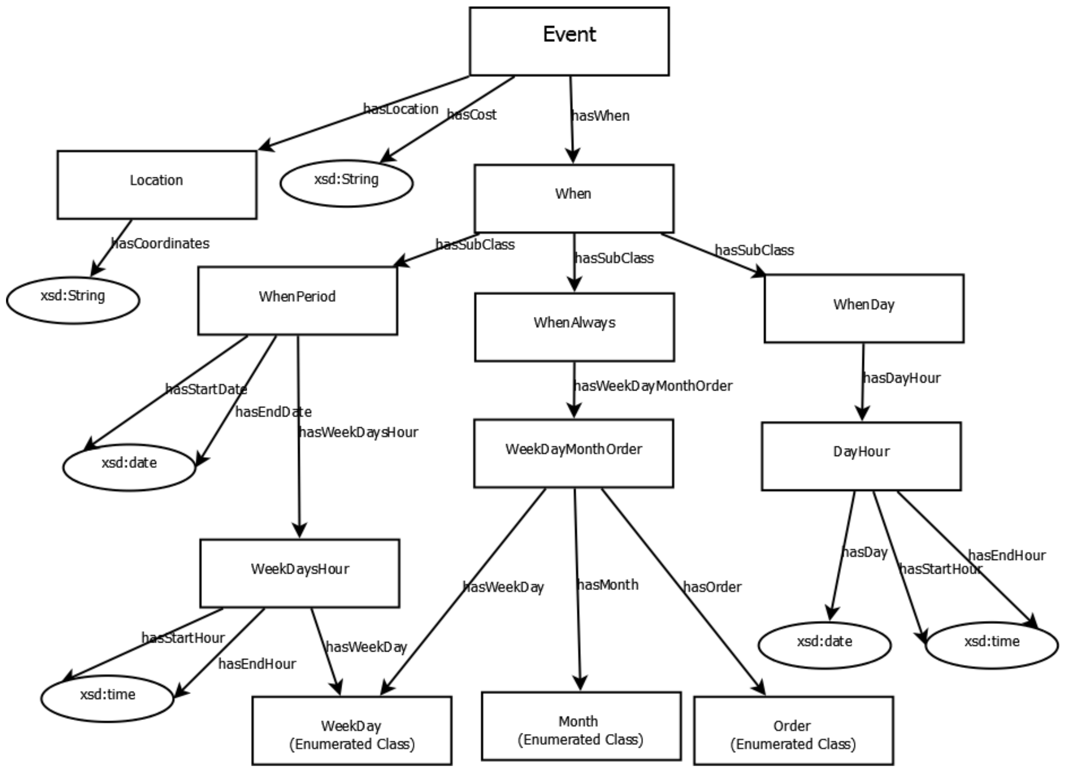


Figure 5.7: The **What'sUp Ontology** structure

Every *Event* instance is related with a particular *When* instance through the object property *hasWhen*. These relations are depicted in Figure 5.7.

We have divided the temporal specifications of the events in three main categories (being represented as subclasses of the class *When* in our OWL 2 ontology):

WhenDay

Corresponds to a particular date and time. The events that have this time relation type occur in certain dates, restricted within the period of one day and at a certain hour or during a certain hourly period. Examples are: "23rd April at 11 pm" or "25th April from 10 am to 1 pm and from 2 pm to 6 pm". We may have one or more hourly periods in a single day on these event types, like "22nd May from 10 am to 1 pm and from 2 pm to 6 pm". This configuration generates two *WhenDay* individuals, one for each time interval, although with the same day. For events that occur in several days, for example "3rd, 5th and 14th July at 1 pm", it generates one individual for each pair <date - time>. In its turn, each of these *WhenDay* individuals have an associated *DayHour* individual, where the proper date value and time value/interval are specified. One *WhenDay* individual has one and only one *DayHour* individual associated. The *DayHour* class deals with the aggregation of a particular date and a starting hour or an hourly period. This means that, in this last example, three individuals of *WhenDay* will be generated. This feature will simplify queries and answers will be more specific and focused on the user interests.

WhenPeriod

Periodic events will be handled by another category, *WhenPeriod*, corresponding to consecutive dates and represent repetitive time information. There is always a starting date and an ending date in a *WhenPeriod*. Examples are: "from 23rd of June to 15th August at 1 pm" or "Thursday, Friday and Saturday at 11 pm , from 1st to 3rd July". Analogously to <*WhenDay-DayHour*> relation, a *WhenPeriod* instance will always have an individual of the class *WeekDaysHour*, in order to aggregate a particular weekday or weekday interval and a starting hour or hourly period. This relation has a one-to-one cardinality. We have an enumerated class that lists all the weekdays, named *WeekDay*.

WhenAlways

The third category, *WhenAlways*, contains events that happen during all the year, like for instance, the Flea Market that opens "every Tuesday and Saturday from 6 am to 6 pm" or the Old Pipes market that opens "every first Sunday of each month, from 10 am to 6 pm". As it happens in the other *When* class types, there is a one-to-one relation between a *WhenAlways* instance and a *WeekDayMonthOrder* instance, which aggregates a particular nth weekday (or weekday interval) of the month and an hour or hour interval that happens during all the year or during a certain month. Two object properties are used for accessing the weekday and order of a certain instance of *WeekDayMonthOrder*, namely *hasWeekDay* and *hasOrder*. *Order* is also an enumerated class representing the nth occurrence of a certain weekday in a month: first, second, third, fourth or last. *Month* lists all the months in a year. We have the possibility to represent the "last Sunday of each month" as an instance of the class *WeekDayMonthOrder*, which aggregates the class *WeekDay* with the class *Order* and *Month*. For example, the Flea Market occurring any Tuesday and Saturday, from 6 am to 6 pm will imply one instance of *WhenAlways* with one instance of *WeekDaysHour* with value of the property *hasWeekDay* aggregating Tuesday and Saturday and the time interval associated with that instance.

Each *DayHour*, *WeekDayMonthOrder* or *WeekDaysHour* instance will have a start time or a hourly period. We did not create special classes to handle those concepts and, instead, used two Datatype properties: *hasStartHour* and *hasEndHour*. While only the first one is necessary to define a starting hour, both are used to define an hourly period.

An event like a theatre show from 13th to 30th June, on every Tuesday and Wednesday at 9 pm, Thursday, Friday and Saturday at 9.30 pm. There will be an instance of *WhenPeriod* for every <weekdays - time> pair of the event: one for Tuesday and Wednesday and another for the rest of the weekdays.

5.4.2 The ontology management

The ontology requires some content management, as it suffers successive updates throughout time. There are events that become obsolete and, at certain time, will never be shown to the user again. For example, an event that occurs on the 3rd July 2012 at 4 p.m. will never be shown after that day because the users queries concern the events happening now or soon, not the events that already happened. So, we regularly have to erase these events from the ontology in order to maintain query performance (less data to query) and a clean ontology, with no useless data. We achieve this by running a SPARQL query that returns all the events that are no longer needed in the ontology, due to its obsolescence. Obviously the Events that have time expressions from the type *WhenAlways* will never be erased from the ontology, given the fact these events occur at least once every year. The remaining events that have obsolete time expressions from the *WhenDay* or *WhenPeriod* will be deleted from the ontology. In case an Event has two *Whens* and only one of them is obsolete, only that one will be returned by the query and consequently deleted. This means that, in general, the number of events standing definitively on the ontology will tend to an increase, due to the also increasing number of events that occur every year, and that are never deleted from the ontology. The rest of the events are eventually renewed and their number never escalates much.

5.5 Connection to a Prolog server

In order to use the DCG grammar capabilities in our application, it was necessary to create a connection between the Prolog grammar and the Java application. This problem was solved using a client/server architecture. We have a Prolog file that acts as a server, registering all the grammar predicates that are defined. In order to start the server, we must open SICstus Prolog (we are using version 4.0.1) and consult the server file. SICstus Prolog is a Prolog development system that permits us to test the predicates built in the grammar. We have achieved this task by calling the SICstus Prolog process remotely from within our application, using the Runtime Java class. Next, we execute a command to load a Prolog file that, in its turn, consults and loads the grammar on the SICstus environment. Then, we run the main predicate on the Prolog server, also using Java methods. After this, the Prolog server is started and is ready to be queried by the Prolog client. All the DCG grammar predicates are loaded and ready to be executed over the expressions coming from the scraping target file. The Prolog client is located on the Java application side which is a part of the global application server. We are using the J2SE framework. The Prolog client is able to connect to the Prolog server and sends queries containing the predicate with the parameters. For example, for executing the `when()` predicate, it sends *when(String, Result)*, being String the string that is to be evaluated and Result the type of When that will be returned by the predicate. The result is printed in the Java application and could be parsed to extract further information about the temporal aspect of the inspected event. So, every time that the target file (that contains data about the scraped events) is changed, this process restarts, in order to handle the new data. It is possible to send queries multiple times with different parameters and therefore, we can observe the overall results and the accuracy of our grammar. The results we have are very promising. We managed to have a success rate (meaning that the predicate returns an expected result) of more than 75% (approximately) in all the temporal expressions that were evaluated by the grammar temporal predicate (`when()`).

5.6 Insertion of event data on the ontology

The Comma Separated Values document contains event information (scraped previously from the websites) that is read by a Java implementation, using Java library JavaCSV[1]. After being read, the data is transferred to a LinkedList data structure in the Java application. There, each String in the LinkedList is handled, in order to be in conditions of being evaluated by the Prolog predicate, that was previously defined in the Definite Clause Grammar.

There are many predicates defined in the DCG, but we decided to give main focus on the predicate that handles “When” expressions. There are more predicates defined in the grammar, for example one for handling “Where” and one for handling the ticket price expression. After each expression is evaluated by the grammar, the result is parsed by a Java application. In order to parse this result, we implemented a method that scans the result as a String and extracts all the useful information about the event. The resulting expression has the data included in parameters, previously defined on the grammar rules. For instance, an evaluation result of the type horas([hora(14), horaMinuto(16,50)] would have as useful data the two starting hours: 2 p.m. and 4.50 p.m.. Our Java parser would have to extract this data, to then process it and include it on the corresponding *When* class of the event individual. In order to perform this parsing, we implemented methods to read a list of hours or a day interval that has always a start date and an end date. More complex results require more complex data extraction algorithms. This information is required to us so we can be able to create the instance of the cultural event in the web ontology according to the evaluation of its characteristics, made previously by the DCG grammar specific predicates.

The Java application creates an instance of the OWL class *Event* that is filled with its properties, according to the parsed results. The instance is created using the Jena API. Jena is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine. In our case, we use Jena framework to create a new ontology model and edit

[1]http://sourceforge.net/projects/javacsv/

its content, inserting instances and its properties. For all the main predicates in the DCG grammar, we build a parser in Java that recognizes which type of result the Prolog predicates return when evaluating an expression, whether it concerns time, category, location or price statements about an event. Each possible type of result coming from the grammar evaluation will trigger a different way of inserting the event data directly on the ontology. As stated before, this task was executed by developing a piece of Java code that operates directly on the ontology, editing it with new individuals. Many obstacles were found on this part of our work, and therefore, it is where our developing effort was greater. We had to found the best way to insert the event data, through Java code developing, knowing that the variability of expression evaluation results was high. The main challenges that we faced were related to the time expressions interpretation. Problems like knowing which weekdays correspond to a certain interval of days of month, or knowing which day of month is the next Saturday were a major challenge to us. We also had to populate the ontology according to its structure and always respecting the rules of its hierarchy and the requirements of each property. Some time expressions have also implicit information. This leads us to make assumptions about the information, being those assumptions the most probable to be right. Let us take for instance, the expression "Friday at 10 p.m.". If this expression was extracted on a Friday, there are two possibilities: this date refers to the present day or it refers to the next Friday, coming in the next week. As we are not the event data providers, we have no way to guarantee the actual date of the event. Therefore, we decided to consider that the event takes place on the present day in the cases where the extraction was made before 6 p.m.. Otherwise, we consider that the event is going to occur on the next week Friday.

We had to develop various methods to work with the Gregorian Calendar, calculating days of month, weekdays and relations between both. When building these methods, we have to be always aware of the present day and hour. This task is done using the Calendar and Date Java classes and associated methods. Finally, in the Java application, a message with the success rate is printed. This rate let us know in which cases the String was not evaluated, returning an error message. So, it is possible to adapt the grammar to those special cases, increasing the success rate.

5.7 Data handling and modification

We have a set of conditions defined to decide if a cultural event has all the elements to be included on the ontology, so it could then be queried. We must ensure that all the ontology individuals are considered in a query and no events are left behind in a response, due to lack of information on those events' properties.

5.7.1 Coordinates Extraction

The events that are scraped from the web have a location associated. In order to provide more information about every event to the user, we focused on obtaining the geographical coordinates of each location. For that, we use Google Geocoding API service, along with a solution that we designed. Google Geocoding service API has certain hurdles in terms of massive usage. This service is subject to a query limit of 2,500 geolocation requests per day. Moreover, the number of requests to the service has a limit per second and per IP on the network. In order to bypass that obstacle, we decided to maintain a database with the Address – Coordinates relation of every location. This approach brings the advantage of not making requests every time a location is extracted from the web, but only when that location is new and is not in our mapping structure. Since it is not mapped yet, we do a geolocation request and add the result to our structure, for further use. This solution makes the event classification process faster (less requests), because the successive geolocation requests require a time interval of around one second between 10 to 15 requests. So, the lesser requests are done, the better classification time we get, in order to then populate the ontology. Nevertheless, location is not a mandatory requirement.

The geolocation obtaining process is made on the server side, in a class that we called Geocoder. This class accesses to the Google Maps service through an URL. That URL contains the location that was extracted from the scraping target file. The Google Geolocation service returns a response which contains an XML file with the results of the geolocation. The main tag is called `<GeocodeResponse>`. This XML contains all the results that the Google Geocoding service obtained. This means that a single search can have multiple addresses (and corresponding coordinates) as a result. Our class Geocoder selects only the first result as the correct address and location, because the first result generally is the one that we are searching for. Each one of these results is located on the field `<formatted_address>`. The geographic coordinates are located inside the field `<location>`. This way, we are able to extract latitude and longitude data of a certain location. When in a location expression search, no results are returned, we redo the search with the expression “Lisbon Portugal” appended on the location expression, to increase the matching probability. If yet no results are obtained by the Google Geocoder, a default coordinate set is assigned to the location. These coordinates are the corresponding coordinates of the Search “Lisbon Portugal” which are latitude 38.723827 and longitude -9.13977. If an event lacks the information about its location (void value previously scraped from the web), this same coordinates default value is attached to that event. This event will be put into the ontology with a “Lisbon Portugal” corresponding coordinates value as its location. With this solution, every events have corresponding geographic coordinates, even if there are no results from the Google Geocoding service, when we search for the extracted location. As we explained before (see Section 1.2), we expect the great majority of events that are scraped from the web to be located in the Lisbon area. So, this solution is the best for our case, as we assume that an event that lacks location data, is located in Lisbon.

5.7.2 Name, Date and Category handling

In order to enter the ontology, an event must comply with some requirements. First, an event must have a name, so it could be presented on the results list. The String containing the name is modified, so it can be inserted on the ontology without any inconvenience (unwanted characters and symbols that may cause issues when creating the ontology individual or that may produce legibility difficulties, are replaced). This goes to any expression that is to be inserted on the ontology. Still on the name handling, it is also inspecting the name of the event that we can understand if an event is a repeated one, and therefore not include that event again on the ontology, avoiding making an unwanted duplicate on our system. The name of the event could also match a category name or have a void value. In those cases, the events are discarded, too. Also, all Datatype values (OWL built-in types) that are introduced on the ontology, such as *xsd:date*, *xsd:string* or *xsd:time* values, are assured to have the right format, so any ontology inconsistency problem is discarded. As we considered the temporal aspect of the cultural event the most important in our application, it is a requirement to all the events stored on the ontology. Any event that lacks date information will be discarded and, consequently, will not be inserted on the ontology. The events which temporal expression could not be processed by our grammar predicates, will as well be discarded. Another requirement that events must fulfil is having a valid category. There are several event categories that we defined and our grammar has predicates to validate those categories (see Section 5.3.2).

5.8 Query Transformation Process

The natural language questions that are made by the user (through text input) could be based on the date (“What is up today?”) but could also have a time restriction, as for example, an hour in the day (“What is up today at 12 am?”). This last question (after being transformed into a query) returns the events that occur on the present day at 12 am. The query expression can also include event category restrictions: “What can I watch in Cinema today at 12 am?” or “What dancing events are around?”.

The conversion of these questions to SPARQL queries is made on the server side. There, it is transformed in a SPARQL query, according to a keyword-based translation process. The user natural language question is scanned and if it contains some pre-defined keywords, like “today”, “now”, “afternoon” or “night”, a pre-set SPARQL query is executed on the ontology. The question may also contain a time restriction (corresponding to a number from 0 to 23) or any event category restriction (Cinema, Concert, Party). There are some expressions that are considered as equivalent to category restrictions, as for instance, plural forms (Concerts, Parties, Galleries) or expressions directly related to a certain category (for example, movies, music, drink). In cases that the question has multiple restrictions, a query that combines all those restrictions will be run. As we adopted a keyword approach, the context of the question is ignored. This approach could easily be improved, by adding new keywords and expression variations (abbreviations, orthography errors and so on). The resulting SPARQL query always concerns the three types of *When* existing on the ontology. The results of the query "What galleries can I visit today?" will return all the art expositions that happen on the present day, whether they have a *WhenDay*, *WhenPeriod* or *WhenAlways* class associated. This fact causes the resulting SPARQL query to have three different sections, one for each *When* class type. These three sections are united through the SPARQL (and SQL) operator UNION. In summary a DL Query to return all the art expositions (or art galleries) existing on the 22nd July 2012 would be as follows:

```
hasWhen some (WhenPeriod and (hasStartDate some
date [<="2012-07-22"^^date]) and (hasEndDate some
date [>="2012-07-22"^^date]) and (hasWeekDaysHour some
(WeekDaysHour and (hasWeekDay value Domingo)
or (hasWeekDay value Todos))))
or hasWhen some (WhenDay and hasDayHour some
(DayHour and hasDay value "2012-07-22"^^date))
or hasWhen some (WhenAlways and
hasWeekDayMonthOrder some (WeekDayMonthOrder
and (hasOrder value Quarto and hasWeekDay value Domingo)
or(hasOrder value Todas and (hasWeekDay value Domingo)
or (hasWeekDay value Todos))))
```

Analogously, the correspondent SPARQL query would be:

```
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX ode:<http://www.co-ode.org/ontologies/ont.owl#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX uri:<http://
www.semanticweb.org/ontologies/2011/8/WhatsUpOntology.owl#>
SELECT DISTINCT ?event
WHERE {{
?event ode:hasWhen ?quando.
?quando ode:hasStartDate ?startDate;
ode:hasWeekDaysHour ?wdh;
ode:hasEndDate ?endDate.
{?wdh ode:hasWeekDay ode:Domingo}
UNION {?wdh ode:hasWeekDay ode:Todos}.
FILTER (?startDate <= "2012-07-22"^^xsd:date &&
?endDate >= "2012-07-22"^^xsd:date ).}
UNION{
?event ode:hasWhen ?quando.
?quando ode:hasDayHour ?dayHour.
?dayHour uri:hasDay "2012-07-22"^^xsd:date.}
UNION{
?event ode:hasWhen ?quando.
?quando rdf:type ode:WhenAlways;ode:hasWeekDayMonthOrder ?wdmo.
{?wdmo ode:hasOrder ode:Todas; ode:hasWeekDay ode:Domingo.}
 UNION {?wdmo ode:hasOrder ode:Quarto; ode:hasWeekDay ode:Domingo.}
 UNION {?wdmo ode:hasOrder ode:Todas; ode:hasWeekDay ode:Todos.}}
}
ORDER BY ?event
```

As we can see, in SPARQL language, the variables are indicated with a ? symbol. We notice also that, in order to make queries concise, SPARQL allows the definition of prefixes and base URIs in a fashion similar to Turtle [33]. In this query, the prefix `ode` stands for “http://www.co-ode.org/ontologies/ont.owl#”. All the prefixes are defined in the first lines, using the keyword PREFIX.

Once this conversion process is completed and we have a natural language question translated to a SPARQL query, we can execute that query on the ontology and obtain the returned results, in response to that query.

Chapter 6

Mobile Application Description

Once we finished all the server back-end workflow and the server side network communication interface, we were ready to start developing the client side of our application. The client application is located in each one of the Android devices that want to connect to the remote server and make questions, searching for cultural event activities. This chapter explains how this development process occurred.

6.1 Why Android

Android Mobile OS has currently a smartphone market share of more than 50%, which means that most smartphone devices run under the Android Mobile OS. Also, most Android applications are written in Java and we are more comfortable with Java Programming language than any other programming language. Moreover, the server side of the application is also written in Java and this helps having a more compatible programming environment and a very consistent application.

The tools to develop Android software are easily reachable because Android has a free and open-source license. For that, we use Eclipse, as it is the officially supported integrated development environment (IDE), using the Android Development Tools (ADT) Plug-in. There is also a huge support from the Android developing community, which is a strong reason to use this mobile operating system, as new libraries are always emerging and code troubleshooting and solution development becomes easier that way.

We initially started testing our client side application by using an emulator (provided on the ADT Plug-in). With this emulator we can simulate GPS coordinates receiving and an Internet connection with a local server (via TCP sockets). The client emulator connects to the server that is located in localhost computer (this computer is the host to both server and client). Nevertheless, we cannot guarantee that our application really works on a real situation (server and client in different devices). Also, we wanted to test a remote connection to the server to see if our communication implementation was working efficiently. An Android device was provided to us, in order to test the client application and evaluate the GUI appearance and usability. The model of our smartphone is Sony Ericsson Live with Walkman [1].

Figure 6.1: Our testing Android Device (from Sony Mobile website)

With this device we can test the application in a real scenario, with real GPS coordinate extraction and a real screen display. The device's screen size is 3.2 inches with a resolution of 320x480. These dimensions are medium size for smartphones.

The specifications of this smartphone are presented on Table 6.1:

Specification	Details
Internal phone storage	Up to 320MB user-accessible memory
RAM Memory	512MB
CPU	1 GHz Qualcomm 8255
Display	3.2", 320 x 480 pixels 16,777,216 colour TFT
Dimensions	106 x 56.5 x 14.2 mm
Weight	115 g
Native Android Version	Android OS v2.3 Gingerbread

Table 6.1: Our testing Android device specifications (from Sony Mobile website)

6.2 Screen orientation adaptability

Our application is adaptable to the smartphone screen orientation. There are two possible screen orientations: landscape and portrait. The first one is engaged typically when the user holds the phone horizontally, with both hands. The second one is engaged when the user holds the device like a cellphone, with only one hand. The accelerometer embedded in smart devices is typically used to align the screen depending on the orientation of the device, i.e., when switching between portrait and landscape modes. This capability provides great opportunities to create better user experiences because it offers an additional

[1] www.sonymobile.com/br/products/phones/live-with-walkman/gallery/

layout with a simple turn of a device, and without pressing any buttons. Therefore, we had to create create an alternate layout that is displayed when the user changes the device orientation. We also provide in our application three different version of every symbol, in order to make our application adaptable to any screen size (smartphone or tablet). The three different sizes are small, normal and large and the three screen densities are ldpi (for low density), mdpi (medium) and hdpi (high). This way, we can support multiple screen sizes.

6.3 Time-zone definition

We assume that the user is always on our target city perimeter, as our application geographical scale concerns only a city (not a worldwide scale) and all our application content is location dependable. For those reasons, we did not concern about time-zone adaptations on the event dates, depending on the user location, because there is no space to ambiguity on the time scale. Our application could be easily adapted to another city, by extracting information about events occurring in that city. The time-zone that we consider is always the local time-zone of our target city. Therefore, our application was designed to be used locally (the user being on the same city in which the events are located)and never on another city. This makes the time-zone adaptation question irrelevant.

6.4 Application specifications

Our application can run in multiple Android versions, namely version 2.3 (GingerBread) and the above versions. Our application is developed for Android Operative System and its GUI is divided into multiple activities (Android screens). The user navigates the application by exploring the different activities during his experience.

Our application depends on a Wi-Fi or 3G/4G connection to work properly, so the server could be accessible. It also requires a GPS connection to retrieve geographic location data from the device. Any user with an Android smartphone can connect to the server, running the client application. If the server is not accessible when the user starts the application, the search menu appears, but no buttons are visible. This solution prevents errors that could occur, if the user could make questions without having a connection to the server. A warning appears to advert the user of the connection failure. The search menu interface will only show up if a connection to the server is established. In case the connection is lost in the middle of a search or on the results menu, a warning message stands on the screen bottom, indicating that the user has currently no connectivity to the server, as we explain in Section 6.9. We also show the layout state when there is no connection to the server, on Figure 6.8.

6.5 Calculating distance between event and user

Once the server obtains the user location and the coordinates of the event, a method to calculate the distance (in meters) between two points is called. This method uses a formula (called the haversine formula) that is for calculations on the basis of a spherical earth (ignoring ellipsoidal effects) – which is accurate enough for most purposes. The value returned by the method is called the Great-circle distance between two points, formula to calculate the great-circle distance between two points – that is, the shortest distance over the earth's surface. This value is obtained ignoring hills or any other obstacles on the course.

The Haversine formula is represented on the method the following way:

$$\texttt{a = sin}^2\texttt{(}\Delta\texttt{lat/2) + cos(lat1).cos(lat2).sin}^2\texttt{(}\Delta\texttt{long/2)}$$
$$\texttt{c = 2.atan2(}\sqrt{a}\texttt{, }\sqrt{(1-a)}\texttt{)}$$
$$\texttt{d = R.c}$$

Where R is earth's radius (mean radius = 6,371km); lat1 and lat2 are the latitude values and long is the longitude values. c is the angular distance in radians, and a is the square of half the chord length between the points. d is the final distance between the two points.

This equation can be solved for d and stand as:

$$d = 2.r \arcsin\left(\sqrt{\sin^2(\frac{\phi2-\phi1}{2}) + \cos(\phi1)\cos(\phi2)\sin^2(\frac{\psi2-\psi1}{2})}\right) . \qquad (6.1)$$

The haversine formula remains particularly well-conditioned for numerical computation even at small distances – unlike calculations based on the spherical law of cosines. The versed sine is

$$1 - \cos\theta$$

and the half-versed-sine is

$$(1 - \cos\theta)/2 = \sin^2(\theta/2)$$

as used above.

6.6 Starting What'sUp Android application

In order to start running Android **What'sUp** application, the user must touch on our application logo on the Android interface (Figure 6.2).

After touching the application logo, the user is taken to our application environment, as the **What'sUp** Search menu shows up. The user can then start to make a question to search cultural events.

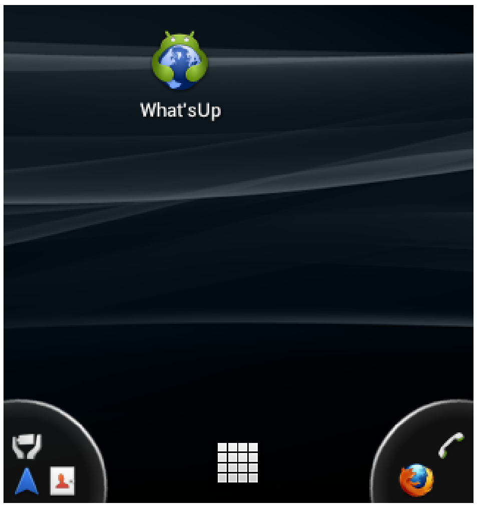


Figure 6.2: Our **What'sUp** application logo on the Android main screen

6.7 Client/Server communication

When the Java server is started, he remains constantly listening to a port, waiting for new clients to connect. When a client connects to the server, the latter creates a dedicated Thread to handle communications with the client.

The client pool has a capability of 30 clients. This pool is acceptable for a prototype version (which is our case) and is according to the number defined in the objectives (see Section 1.2). However, we can increase this value if we consider a more advanced platform to handle the client connections. The server was programmed to be reliable and stable, once it is started. The clients do not need a login system to communicate with the server, so the communication is much more accessible to make. Every users are equal and have the same privileges. If the client wants to close the application, he can do it at any moment, during the his interaction with the application, using the Android device Home button or the Back button if the user is in the Search menu (see Section 6.8)). When the user does so, and before the connection is closed, the client application sends a message containing ⋆⋆ to the server, so the server can kill the thread associated to that client and restore the memory that was previously requested for the client, adding one more free space to the client pool.

When requesting for an event searching result, the user does not need to specifically state his location, given the fact that it is automatically extracted from the GPS provider system. This location information is sent to the server (after being obtained), together with the user natural language question and the geographical radius restriction (see Section 6.8). When the user presses the button to make a search for events, a message with all the information the server needs is sent. The format of the message which is sent from the user to the server is the following:

```
Radius Distance+Latitude_Longitude*Question
```

On the server side, all these three elements (radius distance, coordinates and question) are split and evaluated. The question is translated to a SPARQL query and run onto the ontology. The set of individuals resulting from that query is transformed in a set of objects of the Java class Event (that implements the interface Serializable). The class Event is a custom Java class that was made to fit the requirements of a data structure to be able to be sent across a network connection. In our case, this information will be sent from the server to the user Android device. This class has all the attributes that an event must have, like location, name, date, category and distance from the user.

So, when all the results from the SPARQL query are included on this Event structure, we are prepared to send the results to the Android Client device. When the client application receives the list of Events, it displays it to the user. That structure is stored in the client memory, without it to ever having to contact the server again for that search. When the client makes a new search, that structure is erased and a new structure is received. The radius distance indicated by the user serves as a restriction to filter even more the results. Using the GPS coordinates of the user and the event coordinates, the Server calculates which events are within the requested range. Only the events that are within the radius are included on the final results list, that is then sent to the user as a response to his natural language question.

6.8 Radius restriction

The user can also restrict the search results, by choosing the geographical search radius around him. The search results are then filtered and only the events within the desired radius (500 or 2000 meters, for instance) are displayed. The user can do this by selecting one of three radio buttons that are displayed on the Search screen layout. If the user opts by not defining any radius restrictions and does not select one, there is already a default value (corresponding to “No restrictions”). This value sets no geographic radius restrictions on the search and triggers a normal search. This feature is very important because it allows the user to filter the results, according to his mobility capabilities at the moment. If the user does not want to walk more than 2000 meters to go to an event, he may set a distance threshold, and therefore, make a better decision with fewer options.

This also enhances the system performance, because it presents fewer events on the interface and processes less information. After making a question on the Android application and selecting the geographical radius, the user presses the “Find Events” button (illustrated on Figure 6.3), to search for results.

The results are presented according to their date and location. The resulting events that appear in the applications GUI (as a response to the user queries) are sorted by proximity to the mobile device GPS position. This way, the results list top events are the nearest from the user. The events that lack coordinates information are put in the bottom of the

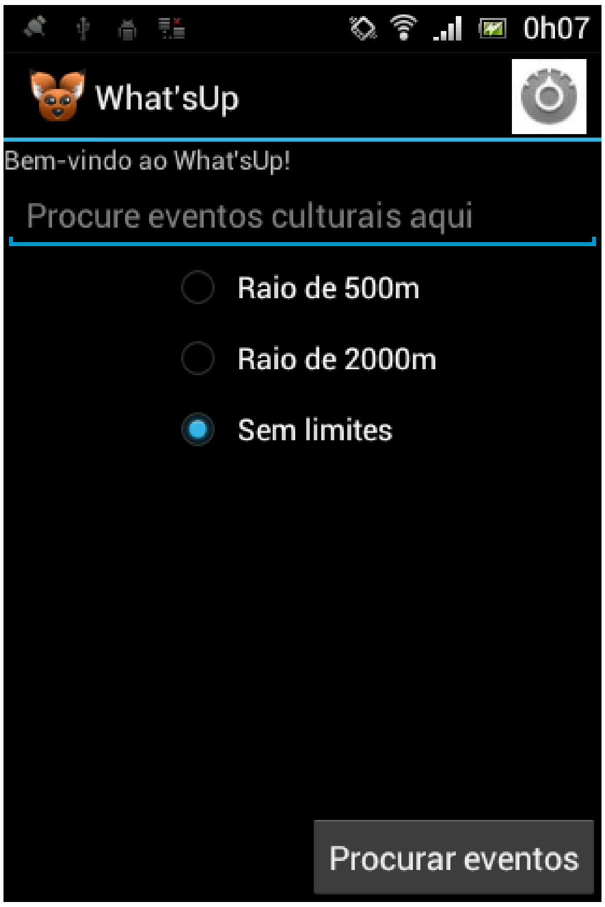
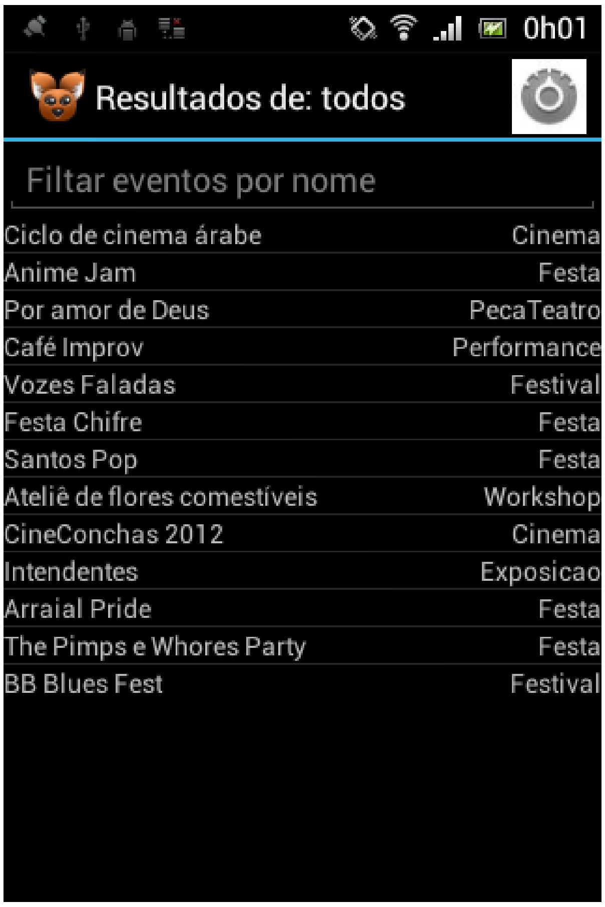


Figure 6.3: Search menu layout

list. If the results are in a large number and all the device screen is full with results, a scrollbar appears on the screen right that shows the relative amount of events that the user can see at once. If that bar is small, the number of events that the user sees on screen is considerably small, comparing to all the result count. This layout has a feature that could make the user find an event more efficiently. It is a filter bar, which is placed above the list of results. It has a hint inside the filter bar which reads “Filter events by name”. The user can filter events by name inside the results list. After, doing this (if needed), the user may choose the event that he wants. This feature is very helpful when the user has an event in mind and wants to search for it or even to restrict the results, in case there is a very large number of events. The results list is dynamically updated as the user types the filtering expression. Each result has a description of the name of the event (on the left) and its corresponding category (on the right), as depicted on Figure 6.4. This result disposal layout helps the user knowing which types of events were returned. We find that the event category is the most important aspect after the event name in this situation. The user may choose the event that he wants. This way, the user already knows what type of event he is exploring when he selects it.

After selecting one event on the results list, the event details screen shows up (see

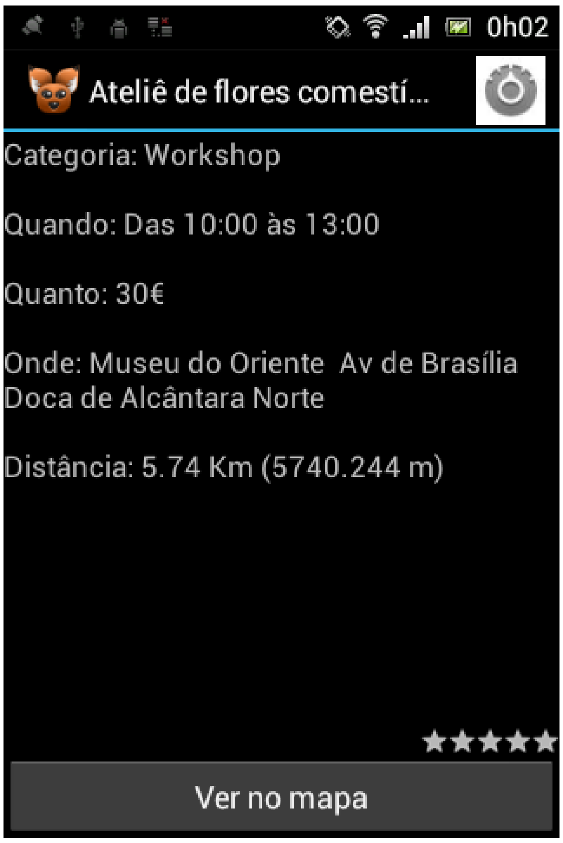


Ciclo de cinema árabe	Cinema
Anime Jam	Festa
Por amor de Deus	PecaTeatro
Café Improv	Performance
Vozes Faladas	Festival
Festa Chifre	Festa
Santos Pop	Festa
Ateliê de flores comestíveis	Workshop
CineConchas 2012	Cinema
Intendentes	Exposicao
Arraial Pride	Festa
The Pimps e Whores Party	Festa
BB Blues Fest	Festival

Figure 6.4: List of results returned by the server, after the user natural language question

Figure 6.5). The details of an event include its name, category, date, location, price and distance from the user (if GPS coordinates are not available, this value is set to the maximum value in the Java Integer type range). The length measurement units that we use are Kilometres and also Meters. The price information may have a "Not available" value, if its original value (during the scraping task) was that or if it had no price reference at all. We preferred to inform the user that the price information was not provided than not including the event on our database. The name of the cultural event appears on the top of the screen so the user can easily identify the current cultural event that he is viewing.

The distance of an event from the user current location is calculated on the server side, as explained in Section 6.5. On the event details screen, the user can choose to see the location of the event on the map, pressing the "See on Map" button, on the bottom of the screen. Our application uses the Google Maps service for providing the event location on the map or even point an itinerary for the event (either the user chooses to get to the location by foot, by car or by bus), pressing the Google Maps button "Directions", on the bottom of the screen. This task is accomplished by connecting our application to the Android Google Maps service and searching for the location that was previously obtained, through the Google Geocoding API service (see Section 5.7.1). The map will appear and



Categoria: Workshop

Quando: Das 10:00 às 13:00

Quanto: 30€

Onde: Museu do Oriente Av de Brasília Doca de Alcântara Norte

Distância: 5.74 Km (5740.244 m)

Figure 6.5: The event details screen

the location spot is marked with a Google maps pin (red balloon marker).

Once on the Google Maps interface, the user can inclusively use the Street View service to get a better panorama of the location of the cultural event in question. This service provides (on most cases) a 360 degree photography view of the location, which permits to have a better preview of the street, so when the user actually arrives on the location, he can easily recognize the place.

6.9 Feedback to the user

In our Android application interface, we did a great effort to always keep the user informed about the state of the application and about the transitions between states. Whenever the user interacts with the system, he is given all the feedback he needs. On Android environment, this feedback could be provided in multiple ways: text messages, sounds, changing colours and more. We make use, of course of the basic feedback that the custom Android elements (like Buttons, Windows and Lists) already provide. In addition to that, we decided to use two types of text feedback: text messages included on Toasts, which are warnings that appear on the screen during a limited period of time and permanent text messages, which remain on screen. These last ones are used on more important situa-

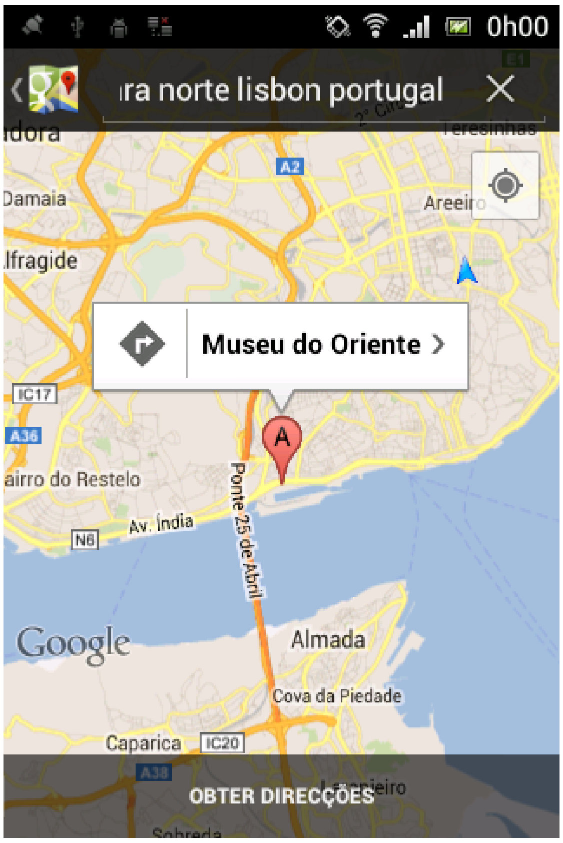
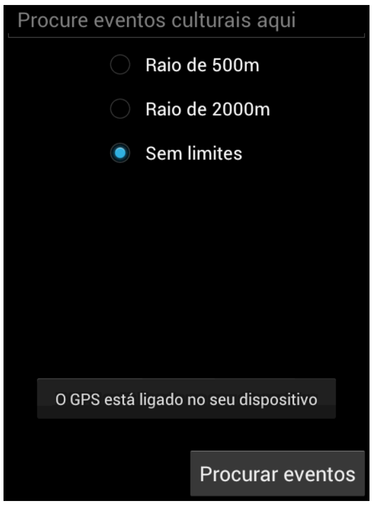


Figure 6.6: Location of an event displayed on the map

tions, when the user really needs to read the information, due to its importance, in order to successfully interact with our application. An example of Toast feedback is when, after starting the application and enter the search screen, the user is warned that the GPS system on his device is turned on. Figure 6.7 illustrates the scenario. This Toast message disappears moments later.

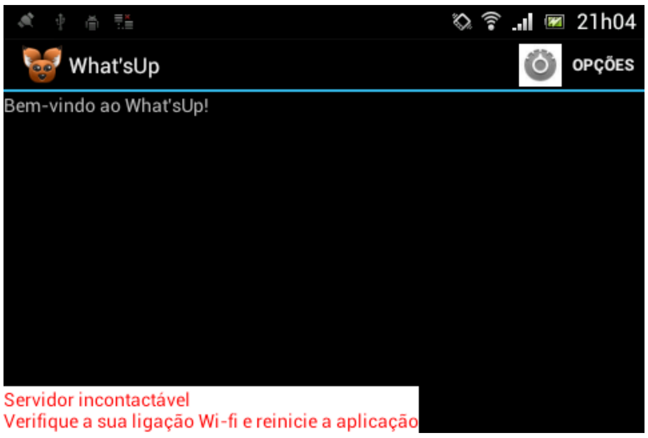

Procure eventos culturais aqui

- ○ Raio de 500m
- ○ Raio de 2000m
- ● Sem limites

O GPS está ligado no seu dispositivo

Figure 6.7: Toast message giving feedback to the user about the GPS receiver state

The other type of feedback can be exemplified in a scenario where the server is offline and the application can not stablish a connection. In this case, the user will not be able to make questions, due to not having a connection to the server. Therefore, we present a message that stands visible to the user until a connection is established. This message has a white background and a red font, in order to have high visibility. It is illustrated on Figure 6.8, which features a landscape view (check Section 6.2).



Bem-vindo ao What'sUp!

Servidor incontactável
Verifique a sua ligação Wi-fi e reinicie a aplicação

Figure 6.8: Warning message giving connection feedback to the user

We also give, always when possible and admissible, hints to the user, so it is easier to the latest to know what to do next. One of these examples is when the user is returned the list of results and a Toast message appears, suggesting him to filter those event results by name, if he wants to restrict them even more, using the filter bar (see Figure 6.4).

6.10 Performance Comparison: DL Query vs SPARQL

We decided initially to adopt DL Query as the query language to use in our application. This query language should be familiar to users of Protégé through the “DL Query” tab. We used the Java OWL API to programmatically run the DL Queries and return the results. Although, we did some charge and performance tests and this implementation proved not to be scalable in some cases, particularly when the query complexity was raised. We ran the same queries over our ontology directly on Protégé and the performance results were the same, which means these results are not related with the Java implementation. We had to find an alternative to this language. SPARQL was a keen choice because it is a standard query language for OWL 2 and provides better performance results than DL Query. Moreover, SPARQL has a query format similar to SQL, which helps us in our case, because we have some experience with SQL querying.

We present a table (Table 6.2) with the performance records of our application at multiple moments, during the whole data extraction, integration and storage: the Event Classification Time (ECT) measures the performance at evaluating time, category and name expressions and processing location data. Ontology Population Time (OPT) is the time spent to populate the ontology with the previously processed information. Query Analysis and Translation Time (QATT) stands for the performance of the keyword processing and translation of the user query to a *SPARQL* or *DL Query* query. Query Execution Time (QET) is the time spent during the proper query execution until the system returns a response. Finally, Events Organization and Sorting Time (EOST) is the time that the server spends gathering the information of the results and sorting them by distance to the user.

		Number of Events (DL Query)			Number of Events (SPARQL)		
		10	100	1000	10	100	1000
Task Times	ECT	0,278 s	1,75 s	27 s	0,278 s	1,75 s	27 s
	OPT	0,305 s	0,9 s	1,8 s	0,305 s	0,9 s	1,8 s
	QATT	0,016 s	0,073 s	0,417 s	0,051 s	0,089 s	0,502 s
	QET	0,015 s	0,281 s	39 s	0,019 s	0,015 s	0,063 s
	EOST	0,017 s	0,135 s	0,5 s	0,05 s	0,145 s	0,66 s

Table 6.2: Performance times comparison for multiple tasks

In order to obtain the events that are processed in this experience, we used a macro that executes a scraper, which in its turn, is prepared to extract events from multiple Lecool magazine issues. We scraped the events from a sufficient number of issues, so we could have 1000 events. This is equivalent to 1000 scraped lines to the CSV file, each one containing information of an event. This scraper was used also to inspect the type of expressions that we could expect from Lecool (see Section 5.2).

The natural language query that we made on this test was the same for both SPARQL and DL Query languages. In this case we input “What is up today”, which is a relatively complex query, from our possible query set. The set of events that we worked in this test was a fixed set. The set of 10 events was included on the set of 100 events and the same was done with 1000 events. This initial number of events was eventually reduced, as the workflow of tasks evolved. This is caused by the events that are discarded, whether before or after being evaluated by the grammar, and never get to be inserted on the ontology, for example. Despite that, these filtering happens the same way in both cases, as the set of events is equal on every case. We made each test two times and then calculated the average value of those two measurements. These testing conditions guarantee that the obtained values are trustworthy. also the same for each number were also the same

We can easily verify that the amount of time spent to process ten events is less than the time needed to process 100 or 1000 events. Event Classification Time and OPT times are the same using SPARQL and DL Query because their corresponding tasks are executed before any contact with query languages. The comparison only differs from the moment that we translate the natural language query to a language specific query. The Query Analysis and Translation Time records are favourable to DL Query side, but the time advantage comparing to SPARQL is relatively low (0,417 seconds for 1000 events against 0,502 seconds). We can see a great difference in performance on the Query Execution Time values, in which DL Query takes almost 40 seconds to process a 1000 events complaining query. On the other hand, SPARQL does the same in 0,063 seconds. These values prove that SPARQL handles much better the scalability issue. We could not trust on DL Query language in our application due to these values. The user would quickly be aware of the poor performance on the query execution and it would be a major issue for our application quality, as we want it to be efficient. An application that is inefficiently is not used. On the Events Organization and Sorting Time, there is a little gain on the DL Query side, but it does not compensate the dropping difference on the Query Execution Time.

These results prove that DL Query is much more expensive in terms of performance than SPARQL. In fact, this could be explained by:

- Reasoning with DL Query can be slow, particularly if we have a considerable number of individuals
- DL Queries do not allow for closed world negation (which is supported by SPARQL filters)
- Not all reasoners support DL Queries
- DL Query is more effective for quickly test definitions of classes or check for class membership of arbitrary descriptions without having to create named class placeholders. It is not designed to handle a large number of individuals.

6.11 System architecture

On Figure 6.9 we can observe all our application workflow, since the back-end, where the event data scraping occurs, until the front-end, where the user interacts with the application. Throughout this workflow, the event data suffers a number of transformations that are, nevertheless, transparent to the user. After all the phases of event data extraction, modification and evaluation, the final state of the event data is reached when this data is stored on the web ontology. The cultural events data is finally ready for being queried. When the user makes a natural language question, the results of that question are returned to the Android application layout as a list of cultural events.

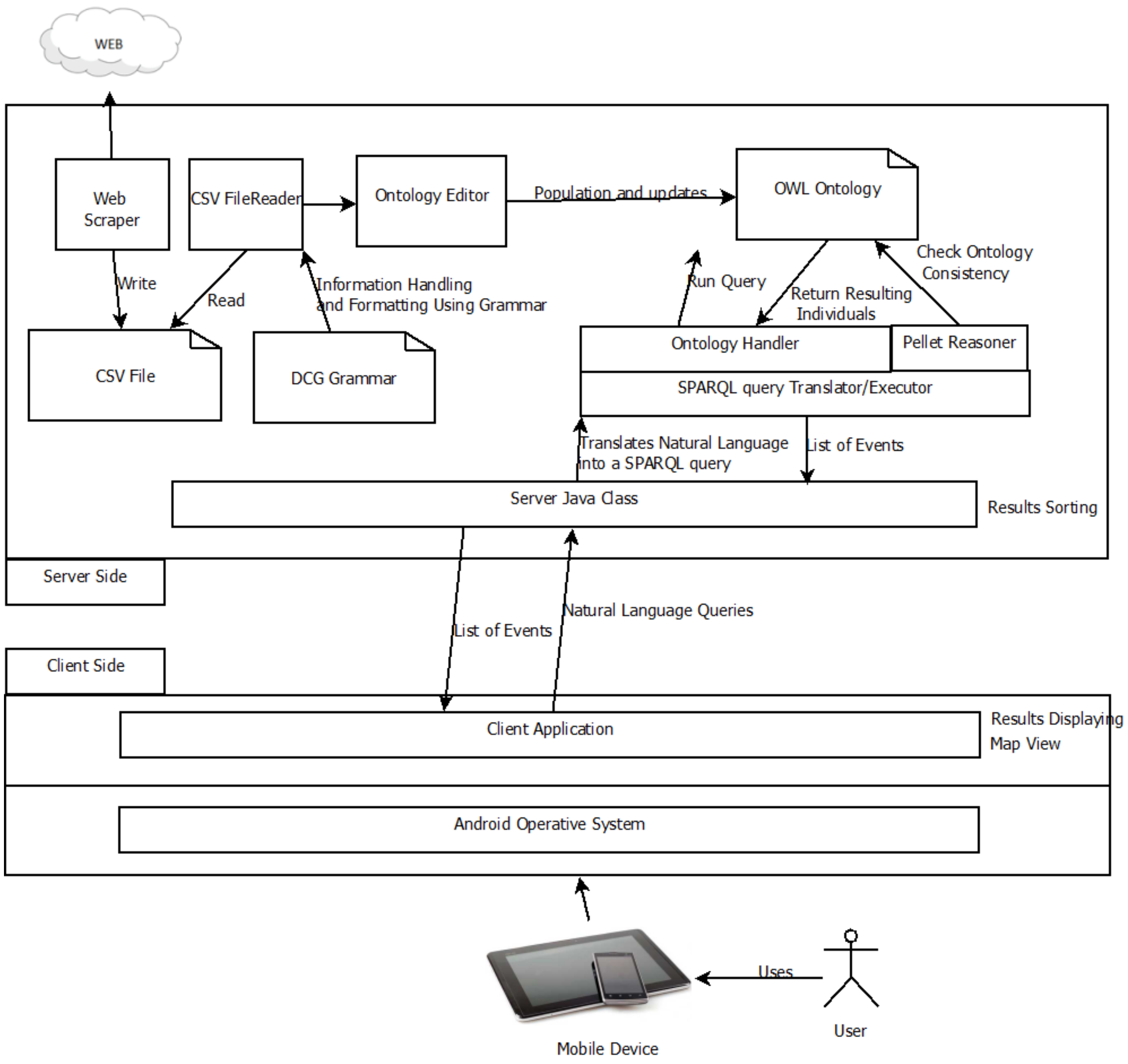


Figure 6.9: The **What'sUp** client-server application detailed architecture

Chapter 7

Conclusion

7.1 Considerations

Facing the objectives defined in Section 1.2, we can conclude that all of them were accomplished with success. We were able to develop an application that corresponds to our initial expectations. All our fundamental modules (like web scraper, the web ontology and the DCG grammar) have a performance level above our initial requirements. We initially agree to try to publish one scientific paper with material from this project. We made an attempt in a relatively advanced phase of the project, so we could show our application capabilities with more details and give more precise explanations. We elaborated a full paper (with 12 pages) and it was accepted for the INFORUM 2012 conference, held in Costa da Caparica, Portugal. This article was written from 21st May to 27th June (including final corrections and adaptations). This additional work was added to the project planning and delayed our dates a bit. Despite that, during the development of this article, we managed to gather a large amount of useful information and researching material. Most of that information was reused (and in some cases improved) for this final thesis report, which means that the articles helped us going forward on our work. The acceptance of our article also comes to grant that our project is interesting, has quality and has a great potential. However, the reason to this delay was not only this. Some modules of the application exceeded their duration expectations, due to implementation issues or adaptations. We made this modifications always thinking that it would make our application more effective and, though, provide a service with more quality to the user. For instance, our ontology structure was modified and rebuilt so our application could return more accurate, refined and reliable responses to the user. Nevertheless, the task duration planning was, in general, respected and the project objectives were successfully accomplished.

7.2 What’sUp Project Conclusions

The main objective of **What’sUp** is to orient and to keep the user aware of the ongoing cultural events in the user surrounding area, without the need to specifically declare all the details about the event and without having to manually provide location information. One of the main elements in this project is the web ontology, which was built to store information about four main properties of cultural events: “What”, “When”, “Where” and “How Much”.

We developed a context-aware mobile recommendation system, named **What’sUp**, to support serendipitous discovery of city events and leisure activities. **What’sUp** Project brings an approach with many potentialities, giving special priority to events that are happening at the moment and in places as near as possible from the user. **What’sUp** project comes towards the objective of bringing the best of these two worlds (Semantic Web and Mobile applications) together, relying on information extraction and transformation techniques and knowledge representation methods. It is an event-driven application that guides the user throughout the city activities, making use of the environment data, like location and time. Our application makes use of web scraping, natural language processing (with a DCG grammar) and semantic knowledge manipulating and querying to respond to the user needs. The user natural language questions, which are made by text input in the application GUI, must be evaluated and transformed into a SPARQL query that is run in the RDF database (which may also be called RDF store, i.e., our OWL ontology). We built a web ontology in OWL 2.0, for representing cultural events and leisure activities in a city in order to promote a semantic web standard. Our event data is periodically extracted from Lecool e-magazine and Myguide websites. In order to extract events from these sources, we rely on web scraping and natural language processing. We were able to integrate digital content about cultural events from different origins. Our preliminary evaluation suggests that **What’sUp** can be useful for users that want to discover what is happening “here and now”, fostering spontaneity.

Expression evaluation is also a complex process that returns very satisfying results, as the recognizing process is effective and relies on simple rules. Also, the performance of the ontology population is very satisfying and every individuals and properties are filled in as we expected.

All the user interaction with our system occurs on the client application, which is installed on the user’s Android device. The user is able to search for nearby events happening on the current day, for example. That query will return all the possible events that can be visited in the specified date and on the user area surroundings.

In summary, What’sUp project is a very promising approach in what concerns the dynamic search of cultural events in digital cities, using mobile technologies.

7.3 Future Work

In the near future, we plan to expand the information extraction to other sites with city cultural and leisure offers. We can also gather more information about locations (opening days, closing days, children allowance, among others) and categorize the tourism places on the ontology (for example, the classes *Museum*, *Garden*, *Theatre*, *Casino* and so on), being each of them sibling classes and subclasses of the OWL class *Location*. This would give the user more information about the location of the event. We should also concentrate on enriching our application capabilities with the introduction of a user profile concept, as this feature is very important nowadays. It would allow the user to like an event, share it with friends and to have preferences that would restrict the results display. The query translation approach can also be improved, by adding new keywords and expression variations like abbreviation, orthography errors admittance and correction (using a spell checker) and context-related expressions processing. Finally, the context-free grammar expression recognizing power can always be improved by adding new rules, so more expressions could be recognized. We can inclusively conduct more field study with groups of users to evaluate the effectiveness of **What'sUp** in supporting serendipitous discovery. Finally, there is one way to improve our data correctness trust, not having to handle data so massively. We can contact the on-line agenda providers and propose (or discuss) a data structure (like a web form) in which they publish their event data and motivate them to update their event database regularly. Having an agreement on this subject, we can offer for exchange more visibility to the events that they are featuring, through our mobile application, that is accessible to everyone who has it installed on their mobile device. Plus, we have more reliability on the content because it is published according to the data structure format and restrictions. However, we can alternatively improve our own digital content integration methods.

Chapter 8

Bibliography

[1] Tintarev, N., Flores, A., and Amatrain, X.: Off the beaten track - a mobile field study exploring the long tail of tourist recommendations. In Proceedings of the 12th International Symposium on Human-Computer Interaction with Mobile Devices and Services (MobileHCI) (2010)

[2] Kruger, A., Baus, J., Heckmann, D., Kruppa, M., and Wasinger, R.: Adaptive mobile guides. In The Adaptive Web, Brusilovsky, P., Kobsa, A. and Nejdl, W. (eds.), vol. 4321. , pages 521 – 549, Springer (2007)

[3] Zheng, V.W., Zheng, Y., Xie, X., and Yang, Q.: Collaborative location and activity recommendations with GPS history data. In Proceedings of the 19th international conference on World wide web, pages 1029–1038, ACM (2010)

[4] Bellotti, V., Begole, B., H. Chi, E. et al.: Activity-based serendipitous recommendations with the Magitti mobile leisure guide, In Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08, pages 1157-1166

[5] Forsblom ,A.,Liikkanen, L., Nurmi, Aman, P.: Out of the Bubble - Serendipitous Event Recommendations at an Urban Culture Festival, IUI'12, ACM (2012)

[6] Knublauch, H.: Ontology-Driven Software Development in the Context of the Semantic Web: An Example Scenario with Protégé/OWL,1st International Workshop on the Model-Driven Semantic Web (MDSW2004)

[7] Siricharoen, W.: Learning Semantic Web from E-tourism. In Nguyen , N. T. (eds.): KES-AMSTA, LNAI 4953, pages 516-525, Springer (2008)

[8] W3C OWL Working Group.:OWL 2 Web Ontology Language Document Overview. W3C October (2009)

[9] Prud'hommeaux, E., Seaborne, A.:SPARQL query language for RDF. W3C (2008)

[10] , Lima, S.: Um modelo ontológico para o contexto turístico. In Conferência IADIS Ibero-Americana WWW/Internet 2007, pages = 99-106, October 2007

[11] Bachlechner, D., Hepp, M., Siorpaes, K.:Towards the Semantic Web in e-Tourism : Lack of Semantics or Lack of Content?. Demos and Posters of the 3rd European Semantic Web Conference (ESWC2006)

[12] Silva, J., Urbano, P., Balsa, J.: What'sUp: A mobile application for searching ongoing cultural events. In : Lopes, A., Pereira, J. (eds.) Proceedings of INFORUM 2012, pages 420-431, September (2012)

[13] Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American, volume 284, pages 34-43, (2001)

[14] Dean Allemang, D., Hendler, J.: Semantic Web for the Working Ontologist Second Edition. Morgan Kaufmann (2011)

[15] Lima, S.: The Semantic Web in Tourism. Chapter 33, pages 675-677,IGI Global (2009)

[16] Cardoso, J.: Semantic Web Processes and Their Applications. Chapter 4, University of Madeira, 9000-390, Funchal, Portugal, Springer (2006)

[17] Dimitris, K.:The Advent of Semantic Web in tourism information systems. In Tourismos: an international multidisciplinary journal of tourism, October (2006)

[18] Maniraj, V., Sivakumar, R.: Ontology Languages - A Review. International Journal of Computer Theory and Engineering, volume 2, number 6,pages 887 - 891, December (2010)

[19] Beckett, D., McBride, B.: RDF/XML Syntax Specification (Revised), on http://www.w3.org/TR/REC-rdf-syntax/. Last Visited on August 2012, W3C (2004)

[20] Chomsky, N.: Three models for the description of language. pages 113-124, (1956)

[21] Cao, T.-D., Phan, T.-H., Nguyen, A.-D.:An ontology based approach to data representation and information search in Smart Tourist Guide System. In 2011 Third international Conference on Knowledge and Systems Engineering, pages 171-175

[22] Fernández, M., Gómez-Pérez, A., Juristo, N.: Methontology: From Ontological Art to Ontological Engineering. In Workshop on Ontological Engineering, pages 33-40, Stanford, CA (1997)

[23] Noël, L., Carloni, O., Moreau, N. et Weiser, S.: Designing a knowledge-based tourism information system. In Int. J. of Digital Culture and Electronic Tourisme, Special Issue on National Tourism Organisations and Exploitation of Information Technologies, Inderscience Publishers Ltd.(2008)

[24] Sowa, J.F.: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley Systems Programming Series, Boston, MA, 1984

[25] Weiser, S., Laublet, P. et Minel,J.-L.: Automatic identification of temporal information in tourism web pages. In E. L. R. A. (ELRA) (Ed.), Proceedings of the Sixth International Language Resources and Evaluation (LREC'08), Marrakech, Morocco

[26] G.Candreva, G.Franco, D.Santo et al.:IDUM a Logic-Based System for e-Tourism, (2008)

[27] Calimeri, F., Galizia, S., Ruffolo, M., Rullo, P.:OntoDLP: a Logic Formalism for Knowledge Representation. pages 1-14, (2003)

[28] Ruffolo, M., Manna, M.: HiLeX: A System for Semantic Information Extraction fromWeb Documents. In: ICEIS (Selected Papers). Volume 3 of LNBIP (2008) 194–209

[29] Siorpaes, K., Bachlechner, D.: Class Hierarchy for the e-Tourism Ontology. e-tourism working group (2004)

[30] Gerasimos, T., Dimitrios, K., Theodore, S.:Querying Ontologies : Retrieving Knowledge from Semantic Web Documents. (2009)

[31] Prantner, K., Ding, Y., Luger, M., Yan, Z.: Tourism Ontology and semantic management system : state-of-the-arts analysis. In IADIS International Conference WWW/Internet 2007, pages 111-115, November (2007)

[32] Legg, C.: Ontologies on the Semantic Web. In Annual Review of Information Science and Technology, Volume 41, Chapter 9, pages 407-451 (2007)

[33] W3C Team, Becket, D., Berners-Lee, T.: Turtle - Terse RDF Triple Language. http://www.w3.org/TeamSubmission/turtle/, Last Visited on July 2012, March (2011)

Acronyms

CFG	Context-Free Grammar.
CPU	Central Processor Unit.
CSV	Comma Separated Value.
DCG	Definite Clause Grammar.
FCT	Fundação para a Ciência e Tecnologia.
GPS	Global Positioning System.
GUI	Graphical User Interface.
HTML	Hyper Text Markup Language.
HTTP	HyperText Transfer Protocol.
MMU	Memory Management Unit.
NER	Named Entity Recognition.
NLP	Natural Language Processing.
OCR	Optical Character Recognition.
OWL	Web Ontology Language.
PDF	Portable Document Format.
POI	Point of Interest.
RDF	Resource Description Framework.
SPARQL	SPARQL Protocol and RDF Query Language.

STAAR	Semantic Tourist informAtion Access and Recommending.
UML	Unifying Modelling Language.
URI	Unified Resource Identifier.
URL	Unified Resource Locator.
W3C	World Wide Web Consortium.
WWW	World Wide Web.
XML	eXtended Markup Language.