Agent- and semantic-based approaches to develop infrastructures for intelligent scenarios

Elton Domnori

Dipartimento di Ingegneria dell'Informazione

Universita degli Studi di Modena e Reggio Emilia

A thesis submitted for the degree of

PhilosophiæDoctor in Information and Communication Technologies

Feb 2012

1. Tutor: Prof. Letizia Leonardi

2. Co-Tutor: Dott. Ing. Francesco Guerra

Day of the defense: February 27, 2012

Signature from head of PhD committee:

Abstract

The aim of the research activity presented in this thesis is the introduction of solutions to support users in intelligent search and coordination. To reach this goal, two different approaches have been taken into account: the semantic-based approach for intelligent search and the agent-based approach for coordination. In particular, on the one hand, to support users in intelligent search we focused on the keyword-based search engines over relational databases. The novelty of our proposal is the exploitation of the semantics expressed in the keywords provided by the user for querying sources without having any direct access to the database instance. In particular, techniques based on regular expressions and semantics associated to the keywords have been implemented and evaluated and the well known Hungarian algorithm has been extended and applied for ranking the results. On the other hand, to support users in coordination the agent-based technology can be a good solution in particular in presence of very dynamic and unpredictable situations such as a disaster scenario. The novelty of our proposal is in this case to offer a fully-distributed coordination among operative units that provides the First Aid service. In fact, coordination is provided by agents that are in charge to collect the necessary information, elaborate it and take the proper decision. This support reduces human errors and helps people in the coordination task.

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Introduction

For many years, computer technology development has aimed to increase the computer performance such as processor speed and disk capability. Smaller computers are able to perform a large number of instructions and elaborate a large amount of data. Recently, computers are gaining an important characteristic that changes their previous nature, that of being *smart*. Researches in the "Artificial Intelligence" area have given an enormous contribution in this direction. Information technology focuses on applications that support people on the everyday life. It tries to move away from the brute force approach to a more intelligent one. Under this point of view, two projects have been carried out, with the aim of introducing solutions to support users in intelligent search and coordination. In particular, two different approaches have been taken into account: the semantic based and the agent-based ones.

The first project is about the development of techniques for the keyword-based search engines over relational databases. Keyword queries over relational databases offer a convenient alternative to traditional SQL queries, especially in case of large, complex or unknown schemas. The challenge in answering a keyword query over a relational database is to find in what database structures the items described by the keywords are modelled, and how these structures relate to each other. Current approaches typically apply some information retrieval technique to collections of data previously gathered and indexed. An Information Retrieval approach is not directly applicable in many interesting application scenarios, including information integration systems and sources from the *Deep Web*, in which the instance data is not directly available for building the required indexes.

For this reason, our research activity was carried on to develop a new approach exploiting only meta-data information of the data sources and using a set of auxiliary information to convert keyword queries into semantically meaningful SQL expressions. The meta-information are extracted from the DBMS and enriched by a human operator or extracted from ontologies such as WordNet. The novelty of this approach lays on the way each keyword is mapped to a schema element (such as a table or an attribute) or to a value (attribute domain). The core of this technique is an extended version of the Hungarian algorithm which significantly reduces the number of possible keyword interpretations that need to be considered. For each interpretation, the set of possible path connections is discovered using graph techniques. For each path, the corresponding SQL query is built, executed and the data retrieved are sent to the user. To perform tests on this new approach, a prototype, Keymantic, has been implemented. Finally, a set of experimental evaluations illustrates its efficiency and effectiveness.

The other project faced during our research activity is the coordination and the cooperation in territorial management scenarios using the multi-agent technology. The management of territorial emergencies and large-scale disasters are one of the most critical activities in healthcare systems. In such scenarios, a very prompt response is necessary in order to reduce the serious damage of the health of the people involved in disasters and to safe as many lives as possible. The criticality of these situations needs coordination, cooperation and decision making. Many are the actors in these scenarios such as medical staff, firemen and policemen. All these actors must cooperate and coordinate in order to reach their target. Decision making must be prompt and accurate. To face such requirements, the multi-agent technology has been employed and in particular the JADE platform has been exploited. The designed architecture, called Ubimedic2 (an extension of Ubimedic), provides a P2P system managed by agents that autonomously exchange the necessary information and organize the rescue operation. This approach is novel and has not been previously proposed in disaster management scenarios. Agents share information using FIPA compliant messages and organize the operations based on the BDI

(Belief-Desire-Intention) model. Each component of Ubimedic2 has been implemented with the main functionality it must provide. Based on the architectural design, the framework has been developed in Java and a GUI prototype is used to simulate real case studies. Moreover, two applications for smart-phones have been developed to perform tests on real devices and evaluate the feasibility of the approach. Finally, the integration with the PIM technology has been done in order to improve the coordination activity among agents, reducing the number of messages exchanged and exploiting the strong mobility to embed the coordination logic in a single process.

The thesis is organized in two main chapters, each one representing one of the projects: the former the Keymantic project and the latter the Ubimedic2 project. In the first chapter, in section 1.1 an analysis of the related work about search engines over relational databases existing in the literature is presented. In Section 1.2 a motivating example is shown to motivate the necessity for a new approach and the problem statement is introduced in Section 1.3. The details of the overall architecture shown in Section 1.4 are discussed from Section 1.5 to Section 1.8. Evaluations are discussed in Section 1.9 and some prototype details introduced in Section 1.10. The second chapter begins with a brief introduction of the topic followed by Section 2.1 that contains an analysis of the related work about multi-agent systems used to support disaster management. A motivating example and the necessity for a new approach are introduced in Section 2.2. In Section 2.3 the new proposed architecture is briefly introduced followed by the exploited technologies in Section 2.4. Some implementation details are introduced is Section 2.5 and in particular an Android application is discussed in Section 2.6. The details of the integration with the PIM technology are introduced in Section 2.7. Finally conclusion ends this thesis.

LIST OF TABLES

1

Keymantic

Recent advances in information and communication technologies and the large use of computers and hand-handle devices (such as smart-phones and tablets) able to access the web have brought companies and individuals to bring data and services online. The World Wide Web has become a new way to market and to share information as people from all over the world, through the web, can access to these data and services increasing the number of potential users. Besides companies that make use of the web not only for advertisement but for business too, individuals are becoming more and more an important element of the web content. Free services such as forums, blogs, websites and social networks have brought people to publish more and more content. This has led to an exponential growth of the amount of data that is available on the web.

Data are published on the web in different forms and under different structures such a web pages and documents, images, audio and video and databases. These kind of data can be divided into two groups: *flat* and *structured* data. The first are accessed by an URL (Universal Resource Location) which is unique in the web. The second are data not directly accessible on the web but they are produced and made available from tools such as web services or web forms.

The huge amount of data available on the web makes difficult the exploration of content of particular interest. In this issue, search engines (such as the well-known Google, Yahoo! and Bingo) come into help. These engines wrap the data available on different web sites, collecting and indexing the content of web pages for fast access during research. These search engines collect only flat data, accessible through URLs, contained in document format such as HyperText Markup Language (HTML) pages, Office Document (DOC/ODP) and Portable Document Format (PDF) file. Besides the enormous number of such documents available on the web, only a small part of them are explored by search engines which can crawl only public static documents.

It has been estimated that the dynamic web content, known as “deep web” or “hidden web”, is currently 4000 to 5000 [1] times larger than the static web. These data come from documents or web pages available under authentication, web pages created dynamically, data relying on databases, multimedia files etc. Another estimation evaluates that 10% of the deep web is accessible without any authentication.

Besides non authorized or non textual data (such as images, audio and video), structured data relying on databases consist of a relevant unexplored amount of web data.

Usually, such data are accessible through web forms, where user provide some structured search criteria and web pages are created on the fly containing the information requested.

Structured data are not covered by the traditional search engines as they are accessed using traditional query techniques based on highly structured and strongly-typed query languages such as SQL. The different structure the data are stored and the different query languages used to access are being day-by-day proven limited for the modern highly heterogeneous web environment, since they require from users a complete knowledge of the underlying data structures and semantics. Unfortunately, the latter may be too large and too complex to be communicated to the user. They know a number of characteristics of the element of interest but they may not know how these characteristics are related to each other, thus, they prefer an interaction with the system to be more of exploratory nature. All these reasons have made users even more uncomfortable with the highly structured query languages.

Keyword-based searching has been introduced as a viable alternative. A keyword query is a list of keywords related in some way to each other and describing the element of interest. The list is typically flat, i.e., no relationship is provided among the keywords, no dependencies, no semantics, and no information on the keyword roles with respect to the repository structures. Due to this simplicity, keyword-based searching is becoming increasingly popular.

Keyword-based searching has been extensively studied and used in the area of Information Retrieval. It typically works by building a set of specialized indexes over sets of documents and then using these indexes to identify the documents that contain as many as possible of the keywords provided in a query. Later, the document or part of the document containing the keywords are shown to the user as research result. The documents are ranked by their relevance using different algorithms and techniques in order to show to the user the most relevant document at the top of the result list. Algorithms have different accuracy and precision and are still a case of study.

The information retrieval techniques combined with the simplicity of the use of keywords has attracted different database research groups. The more the relational data complexity is increasing and the user base is shifting towards the less technically skilled, the more the keyword searching is becoming an attractive alternative to traditional SQL queries, mainly due to its simplicity. Unfortunately, this simplicity

comes with the price of two main issues: ambiguity and lack of semantic in structured data. Thus, the challenge of answering a keyword query over a relational database is to discover the database structures that contain the keywords and explore how these structures are inter-connected to form an answer. The discovered structures, alongside their interconnections, are actually representing in relational terms the semantic interpretation of the keyword query.

Moreover, existing techniques suffer from three main limitations. The first is that they require a-priori access to the data instance in order to build the indices that will locate the tuples related to the given keywords at run time. This seriously limits their applicability if such access is not possible.

The second limitation is that no considerable attention has been paid to the inter-dependencies among the query keywords. The likelihood that a specific data structure represent the same semantics as a keyword in a user query does not only depend on the relationship between the keyword and the data structure, but also on the data to which the other keywords in the query are mapped. This is because despite the fact that a keyword query is a flat list of keywords, the meaning of each keyword is not independent of the meaning of the others, but they all collectively represent the intended concepts the user had in mind posing the query.

The third limitation is that all the keywords are considered as instance values.

In this thesis, a novel technique for answering keyword queries over relational databases has been proposed. The queries are translated into a set of SQL queries that capture the possible semantics of the keyword query. The generated SQL queries can be evaluated on the database, and their results serve as the answer to the keyword query. One of the novelties of the technique is that it is not based on an a-priori access to the database instances. Moreover, this approach exploits the relative positions of the keywords in the query alongside auxiliary external knowledge in order to make a more educated guess of the semantics that most likely represent those of the keyword query, and then rank them accordingly. The strategy can not only be easily incorporated to many relational database management systems, but it can also be used as an enhancement of existing keyword searching techniques that utilize the database instance, offering them significant improvements over effectiveness and efficiency.

An advantage of this approach is that it can be used to assist users browsing databases with large unknown schemas. It is often the case that users formulate key-

word queries without some specific semantics in mind but in an exploratory manner, mainly when they are neither fully aware of the type of information that is stored in a database, nor of the way this information is stored. The possible interpretations of a keyword query according to the schema and domain information of the database will be generated by our approach. In contrast to other keyword searching techniques on relational data that return sets of linked tuples, the new search engine returns the interpretations expressed in SQL. The study of these queries can reveal tables, attributes and join paths, providing the user with enough information to understand the kind of data that is stored in the database and the way this data is structured.

The key contributions are as follows:

(i) The problem of keyword querying over relational databases that lack a-priori access to the database instance has been formally defined.

(ii) The notion of a weight as a measure of the likelihood that the semantics of a keyword are represented by a database structure, i.e., a table, an attribute, or a value has been introduced. Moreover, the weights to intrinsic and contextual, has been distinguished, to emphasize that this likelihood does not depend only on the meaning of the keyword semantics when the keyword is considered in isolation (intrinsic), but also on the way the semantics of the remaining, especially the neighbouring, keywords are represented in the data (contextual).

(iii) The Hungarian (a.k.a., Munkres) algorithm has been extended and exploited [2] to develop a technique for the systematic computation of the contextual weights that leads into to the generation and ranking of the different interpretations of a keyword query in terms of SQL.

(iv) This new approach has been experimentally evaluated on real application scenarios.

1.1 Related work

The popularity of the web search engines and the efficiency they manage large amount of data has attract the attention of many researchers in the field of structured (such as databases) and semi-structured (such as XML) data. This success is due to the Information Retrieval (IR) model in use for more than a decade for text databases [3] and the web [4] and used by search engine such as Google, Yahoo! and Bingo.

[No structured data] The idea behind the IR model is the indexing of the whole content of the documents, building a dictionary with all the words and an association between the single word (element of the dictionary) and the documents it appears. To increase efficiency, some techniques exclude stop words which are not relevant in the research process. The search engine takes in input a set of keyword, provided by the user. The keyword are expected to be not only part of the document content but also expressing particularities of it. The better the keywords will describe the subject of our research the better the result will fulfil our expectation. The use of general words will bring to a large number of document retrieved and to less relevant ones related to our interest. The search engine retrieve all the document containing, partially or fully, the set of keywords and provide these documents to the user. The document are ranked before they are shown to the user. The rank is supposed to represent the document relevance. (for e.g., the simplest algorithm consider more relevant the document where all the set of keywords appear and less relevant documents where only part of the keyword appear on it). For the web, the ranking algorithms are mush more complex and still involves many research activity. The interaction with user changes from engine to engine. Some provide only the links to access the documents and other show part of the document content where the keyword appear.

[Semi-structured data] The model described previously is the dominant model used in search engine for text documents. It has been adopted by the data management community in the context of XML [5, 6, 7]. To answer a keyword query over XML data, the appearances of the query keywords are first identified within the documents (possibly through some free-text search) and then combined together into Meaningful Lowest Common Ancestor Structures (MLCAS). A score is computed based on this structure and according to this score the respective XML documents containing the MLCAS are ranked and returned to the user. XML benefits from the fact that its

basic information model is the "document", which is the same as in IR, thus, many IR techniques can be easily employed in the XML context.

[Structured data] On the other hand, keyword search in relational databases is particularly challenging [8], first because the database instances are way larger, and second because the basic model is fundamentally different making hard the identification of the information units that are to be returned. Nevertheless, keyword search over relational data is particularly appealing, and there are already many interesting proposals in the scientific literature [9, 10]. DISCOVER [11] and DBXplorer [12] have been among the first such systems. The typical approach is to build a special index on the contents of the database and then to use that index to identify the appearances of the query keywords in the attribute values. Many approaches use inverted indexes [11, 13, 14] for that purpose, while others, like DBXplore, use symbol tables. A symbol table is an index consisting of a set of triples $\langle$*value, attribute, relation*$\rangle$, used to map every value to its schema information. Once the keywords are located, the different ways by which their respective tuples are connected are discovered, forming the so-called "joining network" of tuples or tuple trees, that often become the information unit returned to the user. The joining networks are constructed either directly from the instances, or by building query expressions and evaluating them [11]. DISCOVER is interested in finding total and minimal joining networks. A joining network is total if each keyword query is contained in at least one tuple of the network, and minimal if the removal of any tuple makes the network no longer total. More recent approaches [15], are oriented towards reducing the number of tuples that need to be considered in order to improve previous techniques. BANKS [13] follows a similar approach but employs the Steiner tree algorithm to discover how the tuples are associated. SQAK [8] is another system that is based on the same generic principles but focuses on the discovery of aggregate SQL expressions that describe the intended keyword query semantics. Since the set of possible answers may be large, and since the results are already ranked, the above systems typically return the top-k results.

In the Appendix A, a detailed analysis has been done over the different search engines developed so far for a better understanding of their approach, their strong and weak points.

1.2 Motivating Example

In this section, an example will be introduced to better show the main issues of the existing approaches and highlight the improvement that my contributions introduced. Consider the database illustrated in Figure 1.1, containing information about academic researchers (table Person), the departments (table Department) they are affiliated to and their publications (table Publication). The tables Affiliated and Author materialize the many-to-many relationships between Person-Department and Person-Publication, respectively. Let "`Date Database`" be a keyword query posed over this database. To answer this query we need to understand the meaning of each keyword and build an SQL query that offers an interpretation of the keyword query semantics in terms of the relational database.

Person

Name	Area	Phone	Address	Email
Watson	Database	(320) 4631234	30 Bloor	watson@aaa.bb
Lenzerini	Database	(390) 6987654	Ariosto 25	lenzerini@bbb.cc
Date	Database	(817) 1937842	107 GACB	date@ccc.dd
Hunt	Inf. Systems	(343) 2920812	17 Helix	Hunt@ddd.ee

Affiliated

Professor	Department
Watson	x123
Lenzerini	cs34
Date	cs34
Hunt	m111

Department

id	DName	Address	Director
x123	CS	25 Blicker	Watson
cs34	IE	15 Tribeca	Hunt
ee67	EE	5 Charles	Date
m111	ME	2 Cottle	Hunt

Author

Name	Publication
Lenzerini	Data Integration
Date	Foundation Matters

Publication

Title	Year	Resource
Data Integration	2002	ACM DL
Foundation Matters	2002	DBLP

Database

Name	Address
DBLP	http://www.informatik.uni-trier.de
ACM DL	http://portal.acm.org/dl.cfm

Figure 1.1: A fraction of a database schema with its data.

The first issue is to decide *what* is the relation among the keywords and the schema elements. A keyword can represents a (i) value or (ii) describes some meta-information (element of the Entity Relation schema). In the above query, the keyword `Date` may represent the value of a name, and the keyword `Database` the value of a research area. In that case, a possible interpretation of the keyword query is information about a person called Date that has done work in the Database area. If the intended meaning of the keyword query was instead to find the online databases containing Date's publications, then the keyword `Database` is actually representing meta-information about the element of interest.

The second issue is to find is to decide *which* part of the database actually models the intended keyword meaning. Consider the keyword query "`Director Watson Address`" and assume that the first and last keyword represent meta information while the second represents a value. The intended information of the keyword `Director` is most probably the one modeled by the attribute with the respective name. It is not clear, however, whether the keyword `Address` refers to the homonym attribute in the table Person or to the one in the table Department. Choosing one over another leads to completely different interpretations of the keyword query. Furthermore, the keyword `Watson`, even though we know it is a value, might be the name of a director, the name of a street, or even a value of some other attribute.

The third issue is to decide *how* these structures relate to each other. In relational databases, such relationships are expressed either through the table-attribute-value hierarchy or through join paths, i.e., chains of key/foreign key relationships. Between two database structures there may be multiple different join paths. Each such path leads to a different interpretation of the query keywords. For instance, consider the keyword query "`email CS`", and assume that the keyword `email` corresponds to attribute Email and keyword `CS` to a value of the attribute DName. Between DName and Email, there are two different paths, one that goes through the table Affiliated and one through the Director. If the semantics of the keyword query were to find emails of the CS affiliated persons, then the database query describing this semantics is one that uses the first path. If instead the semantics were to find the email of the CS department director, the query is one that uses the second path.

Finding the different semantic interpretations of a keyword query is a combinatorial problem which can be solved by an exhaustive enumeration of the different ways

that mappings can be associated to database structures and values. The challenging task is to develop a mechanism that is able to significantly reduce the possible associations that most likely represent the intended keyword semantics. One way to do this is to exploit syntactic knowledge or other forms of auxiliary information. For instance, a keyword ``(320)463-1463'' is more likely to represent a phone number due to its format, while a keyword Everest, given the publicly available Geo-name knowledge, is very likely to represent the mount Everest. Similarly, the keyword "article" is likely to correspond to the table Publication, based on some lexical database knowledge like WordNet[1].

The quest for the database structure that a keyword corresponds should not be an isolated task. Keywords are not independent entities, but it should be seen in the *context* of the presence of the other keywords in the query. To illustrate this fact, consider the keyword query "Name Date Database". A possible interpretation of the query is that the user is looking for the person called Date who works in the area of databases. This means that keywords Name and Date may correspond to the attribute Name of the table Person and one of its values, respectively, while the keyword Database to one of the values of the attribute Area. Consider now the query "Name Date Database DBLP". In this case, a possible interpretation is that the user is looking for data items related to Date in the DBLP database, hence, the meaning of the keyword Database is represented by the table Database and not by some value of the Area attribute.

A natural and challenging question that comes to mind is which of the different semantic interpretations of a query should be considered as the correct one. It is a well-known and documented fact that keyword queries are under-specified queries [16]. As such, any of the possible interpretations may be correct, and all have to be considered. However, based on some known patterns of human behaviour [16], we know that the order of keywords is important and that correlated keywords are typically close. This means that certain interpretations of a keyword query are actually more likely than others. Thus, instead of returning a flat list of possible interpretations to the user, one can return a ranked list based on the likelihood that they represent the intended keyword semantics.

[1] wordnet.princeton.edu

1.3 Problem statement

In the following section will be introduced a set of definitions and examples necessary for the further understanding of the main problem and all its elements.

Definition 1.3.1 *A database D is a collection of relational tables $R_1, R_2, \ldots, R_n$. Each relational table R_i is denoted as $R_i(A_1^i, A_2^i, \ldots, A_n^i)$, where R is the name of the table and $A_1, A_2, \ldots, A_n$ its attributes. Each attribute A has a domaim denoted as $dom(A)$.*

Let V_t represent the collection of relation R, $V_a = \{A \mid A \in R \wedge R_{\in} V_t\}$ represents the set of all the attributes of all the tables in the database and $V_d = \{d \mid d = dom(A) \wedge A \in V_a\}$ represents the set of all their respective domains.

Definition 1.3.2 *The* database vocabulary *of D, denoted as V_D, is the set $V_D = V_t \cup V_a \cup V_d$. Each element of the set V_D is referred to as a* database term

In other words, the vocabulary of a database is the set of all its relation names, their attributes and their respective domains. A *database term* is a member of its vocabulary. Two subsets of the database vocabulary are distinguished: the *schema vocabulary* $V_{SC} = V_t \cup V_a$ and the *domain vocabulary* $V_{DO} = V_d$ that concerns the instance information.

A keyword query KQ is an ordered l-tuple of keywords $(k_1, k_2, \ldots, k_l)$. Each keyword is a specification about the element of interest. The specification may have been modeled in the database as a relational table, an attribute, or a value of an attribute. A configuration is a mapping function that describes a specification for each query keyword in terms of database terms.

In this approach, it has been assumed that some of the keywords may express data (like in the previous search engines approach) and other description of the data. As introduces in the motivation example, the user used to give both information in the keyword list. The keywords that describe data tell us what the user is looking for, e.g.: hotels, cars, etc.

The same does the ER schema giving a description of the data. An example of relational table is Hotel(name, address, city, star, services, ...). This tell us that the data are about *hotels* and the attributes of the table inform us about the characteristics of the data such as name of the hotel, its address etc. Combining these two information is possible to associate the keyword given by the user the schema elements.

Definition 1.3.3 *A* configuration $f_c(KQ)$ *of a keyword query* KQ *on a database* D *is an injective function from the keywords in* KQ *to database terms in* V_D*. In other words, a configuration is a mapping that describes each keyword in the original query in terms of database terms.*

The reason a configuration is considered to be an injective function is because it has been assumed that: *(i)* each keyword cannot have more than one meaning in the same configuration, i.e., it is mapped into only one database term; *(ii)* two keywords cannot be mapped to the same database term in a configuration since overspecified queries are only a small fraction of the queries that are typically met in practice [16]; and *(iii)* every keyword is relevant to the database content, i.e., keywords always have a correspondent database term. Furthermore, while modelling the keyword-to-database term mappings, it also has been assumed that every keyword denotes an element of interest to the user, i.e., there are no stop-words or unjustified keywords in a query. This work does not address query cleaning issues. Moreover, it has been assumed that the keyword queries have already been pre-processed using well-known cleansing techniques.

Definition 1.3.4 *The schema graph* $G(N, E)$ *is a directed graph that captures the primary key to foreign key relationships in the database schema. It has a node* $n_i \in N$ *for each relation* R_i *of the database and an edge* $e_i \in E$ *for each primary key to foreign key relationship,* $R_i \rightarrow R_j$*, form a set of attributes. The graph* G_j *is defined to be the undirected version of* G*.*

Answering a keyword query over a database D means finding a set of SQL queries, with each of these queries using all the database elements that belong to the range[1] of a specific configuration. Such an SQL query is referred to as an *interpretation* of the keyword query, since it provides a possible meaning of the keyword query in terms of the database vocabulary. In the current work, it has been considered only select-project-join (SPJ) interpretations, that are typically the queries of interest in similar works [11, 12]. Nevertheless, interpretations involving aggregations [8] are also in our list of future extensions.

[1] Since a configuration is a function, we can talk about its range.

Definition 1.3.5 *An* interpretation *of a keyword query $KQ = (k_1, k_2, \ldots, k_l)$ on a database D using a configuration $f_c^*(KQ)$ is an SQL query in the form*
select $A_1, A_2, \ldots, A_o$ *from* R_1 JOIN R_2 JOIN ... JOIN R_p *where* $A'_1{=}v_1$ AND $A'_2 = v_2$ AND ... AND $A'_q = v_q$
such that the following holds:

- $\forall A \in \{A_1, A_2, \ldots, A_o\}$*:* $\exists k \in KQ$ *such that* $f_c^*(k) = A$
- $\forall R \in \{R_1, R_2, \ldots, R_p\}$*: (i)* $\exists k \in KQ$*:* $f_c^*(k) = R$ *or (ii)* $\exists k_i, k_j \in KQ$*:* $f_c^*(k_i) = R_i \wedge f_c^*(k_j) = R_j \wedge$ *exists a join path from* R_i *to* R_j *that involves* R
- $\forall$ "$A' = v$" $\in \{A'_1 = v_1, A'_2 = v_2, \ldots, A'_o = v_o\}$*:* $\exists k \in KQ$ *such that* $f_c^*(k) = dom(A') \wedge k = v$
- $\forall k \in KQ$*:* $f_c^*(k) \in \{A_1, A_2, \ldots, A_o, R_1, R_2, \ldots, R_p, dom(A'_1), \ldots, dom(A'_q)\}$

The existence of a database term in an interpretation is justified either by belonging to the image of the respective configuration, or by participating in a join path connecting two database terms that belong to the image of the configuration. Note that even with this restriction, due to the multiple join paths in a database D, it is still possible to have multiple interpretations of a keyword query KQ given a certain configuration $f_c^*(KQ)$. The notation $\mathcal{I}(KQ, f_c^*(KQ), D)$ has been used to refer to the set of these interpretations, and $\mathcal{I}(KQ, D)$ for the union of all these sets for a query KQ.

Example 1.3.1 *Consider the keyword query "email CS" over the database of Figure 1.1,and a configuration that maps* `email` *to* Email *and* `CS` *to* DName*. At least two different interpretations can be generated from this. One is the:*

select Email
from Person P JOIN Affiliated A ON (P.name=A.professor)
JOIN Department D ON (A.Department=D.id)
where D.DName='CS' AND *D.id=A.Department*
AND *A.Professor=P.Name*

and the other is the:

select Email
from Person P JOIN Department D ON (P.person=D.Director)
where D.DName='CS' AND *D.Director=P.Name*

Definition 1.3.6 *An* answer *to a keyword query q over a relational database D is the set $ans(q) = \{t \mid t \in eval^D(q') \wedge q' \in \mathcal{I}(q, D)\}$, where $eval^D(q')$ denotes the evaluation of the relational query q' on the database D.*

Since there is no restriction on the number of attributes in the select clause of an answer, the answer set of a keyword query will naturally be heterogeneous.

Since each keyword in a query can be mapped to a relation name, an attribute name or an attribute domain, there are $2 * \Sigma_{i=1}^{|D|}|R_i|$ + $|D|$ different configurations, with $|R_i|$ denoting the *arity* of the relation R_i and $|D|$ the number of tables in the database. Based on this, and on the fact that no two keywords can be mapped to the same database term, for N keywords, there are $\frac{|V_D|!}{(|V_D|-N)!}$ possible configurations. Of course, not all the interpretations generated by these configurations are equally *meaningful*. Some are more likely to represent the intended keyword query semantics. In the following sections is show how different kinds of meta-information and inter-dependencies between the mappings of keywords to database terms can be exploited in order to effectively and efficiently identify these interpretations and present them first. Our work is based on the natural assumption that the intended semantics of the keyword query can be expressed as a query over the relational database. If those semantics cannot be expressed, no answer can be provided.

1.4 Architecture

The main idea that stays behind the search engine approaches analyzed so far is the mapping of a set of keywords inserted by the user with the relational table-attribute of the database. This is achieved with the exploitation of indexed data which enables a fast search of where (under which table and attribute) a keyword appears in the database. Once the relational tables and attributes and detected, the minimum joining tree is calculated and data retrieved. The Keymantic approach is slightly different due to the assumption that there is no knowledge about the data of the database but what is known is an extraction of meta-data (semantic information about the data) of the Entity Relational schema. A set of techniques produce the possible mappings between each keyword and the relational table-attribute from which the join trees are calculated and data retrieved. In Figure 1.2 the architecture schema of Keymantic is shown.

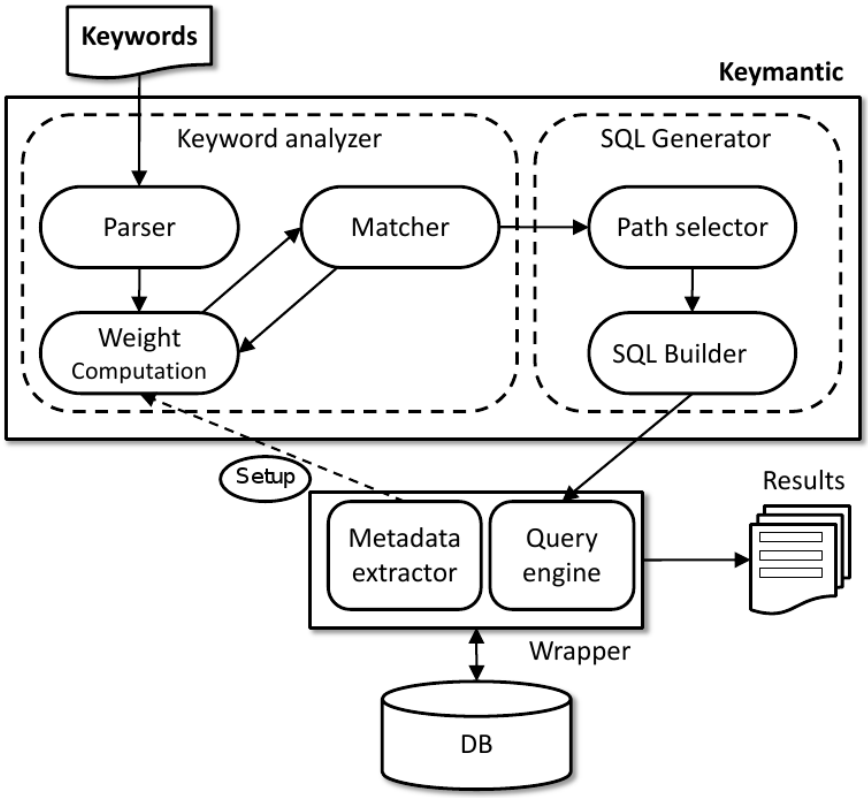

Figure 1.2: Keymantic Architecture

The two main modules are the “Keyword Analyzer” and “SQL Generator”. The

first takes in input the keyword query string and produces the possible configuration (mapping) between the keywords in KQ and the database term V_D. The second takes in input these configuration and generates possible joining tree and generate the respective SQL query string.

[MetaData extractor] This module operates offline and consists in a setup phase where the information about the ER schema are extracted from the database. The information retrieved, such as table names and their attributes, are used to create the database vocabulary V_D introduced in the previous section. This vocabulary is enriched by the human database manager or with the use of automated tools such as WordNet [1] and DMoz [2] to retrieve possible synonyms, hyponyms and hypernyms. Moreover, the structure of the ER schema is retrieved in order to create the graph by which the Minimum Spanning Trees will be calculated. A custom wrapper has been implemented for this purpose. It has been developed in JAVA using the JDBC libraries and in particular the Metadata class which make available a set of public methods to access the necessary information about the database schema.

[Parser] This modules takes in input the keyword query string and parses it detecting the single keywords. The keyword may represent compound names such as "New York" or "Computer Science". The user inputs the query string by inserting the compound name enclosed in brackets.

[Weight computation] assigns a weight to each keyword that represents a schema element and to each keyword that represent value related to schema element in order to build a weighted configuration;

[Matcher] using the weighted tables build previously, it creates the configuration list. The number of configurations retrieved depends on the threshold parameter which defines how far from the best solution the system must go. This value goes from 0 to 1 where o means all the solutions and 1 only those configurations that has a ranking value equal to the best one.

[Path selector] discovers all the possible paths (joining trees) connecting the relations in the configuration. This process is similar to the Candidate Network discovery shown in the other approaches. All the paths are ranked form the shortest one to the

[1] WordNet: wordnet.princeton.edu
[2] DMoz: www.dmoz.org

longest and are shown to the user who will decide (through the user interface) the one to explore.

[SQL builder] After the user decides which path to explore, this module builds the query that will be sent to the database manager to retrieve the data. The query string depend on the database query language (such as SQL and SPARQL). More than one query string can be generated if more databases are accessed. All the the query strings are generated based on the same joining path tree.

[Query engine] This module receives the query strings from the "SQL builder" and execute them to retrieving the requested data. The data are sent to the output interface of the prototype or can be saved in any other format requested such as XML or CSV.

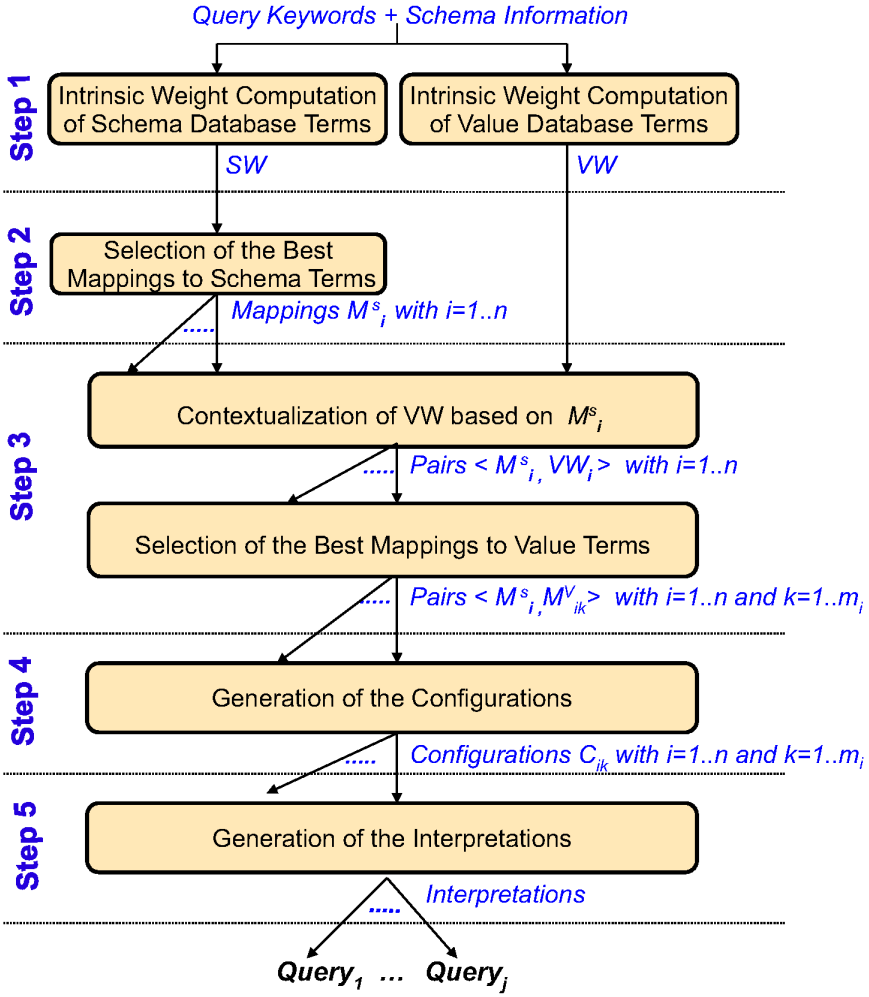

Figure 1.3: Overview of the keyword query translation process

1.5 From Keywords to Queries

The generation of interpretations that most likely describe the intended semantics of a keyword query is based on semantically meaningful configurations, i.e. sets of mappings between each keyword and a database term. The notion of *weight*, introduced in this work, offers a quantitative measure of the relativeness of a keyword to a database term, i.e., the likelihood that the semantics of the database term are the intended semantics of the keyword in the query. The sum of the weights of the keyword-database term pairs can form a score serving as a quantitative measure of the likelihood of the configuration to lead to an interpretation that accurately describes the intended keyword

	R_1	...	R_n	$A_1^{R_1}$	...	$A_{n_1}^{R_1}$	...	$A_{n_n}^{R_n}$	$\underline{A_1^{R_1}}$	...	$\underline{A_{n_1}^{R_1}}$	...	$\underline{A_{n_n}^{R_n}}$
$keyword_1$													
$keyword_2$													
...													
$keyword_k$													

Table 1.1: Weight table with its SW (light) and VW (dark) parts

	P	D	P.Na	P.Ar	P.Ph	P.Ad	P.Em	D.Id	D.Dn	D.Ad	D.Di
$workers$	68	0	0	0	0	0	0	0	0	0	0
$department$	0	100	0	0	0	0	0	0	0	0	0
CS	0	0	0	0	0	0	0	0	0	0	0

Table 1.2: Intrinsic Weight SW matrix.

query semantics. The range and full semantics of the score cannot be fully specified in advance. They depend on the method used to compute the similarity. This is not a problem as long as the same method is used to compute the scores for all the keywords. This is the same approach followed in schema matching [17] where a score is used to measure the likelihood that an element of a schema corresponds to an element of another.

The naive approach for selecting the best configurations, and, as a consequence, generating the most prominent interpretations of a keyword query, is the computation of the score of each possible configuration and then selecting those that have the highest scores. Of course, the target is to avoid an exhaustive enumeration of all the possible configurations, and compute only those that give high scores. The problem of computing the mapping with the maximum score without an exhaustive computation of the scores of all the possible mappings is known in the literature as the problem of *Bipartite Weighted Assignments* [18]. Unfortunately, solutions to this problem suffer from two main limitations. First, apart from the mutual exclusiveness, they do not consider any other interdependencies that may exist between the mappings. Second, they typically provide only the best mapping, instead of a ranked list based on the scores.

To cope with the first limitation, two different kinds of weights has been introduce: the *intrinsic*, and the *contextual* weights. Given a mapping of a keyword to a database term, its intrinsic weight measures the likelihood that the semantics of the

	P.Na	P.Ar	P.Ph	P.Ad	P.Em	D.Id	D.Dn	D.Ad	D.Di
$workers$	1	1	0	1	0	1	1	1	1
$department$	1	1	0	1	0	1	1	1	1
CS	1	1	0	1	0	1	1	1	1

Table 1.3: Intrinsic Weight VW matrix.

keyword is that of the database term if considered in isolation from the mappings of all the other keywords in the query. The computation of an intrinsic weight is based on syntactic, semantic and structural factors such as attribute and relation names, or other auxiliary external sources, such as vocabularies, ontologies, domains, common syntactic patterns, etc. On the other hand, a contextual weight is used to measure the same likelihood but considering the mappings of the remaining query keywords. This is motivated by the fact that the assignment of a keyword to a database term may increase or decrease the likelihood that another keyword corresponds to a certain database term. This is again based on observations that humans tend to write queries in which related keywords are close to each other [16]. As an example, for the keyword query ``Watson Area Database'' expressed on the database in Figure 1.1, since the keyword ``Database'' is right next to keyword Area, mapping the keyword Area to the attribute Area of the table Person makes more likely the fact that the keyword Database is an area value, i.e., should be mapped to the domain of the attribute Area. At the same time, it decreases its relativeness to the table Database.

A similar idea has already been exploited in the context of schema matching [19] with many interesting results. To cope with the second limitation, a novel algorithm for computing the best mappings has been developed. The algorithm is based on and extends the Hungarian (a.k.a., Munkres) algorithm [2] and will be described in detail in Section 1.8.

A visual illustration of the individual steps in the keyword query translation task is depicted in Figure 1.3. A special data structure, called *weight matrix*, plays a central role in these steps. The *weight matrix* is a two-dimensional array with a row for each keyword in the keyword query, and a column for each database term. The value of a cell $[i, j]$ represents the weight associated to the mapping between the keyword i and the

database term j. Table 1.1 provides an abstract illustration of a weight matrix. An R_i and $A_j^{R_i}$ columns correspond to the relation R_i and the attribute A_j of R_i, respectively, while a column with an underline attribute name $\underline{A_j^{R_i}}$ represents the data values in the column A_j of table R_i may have, i.e., its domain. Two parts (i.e., sub-matrices) can be distinguished in the weight matrix. One corresponds to the database terms related to schema elements, i.e., relational tables and attributes, and the other one corresponds to attribute values, i.e., the domains of the attributes.In the following, *schema database terms* refers to database terms related to schema elements and *value database terms* to those related to domains of the attributes. In Table 1.1, these two sub-matrices are illustrated with different shades of gray. *Schema weights* refers to the weights in the first sub-matrix and *value weights* to those of the second. The notation SW and VW to refer either to the respective sub-matrix, or to their values. The details of the individual steps of Figure 1.3 are provided next.

Intrinsic Weight Computation. The first step of the process is the intrinsic weight computation. The output is the populated SW and VW sub-matrices. The computation is achieved by the exploitation and combination of a number of similarity techniques based on structural and lexical knowledge extracted from the data source, and on external knowledge, such as ontologies, vocabularies, domain terminologies, etc. Note that the knowledge extracted from the data source is basically the meta-information that the source makes public, typically, the schema structure and constraints. In the absence of any other external information, a simple string comparison based on tree-edit distance can be used for populating the SW sub-matrix. For the VW sub-matrix the notion of *Semantic Distance* [20] can always be used in the absence of anything else. As it happens in similar situations [17], measuring the success of such a task is not easy since there is no single correct answer. In general, the more meta-information has been used, the better. However, even in the case that the current step is skipped, the process can continue with the weight matrix where all the intrinsic values have the same default value. The computation of the intrinsic weights is detailed in Section 1.6.

Selection of the Best Mappings to Schema Terms. The intrinsic weights provide a first indication of the similarities of the keywords to database terms. To generate the prominent mappings, we need on top of that to take into consideration the inter-dependencies between the mappings of the different keywords. We consider first the

prominent mappings of keywords to schema terms. For that we work on the SW sub-matrix. Based on the intrinsic weights, a series of mappings M_1^S,M_2^S, ..., M_n^S, of keywords to schema terms are generated. The mappings are those that achieve the highest overall score, i.e., the sum of the weights of the individual keyword mappings. The mappings are partial, i.e., not all the keywords are mapped to some schema term. Those that remain unmapped will play the role of an actual data value and will be considered in a subsequent step for mapping to value database terms. The selection of the keywords to remain unmapped is based on the weight matrix and some cut-off threshold. Those with a similarity below the threshold remain unmapped. For each of the mappings M_i^S, the weights of its SW matrix are adjusted to take into consideration the context generated by the mapping of the neighborhing keywords. It is based on the observation that users form queries in which keywords referring to the same or related concepts are adjacent [16, 21]. The generation of the mappings and the adjustment of the weights in SW are performed by our extension of the Hungarian algorithm that is described in detail in Section 1.8. The output of such a step is an updated weight matrix SW_i and, naturally, an updated score for each mapping M_i^S. Given the updated scores, some mappings may be rejected. The selection is based on a threshold. There is no golden value to set the threshold value. It depends on the expectations from the keyword query answering systems. The higher its value, the less the interpretations generated at the end, but with higher confidence. In contrast, the lower the threshold value, the more the mappings with lower confidence.

Contextualization of VW and selection of the Best Mappings to Value Terms. For each partial mapping M_i^S of keyword to schema terms generated in the previous step, the mappings of the remaining unmapped keywords to value terms needs to be decided. This is done in two phases. First, the intrinsic weights of the VW sub-matrix that were generated in Step 1 are updated to reflect the added value provided by the mappings in M_i^S of some of the keywords to schema database terms. This is called the process of contextualization of the VW sub-matrix. It is based on the documented observation that users form queries in which keywords specifying metadata information about a concept are adjacent or at least neighboring [16, 21]. Thus, when a keyword is mapped to a schema term, it becomes more likely that an adjacent keyword should be mapped to a value in the domain of that schema term. The contextualization process increases the weights of the respective values terms to reflect exactly that. For example, in the

keyword query ``Name Alexandria'' assume that the keyword Alexandria was found during the first step to be equally likely the name of a person or of a city. If in Step 2 the keyword Name has been mapped to the attribute Name of the table Person, the confidence that Alexandria is actually the name of a person is increased, thus, the weight between that keyword and the value database term representing the domain of attribute Name should be increased, accordingly.

In the second phase, given an updated VW_i sub-matrix, the most prominent mappings of the remaining unmapped keywords to value database terms are generated. The mappings are generated by using again the adapted technique of the Hungarian algorithm (ref. Section 1.8). The result is a series of partial mappings M_{ik}^V, with $k = 1..m_i$, where i identifies the mapping M_i^S on which the computation of the updated matrix VW_i was based. Given one such mapping M_{ik}^V the value weights are further updated to reflect the mappings of the adjacent keywords to value database terms, in a way similar to the one done in Step 2 for the SW sub-matrix. The outcome modifies the total score of each mapping M_{ik}^V, and based on that score the mappings are ranked.

Generation of the Configurations. As a fourth step, each pair of a mapping M_{ik}^V together with its associated mapping M_i^S is a total mapping of the keywords to database terms, forming a configuration C_{ik}. The score of the configuration is the sum of the scores of the two mappings, or alternatively the sum of the weights in the weight matrix of the elements $[i, j]$ where i is a keyword and j is the database term to which it is mapped through M_{ik}^V or M_i^S.

Generation of the Interpretations. Having computed the best configurations, the interpretations of the keyword query, i.e., the SQL queries, can be generated. The score of each such query is the score of the respective configuration. Recall, however, that a configuration is simply a mapping of the keywords to database terms. The presence of different join paths among these terms results in multiple interpretations. Different strategies can be used to further rank the selections. One popular option is the length of the join path [22] but other heuristics found in the literature [11] can also be used. It is also possible that a same interpretation be obtained with different configurations. A post-processing analysis and the application of data-fusion techniques [23] can be used to deal with this issue. However, this is not the main focus of the current work and we will not elaborate further on it. We adopt a greedy approach that computes a query for every alternative join path. In particular, we construct a graph in which

each node corresponds to a database term. An edge connects two terms if they are structurally related, i.e., through a table-attribute-domain value relationship, or semantically, i.e., through a referential integrity constraint. Given a configuration we mark all terms that are part of the range of the configuration as "marked". Then we run a breath-first traversal (that favors shorter paths) to find paths that connect the disconnected components of the graph (if possible). The final SQL query is then constructed using the "marked" database terms, and in particular, the tables for its from clause, the conditions modeled by the edges for its where clause and the remaining attributes for its select clause. Then the process is repeated to find a different interpretation, that will be based on a different join path. The final order of the generated interpretations is determined by the way the different paths are discovered and the cost of the configuration on which each interpretation is based.

It is important to note here that if the thresholds used in the above steps are all brought down to zero, then our technique is able to generate all the possible interpretations that can be defined on a database, even the most unlikely. In that sense, our technique is complete. The thresholds serve only to exclude from the results any interpretation that is not likely to represent the semantics of the keyword query, while the weights are used to provide the basis for a ranking metric.

1.6 Intrinsic Weight Computation

To compute the intrinsic weights, we need to compute the relevance between every query keyword and every database term. The fundamental information that must be used towards this directions are found is the schema information. The main DBMS-es provide all the information about the schema such as table and attribute names, the domains of the attributes, foreign key to primary key relation etc. Syntactic descriptions of the contents of an attribute (e.g., regular expressions) can also lead to a better matching of keywords to database terms since they offer indications on whether a keyword can serve as a value for an attribute or not. There are already many works that offer typical syntax for common attributes such as phone numbers, addresses, etc. [17], and have been used extensively and successfully in other areas. For specific syntax description, regular expressions are very useful as they are able to clearly detect keywords expressing IP address, ISBN codes, dates etc. This improves the associations of keywords to database terms reducing the number of non significant configuration.

If access to the catalog tables is possible, assertion statements can offer an alternative source of syntactic information. In the same spirit, relevant values [24], i.e., clusters of the attribute domains, are also valuable auxiliary information. This can be a more efficient alternative to the data collection used in the current approaches. The data collection is much more restrained than all the attribute domain as it consists of reduces and most significant data of the domain.

During search people may insert different keyword that express the same concept. In this case, an exact match between a keyword and one ore more database terms is too restrictive because the search result will be influences by the way the database terms have been chosen. The most common case are the *synonyms* (such as street, road and avenue) but other set of words may express approximately the same idea (such as hotel, hostel and motel). Two other relations that come into help are the *hyponyms* and *hypernims*. In linguistics, a *hyponym* is a word or phrase whose semantic field is included within that of another word which is called *hypernim*. For example, if the user is looking for the a “scientific paper” and in the database we found the table called “Publication” (which is hypernim of scientific paper) we must relate the two. This relation is less strong than the synonym one but this will be discussed ahead. Fortunately there is today a large volume of grammatical and semantic information

that is publicly available on the Web and can be used as a service. Examples include the popular WordNet [1] and the many community specific ontologies. This tool enriches the dictionary with three group of words; synonyms, hyponyms and hypernims.

Besides the automatic tools available to enrich the glossary of the database term, a particular attention must be payed to the data extracted for DBMS. Sometimes, table and attribute names are not semantically significant. Database administrators may use code names such as "T01", or acronyms or abbreviations to name tables and attributes. For this reason, automatic wrappers are not always sufficient and a human intervention is necessary.

1.6.1 Weights for Schema Database Terms

Before processing the keyword set, we must take into account two different aspects: keyword may not match exactly to database term (because the user may write the word in plural or may miss any letter while writing text) and keyword may express a concept similar to a database term (such as synonyms). To face the first issue there are two main techniques used in the information retrieval area: stemming and string similarity.

Stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form - generally a written word form. The stem need not be identical to the morphological root of the word. It is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. This technique is useful to avoid all possible derived or inflected words such as in the above example of plural words. The stemming processing has not been widely adopted from the search engines because of the lack of significant results and the necessity of large dictionaries. Roles are not sufficient. In English, many of the word ending with "-ing" are derived words but this is not an universal role. Words such as "boeing" (airplane model) does not derive from "boe" such as "doing" derived from "do". Considering the complexity and the low efficiency of this technique we decided not to include it in our engine.

The **string similarity** [25] is the process which evaluates quantitatively the similarity between to words. There are different metrics developed so far such as the Jaccard, the Hamming and the Levenshtein. This technique is useful to detect errors spelling error and is used for many of the search engines and text editors. While the user writes

[1] http://wordnet.princeton.edu/

the text, if the engine do not recognize any word it provides the user a list of alternative words. This process is computationally fast and do not need any other support (such as dictionaries). It returns a number, within a range, that measures (based on the different metrics) the similarity between two words. In our system we adopted the Levenshtein metric. The range of the metric is from 0 (no similarity at all) to 1 (exact matching).

To deal with the second issue we used three set of words provided by WordNet: synonyms, hypernyms and hyponyms. Hypernyms (hyponyms) between word$_A$ and word$_B$ is divided into two categories: "direct hypernym" (word$_A$ = hypernym(word$_B$)) and "derived hypernym" (word$_A$ = hypernym(word$_C$) and word$_C$ = hypernym(word$_B$)). From these considerations we formulated the following coefficients:

Type	Function	Value
synonym	synonym(A,B)	1
direct hyponym	hyponym(A,B)	0.8
direct hypernym	hypernym(A,B)	0.6
derived hyponym	hyponym2(A,B)	0.4
derived hypernym	hypernym2(A,B)	0.2

Table 1.4: Differences similarity functions

Algorithm 1 describes the computation procedure of the intrinsic schema weight matrix SW. As shown in figure reffig:, the matrix SW represents the relationship between the a keyword of the query Q and a database term. For each couple (w, t) where $w \in Q$ and $t \in T$, the algorithm calculates the similarity of the two. The set Σ represents the sum of the similarity methods we employed. We have a number of default methods that represent the state of the art in the area, but additional methods can be included. Each such method takes as input two strings and returns their respective similarity in a range between 0 and 1. We trust the method that gives the highest similarity. The algorithm does not make the sum of all similarities but takes only the higher one. If the maximum similarity between a keyword and a schema term is found below a specific threshold (that is set by the application) then the similarity is set explicitly to 0. As a result, at the end of the procedure there might be rows in the matrix SW containing only zeros. These rows represent keywords that are not similar

Algorithm 1: Intrinsic SW Matrix Computation

Input: Q: Keyword Query
T: Schema Database Terms
Output: SW matrix

COMPUTEISW(Q,T)
(1) $SW \leftarrow [0, 0, \ldots, 0]$
(2) $\Sigma \leftarrow$ { Synonyms(w,t), Hyponyms(w,t), Hypernyms(w,t), StringSimilarity(w,t) }
(3) **foreach** $w \in Q$
(4) **foreach** $t \in T$
(5) $sim \leftarrow 0$;
(6) **foreach** $m \in \Sigma$
(7) **if** $m(w,t) > sim$
(8) $sim \leftarrow m(w,t)$
(9) **if** $ssim \leq threshold$
(10) $sim \leftarrow 0$;
(11) $SW[w,t] = ssim$ * 100

enough to any of the schema database terms, thus, they will be considered later as candidates for mapping to value database terms, i.e., domains of the schema attributes. The fact that their rows in the SW matrix are 0 instead of some low value, makes the similarities of the keyword to the value database terms that will be computed in a later step to be the dominating factor determining the guess on the role a specific keyword can play.

Example 1.6.1 *Consider the keyword query "workers department CS" posed on the database of Figure 1.1. Table 1.2 illustrates a fraction of the weight matrix containing the intrinsic weights for the database terms derived from the tables* Person *and* Department*. Instead of the full names of tables and attributes, only the first letter of the tables and the first two letters of the attributes are used. The schema weights SW are the light gray colored part of the matrix. Note that the keyword* CS *has not been mapped to any of the schema terms since all the values of its row in* SW *are 0.*

1.6.2 Weights for Value Database Terms

For computing the intrinsic value weights, we mainly exploit domain information, and base our decision on whether a keyword belongs to the domain of an attribute or not. Furthermore, we have adapted the notion of *Semantic Distance* [20] that is based on results retrieved by a search engine in order to evaluate the relatedness of two concepts. In particular, we define the *semantic relatedness* $SR(x, y)$ of two terms x and y as: $SR(x, y) = e^{-2NXD(x,y)}$ where $NXD(x, y) = \{max\{logf(x), logf(y)\} - logf(x, y)\}/\{logN - min\{logf(x), logf(y)\}\}$ with $f(x)$ denoting the number of web documents containing x, and $f(x, y)$ the number of documents containing both x and y, as these numbers are reported by specific search engines such as Google, Yahoo!, Cuil, Excite!, etc. The number N represents the number of documents indexed by the corresponding search engine. For our purpose, we compute the semantic relatedness of every keyword - attribute domain pair and this gives us an indication of the similarity degree between the keyword and the attribute domain.

Information about possible values that an attribute can accept is also an important factor. The information is based on the explicit enumeration of values, as in the *Relevant Values* approach [24]. When a keyword is found among (or is similar to) the valid values that an attribute can get, the keyword receives a high weight. Additional comparison techniques include semantic measures based on external knowledge bases.

Example 1.6.2 *For the keyword query introduced in Example 1.6.1, the intrinsic value weights are indicated in the VW part of Table 1.2 i.e., the dark gray-colored part. These weights have been computed by using domain knowledge and regular expressions. Note that these are the value weights, thus, the similarity is not between the keyword and the name of the attribute, but concerns the compatibility of the keyword with the attribute domain.*

1.7 Contextualization

The process of contextualization, as previously explained, exploits the inter-dependencies across mappings of different keywords. The previous step, aims to identify the keywords, at the query string, that are mapped to a database schema element such as table or attribute name. The remaining keywords, which are not mapped to any database schema element, are supposed to be mapped to the database attribute domain. The whole mapping of the keywords to the database term defines a configuration. The keyword query is an unstructured list of keywords which express the intention of the user. Anyway, it has been observed that there is some relation among the keywords. There are three different forms of contextualization that we consider:

1. users provide more than one specification for a concept in a keyword query (for instance, they may use the keyword `Person` before the keyword `Name` to specify that they refer to the name of a person. This allow the engine to map the keyword `Person` to the table Person and the keyword Name to the attribute Persone.Name);

2. the keyword corresponding to value database terms, which follows a keyword corresponding to a schema attribute term, it express a value of the domain of that attribute. (for instance, they may use the keyword `Name` before the keyword `John` to specify that they refer to an entity (in this case a person) named John. This allow the engine to map the keyword `Name` to the attribute Name and the keyword John as a value of the attribute Name);

3. the keyword corresponding to value database terms, which follows a keyword corresponding to a schema table term, it express a value of the domain of the most representative attribute of that table (for instance, they may use the keyword `Person` before the keyword `John` to specify that they refer to a person named John. This allow the engine to map the keyword `Person` to the table Person and the keyword John to the value of the attribute Persone.Name);

Before we follow the description of the contextualization process we will introduce a set of definitions:

- K the qyery keywords list;
- K^S the subset of K containing the keywords mapped to database schema terms;
- K^V is the remaining set of keywords from K containing the keywords mapped to database attribute domain ($K^V = K \cap K^S$);
- WV is the value weighted matrix which defined the confidence between each keyword and a database domain attribute;
- $WV_{reduced}$ is the value weighted matrix excluding the keyword which are mapped to a database schema term;
- M_i^S a partial mapping of keywords in K to schema terms;
- T(k) is defined as the *free trailing keywords* of a keyword k, the maximum set k_1, k_2, $\ldots k_m$, of consecutive keywords in K that are between two keywords k_s and k_e in the query and for which $k_s = k$, M_i^S is defined for k_s and k_e and undefined for every k_i with $i = 1..m$.
- P(k) is defined as the *free preceding keywords* of a keyword k is defined accordingly to the previous definition, with k playing the role of k_e.
- Δw a constant value by which increase the weights of the trailing and preceding keywords $T(k)$ and $P(k)$ for terms representing the domains of the attributes of the relation R for every keyword k mapped to a relation R through M_i^S.

As an initialization step, all the weights in the rows of VW corresponding to keywords already mapped to database terms are set to zero. This is done to guarantee that in the three phases that are described next, none of these keywords will be mapped to a value term.

In the first phase of the contextualization, for every keyword k mapped to a relation R through M_i^S, the weights of the trailing and preceding keywords $T(k)$ and $P(k)$ for terms representing the domains of the attributes of the relation R are increased by a constant value Δw. The rational of this action is that queries typically contain keywords that are generic descriptions for the values they provide. For instance, "`Person`

Algorithm 2: Contextualizing Value Weight Sub-Matrix VW

Input: M : Partial keyword assignment to schema database terms
Q : Keyword Query
SW : Schema intriscic weight sub-matrix
VW: Value intrinsic weight sub-matrix
Output: Updated VW sub-matrix

COMPUTECVW(SW,VW,M)
(1) **foreach** $k \in Q$
(2) $t \leftarrow x$: $\exists\,(k, x) \in M$
(3) **if** $t = null$
(4) **continue**
(5) **if** t is a relation R
(6) **foreach** attribute A of R
(7) **foreach** $k' \in T(k) \cup P(k)$
(8) $VW[k', \underline{R.A}] \leftarrow VW[k', \underline{R.A}] + \Delta w$
(9) **else if** t is an attribute A of a relation R
(10) **foreach** attribute A' of R with $A' \neq A$
(11) **foreach** $k' \in T(k) \cup P(k)$
(12) $VW[k', \underline{R.A}] \leftarrow VW[k', \underline{R.A}]$ + Δw
(13) **foreach** attribute A' of R' s.t. $\exists$ join path from A' to A
(14) **foreach** $k' \in T(k) \cup P(k)$
(15) $VW[k', \underline{R.A}] \leftarrow VW[k', \underline{R.A}]$ + Δw

`Bill`", "`Department CS`", etc., which means that consecutive keywords may correspond to a relation and a value of one of that relation's attributes.

During the second phase, for every keyword k mapped to an attribute A through M_i^S, the weights of the trailing and preceding keywords $T(k)$ and $P(k)$ with the database terms representing domains of attributes in the same relation as A are also increased by a constant value Δw. The rational of this rule is that consecutive keywords may represent value specifications and related values. An example is the query "`Name Bill Databases`" intended to ask about the person Bill who works in the area of databases.

		Same table ID	Same table	FK connected ID	FK connected
FW	$d = 1$	0.75	0.50	0.30	0.18
	d>1	-	0.25	0.12	0.12
BW	$d = 1$	0.37	0.25	0.17	0.09
	d>1	-	0.12	0.06	0.06

Table 1.5: Contextualized Value Weights VW

	P.Na	P.Ar	P.Ph	P.Ad	P.Em	D.Dn	D.Ad	D.Di
$workers$	0	0	0	0	0	0	0	0
$department$	0	0	0	0	0	0	0	0
CS	30	18	0	18	50	75	50	50

Table 1.6: Contextualized Value Weights VW

In the third phase, if a keyword k is mapped to an attribute A, the weights of the trailing and preceding keywords related to domains of attributes related to A through some join path are increased by the constant value Δw. The rational is that users use keywords referring to concepts that are semantically related, even if these concepts have been modeled in the database in different tables. An example of this situation is the keyword query "`Phone Tribeca`". If `phone` is mapped to the attribute Phone of the relation Person, then the keyword `Tribeca` most likely represents the address of the department, which is stored in a separate table, and then the keyword query is about finding the phone number of the department that is on Tribeca street. Note that the weight increase Δw can also be a percentage, instead of a constant, but our experimentations have showed no significant differences. The three phases described above are illustrated in pseudocode in Algorithm 2.

Example 1.7.1 *Table 1.6 illustrates the VW sub-matrix after the value weights of Table 1.2 have been contextualized based on the mapping that maps the keyword* `person` *into* workers *and* `department` *to* department*. Note that the lines for keywords* `person` *and* `department` *are 0, a result of the initialization step.*

1.8 Selection of the Best Mappings

Given a weight matrix, computing the best possible mapping of keywords to database terms is known as the *assignment problem* [18]. Unfortunately, the traditional solutions return the first best mapping while we would like them all in descending order, or at least the top-k. Furthermore, a solution that, during the computation of the best mappings takes into consideration inter-dependencies of the different assignments (i.e., the contextualization process) is required. For this reason, the popular systematic Hungarian (a.k.a. Munkres) algorithm [2], has been adapted, in order not to stop after the generation of the best mapping but to continue to the generation of the second best, the third, etc. Furthermore, some of its internal steps have been modified so that the weight matrix is dynamically updated every time that a mapping of a keyword to a database term is decided during the computation.

The execution of the algorithm consists of a series of iterative steps that generate a mapping with a maximum score. Once done, the weight matrix is modified accordingly to exclude the mapping that was generated and the process continues to compute the mapping with the second largest score, etc. More specifically, the maximum weight of each row is first identified and characterized as *maximum*. If the characterized as maximum weights are all located in different columns, then a mapping is generated by associating for each of the characterized as maximum weights the keyword and the database term that correspond to its respective row and column. On the other hand, if there is a column containing more than one weight characterized as maximum, all maximums in the column except the one with the maximum value loose their characterization as maximum. This last action means that some of the rows are left without some characterized weight. The values of the weights in these rows are then updated according to a number of contextual rules mentioned in Section 1.5. This is the effect of the mapped keywords to those that have remained unmapped.

In the sequel, for each row with no characterized weight, the one with the maximum value that belongs to a column corresponding to a database term different from the previous one is selected and characterized as *maximum*. If this leads to a matrix that has each characterized weight in a different column, then a mapping is generated as above or the process of un-characterizing some of the values as previously described is repeated.

Once a mapping of all the keywords has been formed, it is included in the set of mappings that will be returned by the algorithm. Then, the algorithm needs to be repeated to generate additional mappings. To avoid recomputing the same assignments, the algorithm is re-executed cyclically with a new matrix as input. The new matrix is the old one modified to exclude mappings that have already been considered in previous runs of the algorithm. This is done by setting the values of the respective weights to a large negative number, forcing the algorithm to never select them again. This whole process is repeated until the scores of the mappings that the algorithm generates fall below some specific threshold. By construction, the most prominent mapping is the one that is first reported by this task.

The original Hungarian algorithm for rectangular matrices has an $O(n^2 * m)$ complexity [2], where n is the number of keywords and m is the number of databases terms. Extending the algorithm to consider dynamic weights as described above brings the complexity to $O(n^3 * m^2)$ which is due to the fact that a mapping may affect any other mapping, thus, in the worst case, $(n-1) * (m-1)$ weight updates may take place. Nevertheless, this worst case rarely happens since only a subset of the matrix is updated in each iteration, and, due to the threshold, not all the possible updates are evaluated.

Algorithm 3 depicts the overall process of computing the set of most prominent mappings of a set of keywords to database terms, given a weight matrix. The expression $HUNGARIAN_{ext}$ refers to our extended version of the Hungarian algorithm.

Example 1.8.1 *Table 1.7 illustrates a VW matrix similar to the one of Table 1.6, but with additional keywords to better demonstrate the steps described in this section. The initial version of the matrix is the one composed of the first 7 lines. The lines of the keywords workers and department will remain unchanged since the keywords are mapped to schema terms, and for this reason they are omitted from the subsequent versions. The weights in white cells are those characterized as maximum. Each one is the largest weight in its row. Note that for column* P.N *there are more than one weight characterized as maximum. From those, only the largest one is kept, in our case the 46. This leaves the row of keyword* `Hopkins` *with no weight characterized as maximum. The result is the second VW matrix illustrated in Table 1.7. The three characterized weights suggest a mapping for the keywords* `CS`*,* `Mary` *and* `Summerhill`*. Given these mappings, the weights are adjusted to reflect the inter-dependencies according*

Algorithm 3: Keyword to db term mapping selection

Input: $I(i_{ij})$ where $I \equiv SW$ or $I \equiv VW$
Output: $M^I = \{M_1^I, \ldots, M_z^I\}$: Mappings generated by I

MAPPING(I, W_{MAX})
(1) $tempM = \bigcup i_{pt} \leftarrow HUNGARIAN_{Ext} * (I)$
(2) $W \leftarrow \sum i_{pt}$
(3) $M^I \leftarrow tempM$
(4) **if** $(W > c * W_{MAX})$
(5) $W_{MAX} \leftarrow W$
(6) **while** $(W > c * W_{MAX})$
(7) **foreach** $i_{pt} \in tempM$
(8) $i_{pt} \in I \leftarrow -100$
(9) $Mapping(I, W_{MAX})$

to the contextual rules. For instance, the mapping of `CS` *to the database term* D.D*, triggers an increase in the weights of the database terms on the attributes in the same table. The result of firing the contextual rules is the third matrix in Table 1.7. In the updated matrix, the highest value is 49, which is in the column* P.N*, but cannot be chosen since the keyword* `Mary` *is already mapped to it. The same applies for the second and third largest weights in the row, which are 42 and 39, respectively. The fourth largest weight, 37, is in a column of a database term that no keyword is mapped to, and it is becoming characterized as maximum. The final outcome is the fourth matrix of Table 1.7 and, based on this, the mappings of keywords* `CS`,`Hopkins`, `Mary` *and* `Summerhill` *to the database terms* D.D*,* D.Di*,* P.N *and* P.Ad*, respectively, which is added to the mapping results generated by the algorithm. After that, the mapping is again considered for generating a new input for the algorithm. Four new matrices are derived from it, each one having one of the weights that was in a white cell in the last matrix reduced. For each obtained matrix, the whole process starts again from the beginning.*

	P.N	P.A	P.P	P.Ad	P.E	D.I	D.D	D.A	D.Di
$workers$	0	0	0	0	0	0	0	0	0
$department$	0	0	0	0	0	0	0	0	0
CS	30	18	0	18	0	50	75	50	50
$Hopkins$	45	30	0	35	10	32	40	35	39
$Mary$	46	20	0	5	7	20	25	30	40
$Summerhill$	10	7	0	34	22	20	10	32	15

CS	30	18	0	18	0	50	75	50	50
$Hopkins$	45	30	0	35	10	32	40	35	39
$Mary$	46	20	0	5	7	20	25	30	40
$Summerhill$	10	7	0	34	22	20	10	32	15

CS	30	18	0	18	0	50	75	50	50
$Hopkins$	49	34	0	39	14	34	42	37	41
$Mary$	50	24	0	9	11	22	27	32	42
$Summerhill$	14	11	0	38	26	22	12	34	17

CS	30	18	0	18	0	50	75	50	50
$Hopkins$	49	34	0	39	14	34	42	37	41
$Mary$	50	24	0	9	11	22	27	32	42
$Summerhill$	14	11	0	38	26	22	12	34	17

Table 1.7: Weight matrix during best mapping computation

1.9 Evaluation

The proposed technique has been implemented into the Keymantic system [**?**] and numerous experiments has been performed to analyze its efficiency and effectiveness. The experiments were performed on a 32bit Centrino Duo vPro Windows machine with 3GB of RAM. For the experiments two real database was selected: the first was a university database containing information about courses, professors, students, publications, employees and other academic issues and the second database was a fraction of the popular IMDB service that is publicly available from the respective web site (www.imdb.com). This Keymantic approach heavily depend on the user semantics, thus, generating synthetic dataset was not an effective alternative. 29 real users were asked to provide a set of keyword queries for these databases alongside a natural language explanation of what they were looking for. In order to avoid influences and reduce noise to the queries provided by the users, they had no access to schema information and they neither were technical experts. Only a high level explanation of what the database contents were about had been provided to them. A database expert translated each natural language explanation of what the user was looking as it had been provided by her, into SQL. Since many queries were vague, high level and of exploratory nature, some of these queries were translated into more than one SQL query. The answer expected for each keyword query was the result of evaluating the SQL queries that the expert had created, and this was used as a reference to evaluate the results returned by our implementation. A total of 112 and 80 queries for the university and the IMDB database, respectively, has been used.

It is important to note that evaluating a keyword search technique for relational databases is a challenging task, mainly due to the lack of standards [26]. The task is becoming more challenging in this case since existing techniques have different assumptions and goals, making sometimes unfair any direct comparison among them. One of the reasons is that most of these techniques assume the availability of the instance.

[The importance of Schema Information] Among the keyword search engines analysed so far, only SQAK [8] take into account the possibility to have keywords expressing schema information. Analysing the keyword provided by the user, has been observed that a significant number of keywords in keyword queries correspond, not

to the data, but to meta-data (schema) information. This reflects the necessity the user have t access aggregated data instead single instances. In the keyword queries provided by the users we found that a third of the keywords are typically referring to schema terms. For queries with 3 keywords, the average number of keywords related to schema and value database terms was 1.83 and 1.19, for the two respective datasets.

[Effectiveness] To measure the effectiveness of the technique used in Keymantic, two different test has been done. The first aimed to observe the number of queries that generate the intended answers (as specified by the users and created by the expert) were among those our algorithm generates. The second test regards the number of queries, among those of the first group, appear in the first place (have the maximum rank). In case a keyword query had an answer generated by more than one SQL query, we considered in the counting multiple copies of the keyword query each one with only one of these SQL queries.

To retrieve the top-k result list, the Algorithm 3 used to generate the configuration is driven by a threshold parameter c. This parameter determine how far from the best result (the result with the maximum rank) we want to go. It takes values from 0 (all the solutions) to 1 (only the best solution) determining the minimum rank $RANK_{threshold}$ where result are taken into account ($RANK_{threshold} = c * RANK_{maximum}$). This threshold parameter plays an important role in the effectiveness of our approach. When it was brought to zero, every expected answer to a keyword query was included in the result. The percentage of keyword queries that returned the expected answer in the first position was not much different from the one in the case of a threshold with some default value. To realize the effect of any threshold modification, the threshold has been reduced by 10% and experiments repeated the university database.

The results are graphically illustrated in Figure 1.4. The "1st position" refers to the percentage of the keyword queries in which we found the expected interpretation in the first position. The "not in 1st position" indicates the percentage of the queries in which our algorithm did generated the expected interpretation but did not return it in the first position. Finally, the "not found" indicates the percentage in which the expected interpretation was not returned at all in the results produced by our algorithm. As expected, the smaller threshold increased the percentage of keyword queries with correct answers in the returned set, from 13.8% to 24.14%. However, it should be noticed that, with this threshold change, the number of possible interpretations that

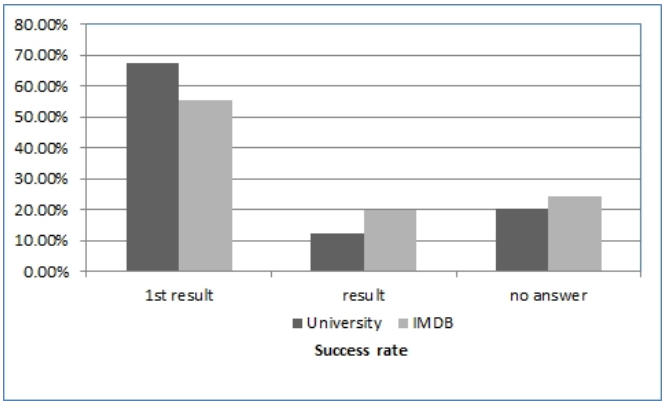

Figure 1.4: Results

were returned also increased. In the IMDB scenario, we obtained worst results than in the university scenario. This was mainly due to the fact that the IMDB schema consists of many nested tables with the same structure, i.e., same name and very similar data-types.

[Efficiency]To measure the efficiency of Keymanitc, its time performance has been studied with respect to three different parameters: the query characteristics, the size of the database and the Algorithm 3 threshold parameter. For the first experiment, the keyword queries considered for the evaluation differ not only on the number of keywords, but also on their type, i.e., whether the keywords correspond to schema or value terms. Moreover, fractions of different sizes from our original evaluation database has been created in order to observe the influence of the database size. The performance measurements are illustrated in Figure 1.5.

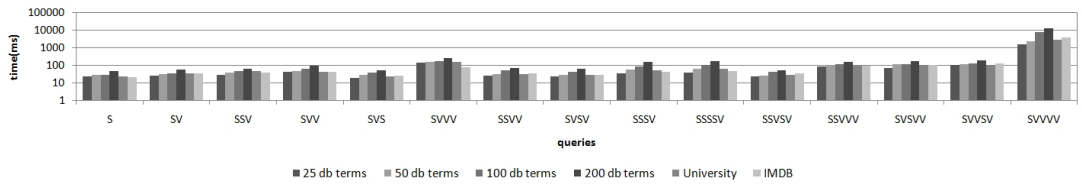

Figure 1.5: Time Performance for different database sizes and different query kinds

Each query is characterized by a set of keywords. Each keyword in the query can be S, if it is intended to be a schema term, or V, if it is intended to be a value term. For instance, the string SVS represents a keyword query in which the first and last keyword represent schema terms while the middle a value (for example: *"department*

CS dean"). As expected, for queries with keywords expressing mainly database term is not expected to be excessively high (see Figure 1.6). Instead, for queries with keywords expressing mainly value terms the time performance may be an issue. As shown in the result in Figure 1.6, the time increase for this group of queries is not extremely high. In general, this increase should have been factorial (ref. Section 1.3), but the experiments showed no such increase, which is because the metadata and the auxiliary information alongside the contextual rules are significantly restricting the number of possible mappings of keywords to database terms. Only in the last case, where there was one keyword referring to a schema element and four keywords referring to data values, the response time increase is significant.

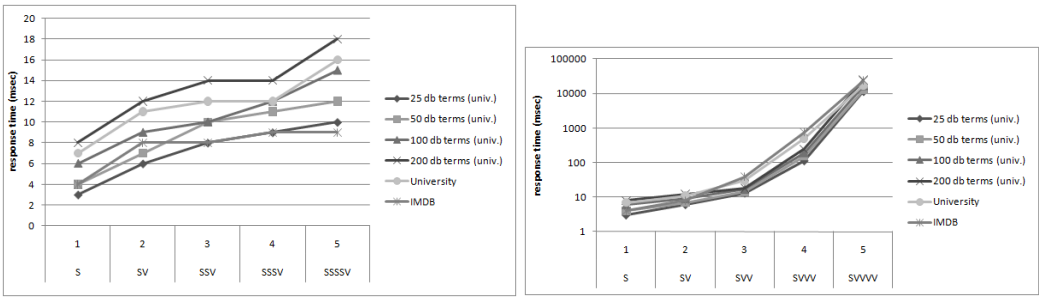

Figure 1.6: Queries with mainly database terms (right) and value terms (left).

However, the performance as well as the overall time needed to answer a keyword query has been measured. The overall time was highly related to the number of results retrieved and the number of join operations required. It ranged from some tenths of a second for selective queries involving one table to a few minutes for general queries. For example, in the IMDB scenario, the keyword query ``film'' took 1.9 sec and retrieved 166936 records, while the query ``actor film 2007'' took 1 min and 40 sec and retrieved about 2.7 million tuples.

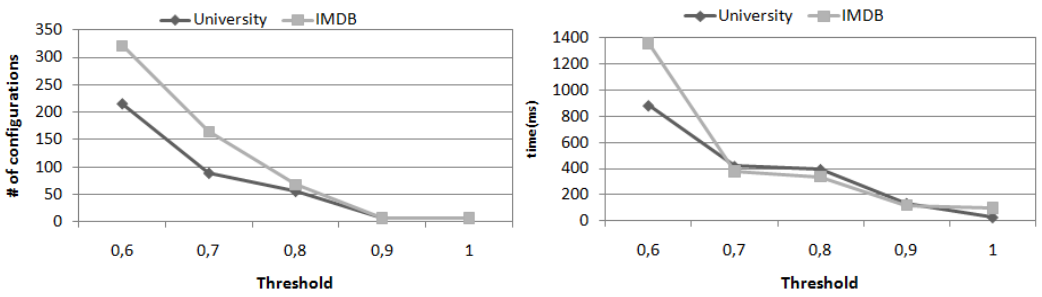

Figure 1.7: Configurations generated and time performance for different threshold values.

For the second experiment, to evaluate the effect of the threshold value in the behaviour of our system, experiments has been performed with the same keyword query but with different threshold values. As expected, the increasing of the threshold value, reduces the number of results and the response time. This because it generated only the best solutions regarding the maximum rank. The algorithm is monotonic descendent so the best solution is expected to be generated first. As shown in Figure 1.7 the number of result increase almost linearly to the decrease of the threshold value. This bring to a good dispersion of the result considering the ranking evaluation. A study of the results allows us to observe that different number of configurations can be obtained for the same threshold value. The reason of this discrepancy is mainly due to the different keyword queries with which the scenarios were evaluated and the different schemas of their respective databases. In particular, the university database is composed of few tables with a large number of attributes and the IMDB database contains many related tables with few attributes. Since the contextual weights used for testing consider the relevance of the attributes in the same table higher than the attributes in related tables, searches in flat schemas generate more configurations than in nested schemas. for the same values of thresholds and coefficients.

Query	DBX (NoI)	DBX (WiI)	KM (NoI)	KM (WiI)
actor	0	2	1	3
actor 2009	0	6	2	0
actor worthington 2009	0	12	7	0
film	0	2	1	3
film avatar	0	3	2	3
film avatar 2009	0	12	15	1
film avatar worthington	0	12	10	0
film avatar worthington 2009	0	25	129	0
film avatar actor	0	12	10	1
film avatar 2009 actor	0	37	26	1

Table 1.8: Results by DBXplorer (DBX) and Keymantic (KM) System with (WiI) and without (NoI) access to the data instance.

[Qualitative comparison to other approaches] In contrast to the approaches developed so fa, the Keymanitc approach is the first to operate under the assumption that there is no prior access to the relational database instance. Under such a condition,

existing works will return no results since they will have no way to determine where the keywords appear.

Is some work, the exploiting of schema information has been taken into account [27] but with the intention to optimization the research after they find the keyword in the database instance. Moreover they consider only exact term matching between the keyword and the database term. Thus, a direct comparison in terms of effectiveness and efficiency would be unfair. Nevertheless, to realize a qualitative comparison, we run some experiments using two popular systems for keyword searching in relational databases, DBXplorer [12] and DISCOVER [11], alongside Keymantic system. As the result generated are the same only the former has been reported. The outcome is illustrated in Table 1.8. The first column indicates some of the queries we run. The second and the third columns represent the result with and without access to the data instance respectively for DBXplorer and forth and the fifth columns represent the result with and without access to the data instance respectively for Keymantic.

The first experiment run was with DBXplorer assuming that the relational database was not allowing access to its instance so that DBXplorer could not create its index. As such, all the queries executed returned zero results. Then, such access was allowed and the experiment repeated. The number of SQL queries generated are reported in the second column. The same queries was executed with the Keymantic system. Naturally, assuming no access to the instances. The number of queries generated by Keymantic was larger than the one generated by DBXplorer. This was expected since the former was using no information about the existing data values and was making more generic guesses of possible semantically meaningful queries.

The second experiment was performed the Keymantic system. Naturally, assuming no access to the instances. The number of queries generated by Keymantic was larger than the one generated by DBXplorer. This was expected since the former was using no information about the existing data values and was making more generic guesses of possible semantically meaningful queries. Allowing the access to the data, Keymantic was able to identify and eliminate the queries generated by mappings of certain keywords to value terms that were leading to zero results. As it can be observed, the number of queries generated by Keymantic was less than the number of queries generated by DBXplorer since in this case some queries that were not very likely to occur

semantically were eliminated by the threshold (the parameter c in Algorithm 3). The same number of queries was obtained bringing the threshold to zero.

1.10 Prototype implementation

When the user runs the Keymantic protorype, the user interface shown in Figure 1.8 will appear. There are four tabs where the user can navigate. The "Keyword" tab is used to insert the query keyword. The "Advanced" tab is used for an advanced search where the user can choose the configuration that is interested to explore and the possible joining path tree. The "Result" tab is used to visualize the data retrieved form the database and "Configuration" tab is used to change the configuration parameters and chance the system behaviour.

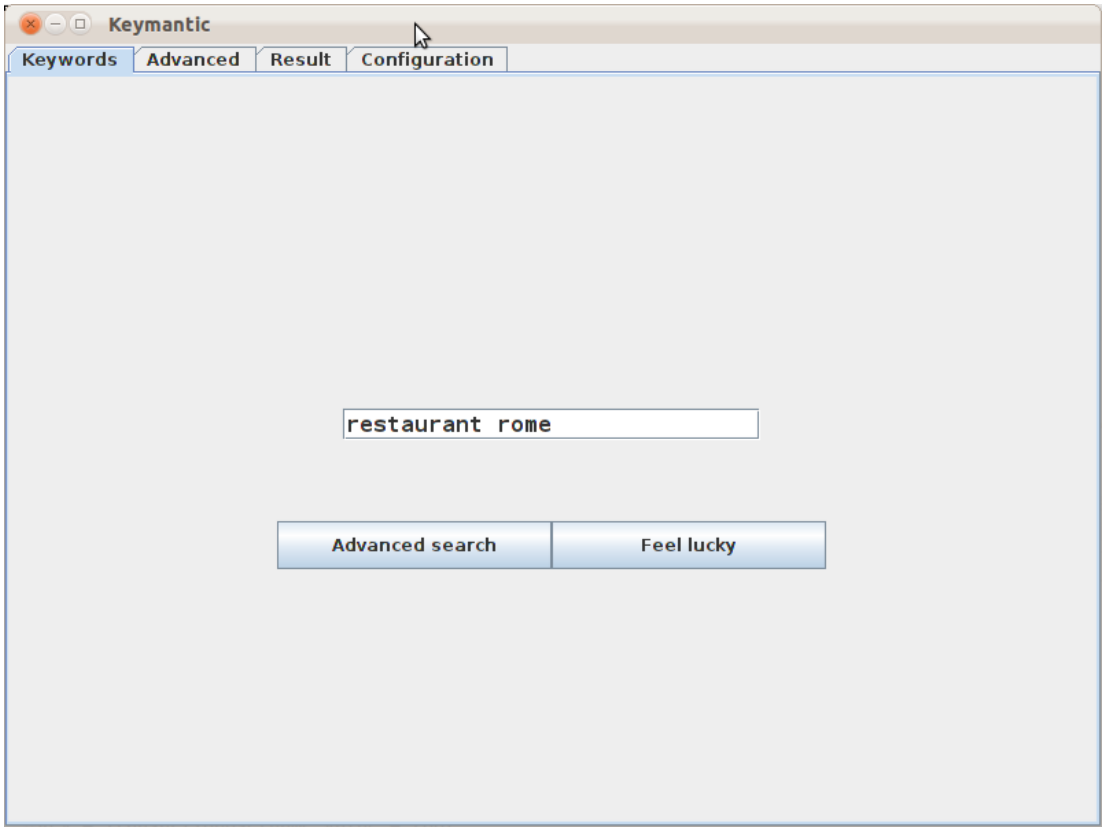

Figure 1.8: Main user interface

To better understand the usability of the prototype a short example will be introduced. The ER schema in Figure 1.8 on "Graph" section, represent a database containing restaurants, hotels and the persons accommodating to the hotel or having dinner to the restaurant. There are four tables: Person, Hotel, Restaurant and City connected via

Foreign Key Primary Key relationship. By this schema, the user can search restaurants and hotels, their location and the person who have frequented.

Let me suppose that the user is looking for a restaurant in London and pose the following query keywords “restaurant rome”.

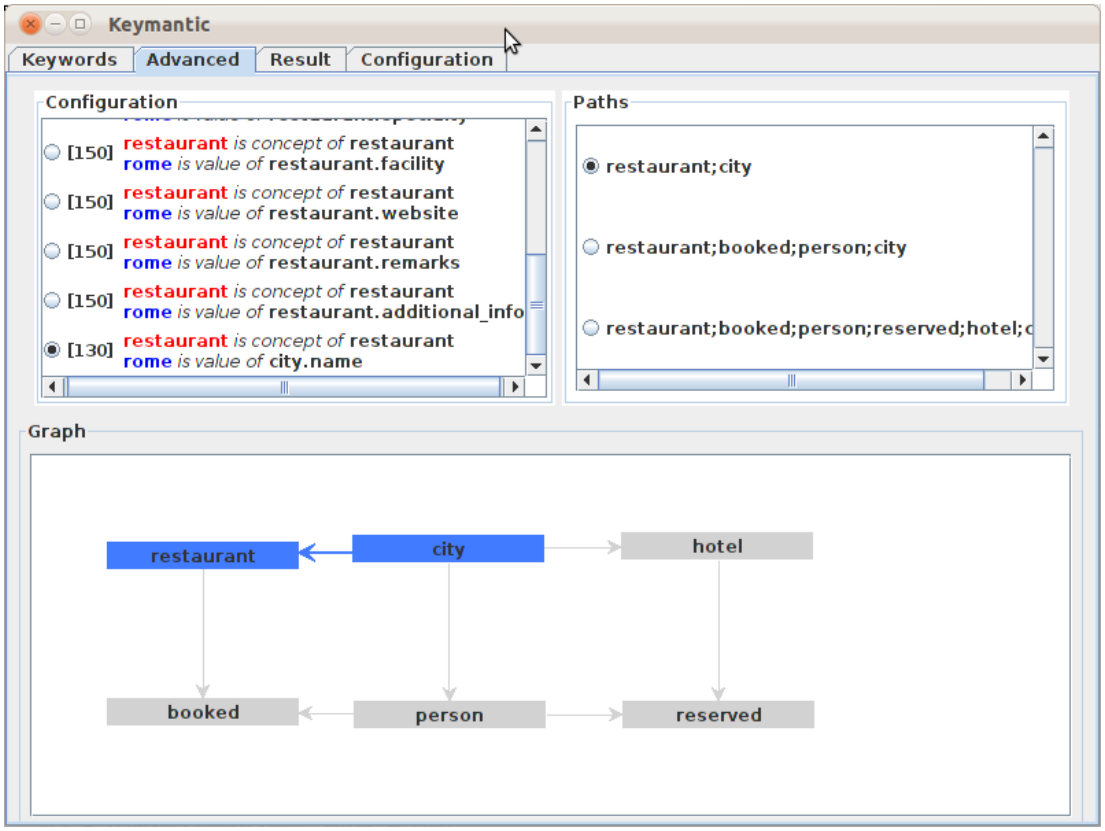

Figure 1.9: Advanced search interface

The intention of the user is ambiguous. He/she may be looking for a restaurant named “Rome”, of placed in Rome city or in Rome street. In Figure 1.9, in the “Configuration” box the user may specify the configuration that most fits to his/her research. The configurations are ranked in descendent order. If the user is looking for a restaurant in Rome the one in the Figure will be selected where “restaurant” represents the table “Restaurant” and the keyword “rome” represents the attribute the value of the attribute “City.Name”.

Later on, the user may choose which path to explore, at the “Path” box. The first path shows the connection between the table “Restaurant” and “City” and the user

may retrieve all the restaurants placed in London. The second one will retrieve all the persons that lives in Rome and have had dinner at a restaurant and so on.

In the "Graph" box, the user will visualize graphically the ER schema the relations involved in the joining path tree.

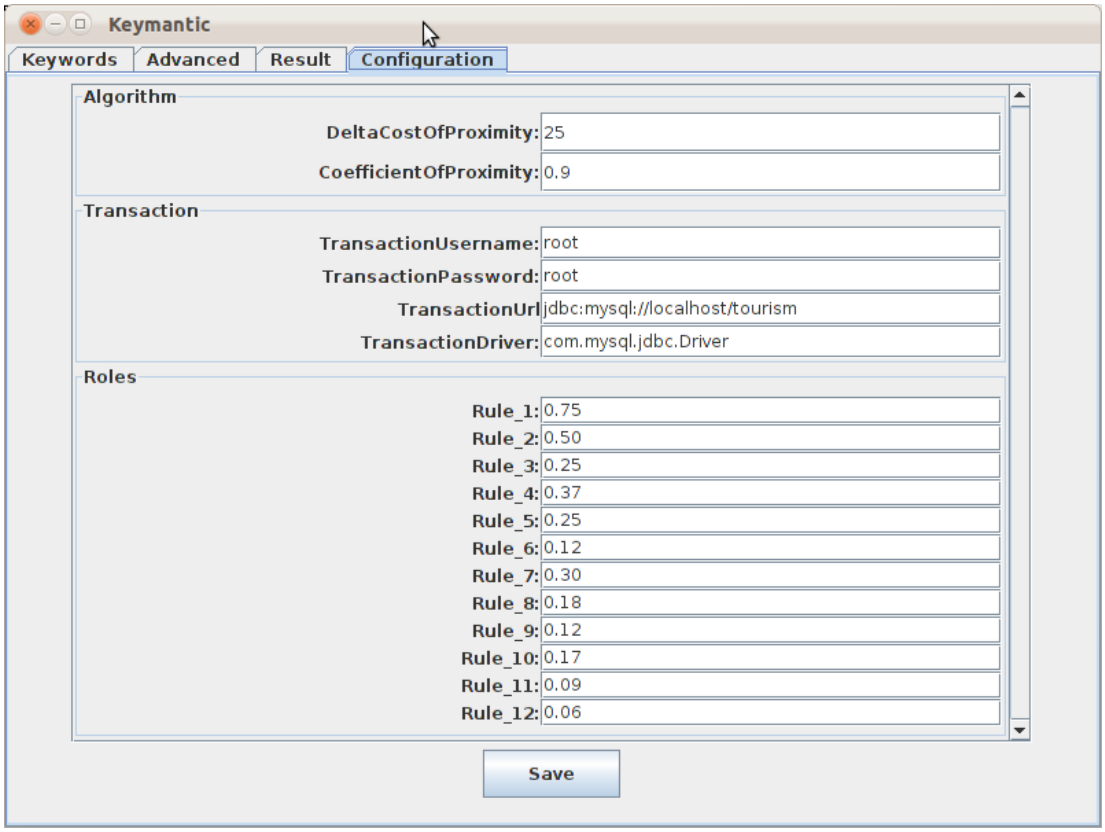

Figure 1.10: Configuration interface

In Figure the 1.10 the user may change the configuration parameters and influence the system behaviour.

2

Ubimedic2

The healthcare system is one of the most complex systems managed by the government with the aim to offer a set of services to the population. These services include diagnosis, treatment and prevention of disease, illness, injury, and other physical and mental impairments. The high complexity of this system is given by the necessity of a large-scale coordination over distributed resources. Besides the complexity, some activities are considered critical as time and coordination play a very important role. One of these activities is the Fist Aid service. Providing this service becomes challenging in territorial emergencies and large-scale disasters where a very prompt and coordinated response is necessary in order to reduce the serious damage of people health involved in disasters and to save as many lives as possible.

Many can be the disaster situations such as large scale accidents, calamities and terrorist attacks. Recent calamity events such as the Indonesian (2004) and Fukushima tsunami (2011), Haiti earthquake (2010), and New Orleans hurricane (2005) are characterized by wide destroyed territories and large number of life lost. Despite the great research activity to predict time and size of these events, they are still challenging issues. Therefore, important efforts are continuously done by governments and research groups to come out with strategies and tools to face such events. As these events are quite different from one another, common strategies and strict protocols are not feasible solutions. Coordination is left to trained people who follow some guidelines but their ability and flexibility in taking decision is fundamental.

In particular, the actors involved in a rescue operation are many and have different qualification not only in the health field, such as paramedic and medical staff, but also may include police department, fire department and civil protection department.

Considering the criticality of these situations, prompt response, coordination, cooperation and decision making are necessary. The promptness of the intervention is decisive for the succeeding of the rescue operations. All these actors must cooperate in order to reach their target. Decision making must be fast and accurate.

Another peculiarity of these scenarios is the dynamism of the events. The situation may change rapidly due to the increase of involved persons or with the worsening of their health condition. Situations are different from one another and almost unpredictable. All the actors must not loose the ability to cooperate and coordinate and to adapt their decision based on the new situations. They must be flexible in their decisions and still observing a set of standards and predefined roles.

The everyday growth of the technology and the widespread of electronic and digital medical devices make already possible to save and exchange digital information. Medical devices have their own memory or are connected with computers that collect and elaborate data. The information collected by these devices can be accessed remotely, even in real-time, using ad-hock networks, the Internet or other communication technologies.

A decision making support can be very useful for the human operators. This can save time, avoid human error and increase performance. In large-scale disasters, many operators come into help from other cites around and have no information about the local hospitals distribution for specific pathologies and the maximum number of patients that they can handle, the best way to reach one hospital and many other useful information.

To meet the requirements previously introduced, the use of the agent technology [28] [29] has been evaluated and, based on its peculiar characteristics [30], a suitable solution has been proposed to these problems. In the following, therefore, a multi-agent architecture, called Ubimedic2 is proposed, able to organize the complex work of rescue operations, taking the appropriate decisions.

2.1 Related Work

The complexity of the activities in the healthcare system has attracted many researchers who often have proposed different solutions over the different aspects. One of the activities that is still an open issue is the First Aid activity. Its main characteristic is the necessity of coordination of a set of units (single persons or groups) with a common target, to offer the best possible first aid support. Researchers in the multi-agent field are focusing on the application of the agent technology to the support of such activity. This interest is growing every day and is covering different aspects. A detailed analysis of the possibility of using agents to deal with the healthcare problems has been done in [31] [32].

The use of agents has widely been proposed in many activities in the healthcare system such as hospital patient scheduling [33] [34] [32] [35], patient remote monitoring [36] [37] [38], decision support system [39] [40] [41], healthcare knowledge based [42] [43]. Related to the disaster management several works can be found focusing on different aspects such as triage [44], fire brigade response strategies [45] and human evacuation procedures [46].

DrillSim [46] is a simulator based on the multi-agent architecture and focuses on the human evacuation in emergency scenarios. Humans are represented by agents, with their own state, rules, profile and social ties. Agents are able to collect information from the environment (representing the world) and elaborate a decision based on the strategies implemented on it.

DEFACTO [45] system focuses in the organization of the fire brigade work in large scale disasters. The system requires a 3D plan of the city and simulates possible solution and strategies the fire brigade must follow in order to turn off multiple fire sources. The system uses team work strategies for the dispatch of the fire tracks in the territory.

In particular, [44] focuses on data collection during the triage using electronic devices to measure blood pressure (electronic sphygmomanometer), heart rate and oxygen percentage in blood (oximeter). The three level architecture enables the measurement of medical data from embedded devices (*level 1*), collected by personal servers (*level 2*) and finally stored and elaborated by the operative centre (*level 3*). Some of the weaknesses of this architecture are: *i)* this system uses only Wi-Fi connections (which

is limited in space) to exchange data, therefore two or more devices need to be close to each-other in space because they cannot communicate in large distances and this limits the possibility to operate in real-time; *ii)* the architecture is centralized and there is no interaction between operators; *iii)* there is no support to decision making.

A similar work has been done in UbiMedic [47] [48] in order to create a framework that makes possible the communication between devices in a distributed client-server approach. The embedded devices (server side), enabled to collect health information from the patient, are accessible from other devices (client side) such as notebooks and PDAs. This framework provides a controlled access based on a set of policy roles. The weakness of Ubimedic is that it focuses only on data exchange and collection.

Ubimedic2 starts from this idea and extends it with an autonomous coordination mechanism with the aim to offer better services to patients in a disaster scenario. It provides the necessary functionalities in order to empower with new intelligent components. It has autonomy and decision making support.

2.2 Motivating example

In the following sections, a real study case will be analysed in order to highlight the weaknesses of the current approach and to motivate the necessity of a technological support to such scenario. A possible and common scenario can be a car accident in a highway involving many vehicles and people inside. Someone calls the emergency number reporting the event and asking for help. In large accidents, more people call the emergency number for the same event but this will be filtered by the human operator of the operative centre. With the guide of the human operator, the person who makes the call describes the event and gives as much information as possible about the place and the people involved. The information provided is commonly vague and not precise as people are usually shocked because personally involved in the accident or due to the scene.

After the operator collects the necessary information about the event, she decides the number of ambulances and which from the available ones to send to the target. Afterwards, she calls by phone or by radio one by one the ambulances giving the necessary information previously collected about the place and the possible number of persons involved and an evaluation of the criticality of the situation.

In some countries, the emergency number is unique for all kind of emergencies such as in US or there are different emergency numbers for different kind of emergencies such as in many European countries. Therefore, if the emergency number is unique the human operator must organize the intervention of the police force and fire team, otherwise she must contact the other departments. Figure 2.1 illustrates in the latter case, the current communication architecture between the operative centres and the mobile units (such as ambulances, police cars and fire trucks), hospitals and other departments. The mobile units cannot communicate directly with each other but only through the respective operative centre. In the following, the analysis will focus on the First Aid activity with the dedicated operative centre (European approach), mobile unites such as ambulances and hospitals.

Once the medical staff reaches the place, it gives a quick look to realise the degree of the emergency situation and an estimation of the number of people involved. Following a set of protocols for the on site triage, it collects more detailed information about every single patient and communicate to the operative centre. Based on

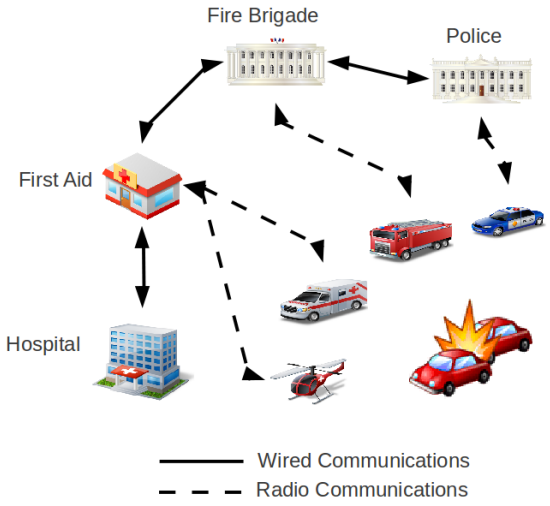

Figure 2.1: Current communication architecture.

the information acquired, the operative centre decides if other equipment is needed, such as more ambulances or a medical helicopter. During the rescue operation, the medical staff collects each single patient information about health parameters (such as heart rate, percentage of oxygen in blood and blood pressure) in order to detect the pathology and the seriousness degree. The data are written in some structured paper sheet for each patient. Currently, this information is not uniform and depends on the qualification of the staff (medical vs. non medical) and the associations operating in the territory (i.e., Red Cross).

The information collected must be exchanged with other staff operators, the operative centre and the hospital triage. All data to the operative centre are provided by medical staff using phone lines or radio channels. Voice communication is very slow and not free from errors, especially while spelling names (for this reason an international phonetic alphabet has been designed).

After the patient receives the first aid, she must be sent to the most appropriate hospital. The dispatch is done taking into account the seriousness of the patient pathology, the kind of treatment she must receive, the number of patients a particular hospital can support, etc. All these variables make the decision very hard to take in real-time. It must be done in collaboration between the operative centre and the staff present in the place who better knows the condition of the patients.

The number of mobile units (such as ambulances) to be employed is decided by the operative centre. If the available ones are not sufficient (with the constraint of a minimum number of available ambulances to cover the territory needs) the human operator must alert the standby staff.

The scenario described above gives an idea of the tasks that must be performed during a rescue operation. The weaknesses of this approach are the way communications occur and the lack of decision making support. The number of communications is large and is done by phone or radio. This means that only one communication at a time can be done and this is a real bottleneck. The lack of support to decision making can bring to errors and takes a lot of time when performed by a human operator. Information is collected in paper sheets making the data transfer difficult and not error free. It gives no way to statistics, digital data storage and information tracking. This approach is centralized and the operative centre can be single point of failure.

Analysing all these aspects that can occur during a disaster rescue operation and being aware of the criticality they carry, a new approach developed in Ubimedic2 has been proposed. It is able to support a large number of communications, digital data storage and transfer, and decision making support, thanks to the agent-based technology.

2.3 Architecture

In the previously introduced scenario, it can be observed a large number of actors involved (such as paramedical and medical staff, policemen and firemen) and a large number of communications with the respective operative centres (in many European countries the emergency departments are not unified). Each rescue unit has its own qualification and reports to its correspondent operative centre, which hardly share information with each other. All decisions are taken from the operative centre, which need as much information as possible to make the correct evaluation that will be communicated to the operative units.

Moving from the centralized to the distributed approach, a suitable technology to meet such requirements is the multi-agent one. An agent is a piece of software, placed in an environment and able to perform autonomous actions in order to reach its target. Agents are *reactive* (perceive the environment changes and act promptly), *proactive* (they take the initiative to undertake an operation) and *social* (they interact with other agents). Each operative unit can be modelled as an agent with its own behaviour (e.g. to distinguish the medical car from ambulance), that has a predefined target (bring the patient to the most appropriate hospital for long term treatment) and, cooperating with the other agents (in order to maximise the resources), tries to find the best solution and acts based on it. In the agent-based technology, this is called the BDI (Belief Desire Intention) [49] model where the *Beliefs* are the knowledge that the agent has from the environment and the world around, the *Desire* is the target (it can be more than one) the agent must find a solution for and the *Intentions* are the tasks the agent will undertake in order to fulfil its *Desire*. The collaboration among agents occurs through the exchange of messages in a P2P architecture where each node (agent) can request some data (client behaviour in the client-server approach) or offer a service (server behaviour).

The new architecture, designed for the Ubimedic2 approach, has the target to offer a fully distributed approach making use of electronic devices with the appropriate technology for facing the First Aid activity. Dealing with the distributed approach, all the operative units must take part to the coordination activity and handle part of this coordination. The operative units must use electronic devices to collect the necessary information, elaborate it and take the right decision compatible with the other units.

The autonomous characteristic of the agents puts the constraint of a self-sufficient capability to succeed at the defined task even in absence of communication or collaboration with other agents. The communication among the agents will help for a better use of resources.

The devices that will be used are medical devices (such as ElectroCardioGram (ECG) and Oximeter) and hand-handle devices (such as smart-phone and PDA) which are necessary to obtain health information by collecting different health parameters such as blood pressure, oxygen percentage in blood and other data provided by medical staff. The coordination will occur among the units such as ambulances, hospitals and the operative centre. The operative centre is still in charge to provide the emergency phone number, to collect the information from the people calling for help and to insert a new task to the system. The human interface with the population is mandatory for an efficient service. People calling for help are usually upset and need a human interaction to deal with.

2.3.1 Agents

Every element of the architecture (from medical device to operative unit) is represented by an agent. The developed agents have different degree of complexity, based on their task. The agent representing a medical device has a low degree of complexity as it must collect the data from the device and send to another agent, while an agent representing an operative unit must hold more logic, must be able to store and elaborate large quantity of data. The latter, referring to the BDI model previously introduced, must have the ability to adapt its behaviour depending on the situations and take decisions preserving the autonomy.

The agent representing the operative unit must support the human operator in the decision process. This may seem an issue to be implemented, but the well defined protocols that the medical staff must follow in such situations simplify the necessary strategy method to be implemented in the agent. In fact, human operators collect the necessary information and operate on the basis of these protocols. In the real life, protocols are necessary to avoid human errors and to get rid of responsibilities if things go wrong. Such protocols can be translated into instruction sequences giving to agents even a margin of self decision in case of unpredictable situations.

In the following, the agent class organisation shown in Figure 2.2 will be introduced. Each class can be extended to better specialize the component it represents.

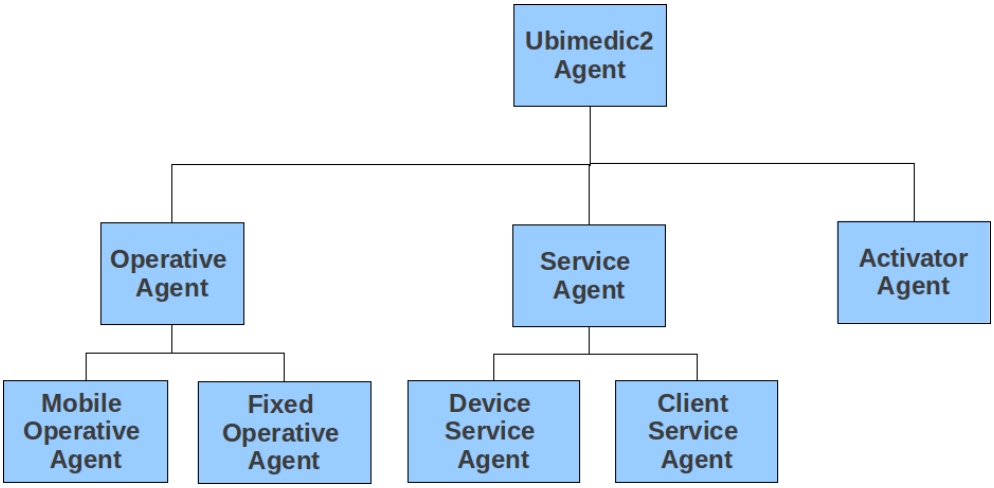

Figure 2.2: Agent class hierarchy

Service agent (SA) is the agent class that represents a device that can be a medical device, a notebook or PDA that a human operator uses to insert or read data. These devices have limited functionalities and there is no decision making. Based on their use, the corresponding agents are divided into two categories:

- *Device Service Agent (DSA)* is the agent class that represents the medical device (such as oximeter and ECG) in use by the medical staff. This agent, after collecting the data from the device sensors, makes them available to other agents. The data can be sent in two different ways: *one-shot* if the requesting agent needs the value on that exact moment (e.g. the hart rate), *periodically* if the requesting agent needs the data every *n* seconds, or *continuously* (such as ECG) where a continuous collection of data is required for that health parameter. This behaviour is close to the client-server approach that has been adopted from many companies in the medical industry. As many medical devices are customized by the company that produces it, Ubimedic2 gives the possibility in this case to adopt both the client-server approach and the P2P one considering the constraints put by the market.

- *Client Service Agent (CSA)* is the agent class that represents the client interface device of an operator such as a notebook, PDA and smart-phone. This agent has two main functionalities, the first is to retrieve the data inserted by the medical staff through a dedicated GUI and the second is to visualize data previously collected from medical devices. The human operator (paramedical or medical one), beyond the data she provides after evaluating the patient, needs to visualize the data coming from medical devices (DSA) such as the ECG trace, heart rate and percentage of oxygen in blood, and visualize them in the smart-phones or the PDA she is using.

Operative Agent (OA) is the agent class that represents an operative unit like ambulance, medical car, hospital, etc. This agent performs much more computation than a Service Agent, so, notebooks or thin clients must be employed. They need a database to store the data and a processor able to elaborate a considerable quantity of data. Besides the information collected, the OAs exchange messages and share information with each other in order to take the appropriate decision. OAs are divided into two categories:

- *Mobile Operative Agent (MOA)* is the agent class that represents a mobile unit like: ambulance, medical car, helicopter, etc. This agent collects the patient health information (from SAs representing devices mounted on the vehicle) and decides the pathology and the seriousness to be assigned to the patient. Afterwards, in collaboration with the other MOAs and agents representing hospitals, it will decide which is the proper hospital to bring the patient. Its behaviour may change during time as the patient condition may get worst. After a new elaboration of data, a new pathology or a new seriousness degree may be assigned to the patient and a new evaluation about the hospital may be taken.

- *Fixed Operative Agent (FOA)* is the agent class that represents a fixed unit such as hospital or temporary first aid camp. This agent communicates only with mobile operative agents. It receives the requests to accommodate the patients with a given pathology and communicates the availability of free resources. There is a phase of negotiation between the MOA and the FOA for the patient dispatch.

In additional to the SA and OA classes, Ubimedic2 defines another special agent class that is called *Activator*. It represents an operative centre that, in this distributed approach, has the task of collecting the different requests for help and for inserting a new task to be dispatched to one or more MOAs. Moreover, this agent evaluates the necessity of activating more operative units (such as standby medical staff) in case of large scale disasters or of satisfying the request with the ambulances almost available in the territory. Usually, if a set of green coded interventions are in queue, no new ambulances are activated but each task waits until an ambulance is free. Anyway, if this queue exceeds a threshold value (which means that the available ambulances are not sufficient to cover the needs of the territory) the appropriate number of units are activated. When the human operator of the operative centre receives a new call, she inserts a new task. The Activator agent, based on the seriousness code, dispatches it to MOAs representing the ambulances chosen for that task. Moreover, the Activator manages the requests for special medical equipment such as medical car or medical helicopter.

In Figure 2.3 there is a simple representation of the new architecture, agent representation and communication.

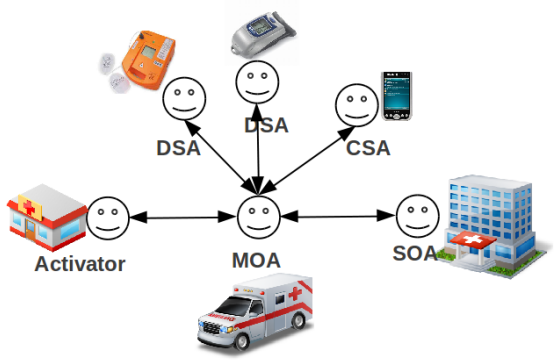

Figure 2.3: Agent and communication

2.3.2 Communication

The agent communication occurs using different technologies as shown in Figure 2.4. A service agent communicates with one operative agent providing necessary information. The environment where they communicate is limited in space. Devices placed

inside an ambulance are used by the ambulance itself. They can use a Wi-Fi or Bluetooth technology to exchange information.

The operative agent uses both GSM and Wi-Fi technology to communicate with other operative agents. The used technology depends on the distance between the operative agents. The ambulances that operate in the same place can communicate faster using the Wi-Fi technology. However, when they need to communicate with units that are far away from the place (i.e., hospitals), the operative agents use the GSM technology.

The RF technology is not taken into account because of the slow data transfer speed and high error rate. It can be used anyway for voice communication between human operators in distance but is not suitable for data.

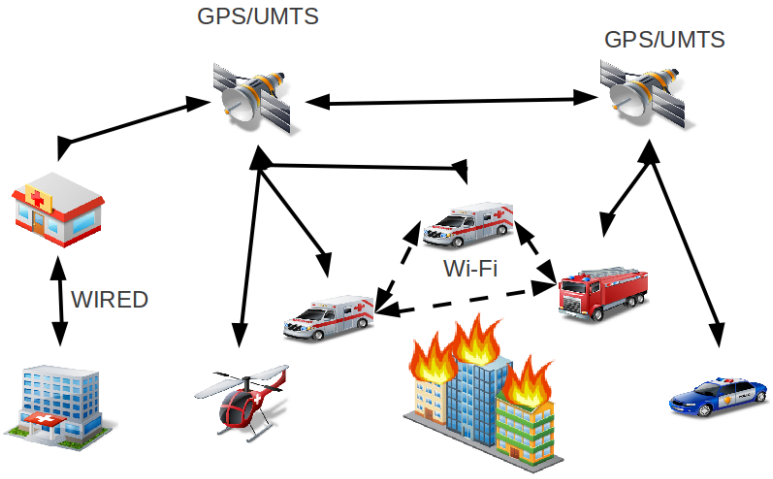

Figure 2.4: New communication model

2.4 Exploited technologies

2.4.1 Why JADE?

There are several agent platforms that enable programmer to develop Multi-Agent Systems. Each platform has its own peculiarities, strong and weak points. In the following a set of most known platforms will be shortly introduced and then the choice of using JADE is motivated.

Aglets [1] is a Java based mobile agent platform and library for building mobile agents based applications. It was originally developed at the IBM Tokyo Research Laboratory. The project is now open source and it is distributed under the IBM Public License. No new releases of Aglets have been made since 2001 and the future of the project is unclear. Aglets is not FIPA compliant.

Voyager [2] is a Java agent-enhanced object request broker. It has been developed by ObjectSpace in 1997. With Voyager, you can: a) remote-enable any Java class without modifying its code in any way; b) use regular Java message syntax to construct remote objects, send them messages and move them between applications; c) quickly create mobile agents that can roam a network and continue to execute as they move; and d) locate agents easily and send them messages as they move and work. Voyager is not FIPA compliant.

SeMoA [3] stands for "Secure Mobile Agents". It is about developing an extensible and open server for mobile agents. The server is written in Java, and agents can be written in Java as well (JavaSE). The focus is on all aspects of mobile agent security, including protection of mobile agents against malicious hosts. Another important feature is SeMoA's interoperability with other platforms such as Aglets and JADE. It uses advanced cryptography mechanisms and certification management for the malicious host detection, for the defence from denial-of-service attacks and for guaranteeing the communication integrity among agents.

Agent Factory Framework [4] is an open source collection of tools, platforms, and languages that support the development and deployment of multi-agent systems. The

[1] http://www.research.ibm.com/trl/aglets/

[2] http://www.inf.fu-berlin.de/lehre/WS99/VS/Misc/Voyager/API/

[3] http://semoa.sourceforge.net/docs/docs.html

[4] http://www.agentfactory.com/index.php/Main_Page

framework is broadly split into two parts: support for deploying agents on laptops, desktops, and servers (realised through Agent Factory Standard Edition (AFSE)); and support for deploying agents on constrained devices such as mobile phones and sensors (realised through Agent Factory Micro Edition (AFME)). The project is FIPA compliant and the development language is the Agent Programming Languages which have been built using the Common Language Framework.

JADE [1] (Java Agent DEvelopment framework) was born in 1997 with the aim to simplify the implementation of multi-agent systems. It is a free software framework (distributed by "Telecom Italia" under the terms of the LGPL v2) fully implemented in Java language and compliant with the FIPA specifications. It provides a set of graphical tools that supports the debugging and deployment phases. The agent platform can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another one when required. JADE is platform independent and do not support only personal computers and server but PDAs and smart-phones too. As they different virtual machine, ad-hock modules has been integrated to the basic platform.

The *Aglets* and the *Voyager* platforms have been excluded because they are not FIPA compliant and in addition their projects are no more supported for bug fixing and tutorial. The *SeMoA* platform focuses more on the security aspect and do not fully satisfy our expectations. The other two platforms, *JADE* and *Agent Factory* offer a complete development kit with user interfaces and debugging tools. Both of them offer integration to the Eclipse Development Environment, offer support to smart-phones, are FIPA compliant and platform (such as Windows/MacOS/Linux) independent. These characteristics make both feasible candidates. Among the two, the *JADE* platform has been chosen for two main benefits: the programming language and the documentation support. It uses the Java language which is very familiar to the majority of programmers and, considering the large number of researchers adopting this platform, there is a huge amount of documentation available on the Web offered not only by *JADE* developers but from users too.

[1] http://jade.tilab.com/

2.4.2 Platform architecture

The standard model of an agent platform, as defined by FIPA, is represented in the Figure 2.5.

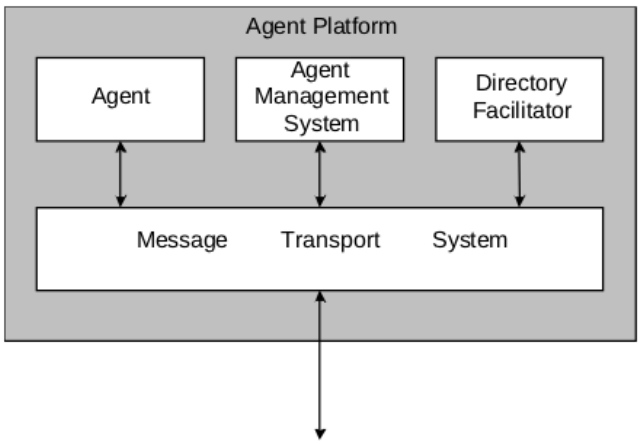

Figure 2.5: JADE Architecture

The **Agent Management System (AMS)** is the system agent who exerts supervisory control over access to and use of the Agent Platform. Only one AMS will exist in a single platform. The AMS provides white-page and life-cycle service, maintaining a directory of agent identifiers (AID) and the state of each agent. Each agent must register to the AMS in order to get a valid AID.

The **Directory Facilitator (DF)** is the system agent who provides the default yellow page service in the platform.

The **Message Transport System**, also called Agent Communication Channel (ACC), is the software component controlling all the exchange of messages within the platform, including messages to/from remote platforms.

The **Agent** class represents a common base class for user defined agents. Therefore, from the programmer's point of view, a JADE agent is simply an instance of a user defined Java class that extends the base Agent class. This implies the inheritance of features to accomplish basic interactions with the agent platform (registration, configuration, remote management, etc.) and a basic set of methods that can be called to implement the custom behaviour of the agent (e.g. send/receive messages, use standard interaction protocols, register with several domains, etc.).

JADE fully complies with this reference architecture and when a JADE platform is launched, the AMS and DF are immediately created. Furthermore the Messaging Service (implementing the ACC component) is always activated to allow message-based communication. The agent platform can be split on several hosts (see Figure 2.6). Typically (but not necessarily) only one Java application, and therefore only one Java Virtual Machine (JVM), is executed on each host. Each JVM is a basic container of agents that provides a complete run time environment for agent execution and allows several agents to concurrently execute on the same host. The main-container is the container where the AMS and DF live. The other containers, instead, connect to the main container and provide a complete run-time environment for the execution of any set of JADE agents.

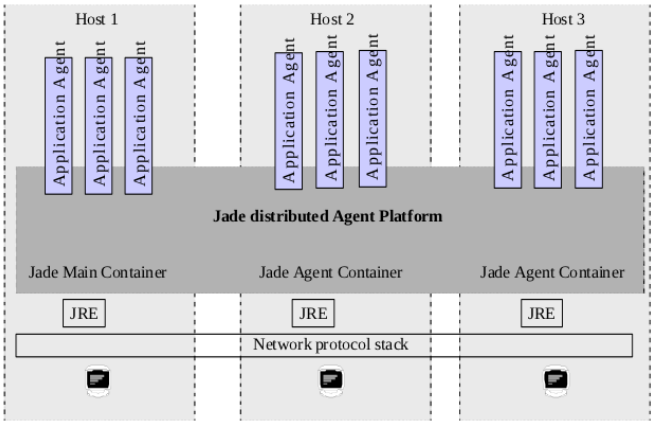

Figure 2.6: JADE Architecture on multiple hosts

2.4.3 Agent life cycle

A JADE agent can be in one of several states, according to Agent Platform Life Cycle in FIPA specification; these are represented in Figure 2.7 and are detailed below:

- *INITIATED*: the Agent object is built, but has not registered itself yet with the AMS, has neither a name nor an address and cannot communicate with other agents;
- *ACTIVE*: the Agent object is registered with the AMS, has a regular name and address and can access all the various JADE features;

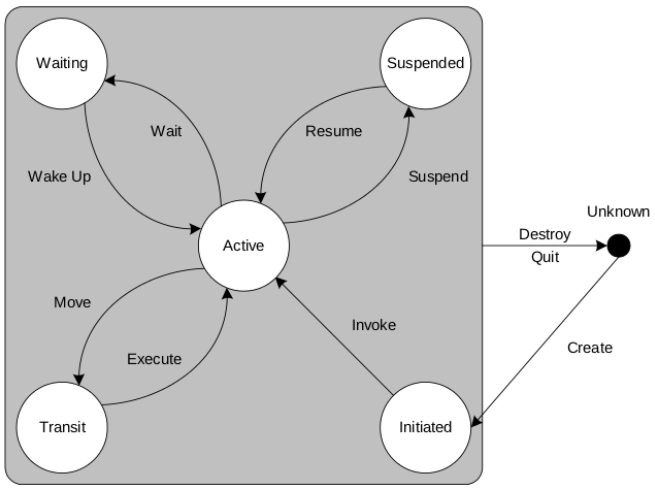

Figure 2.7: JADE Agent life cycle

- *SUSPENDED*: the Agent object is currently stopped. Its internal thread is suspended and no agent behaviour is being executed:
- *WAITING*: the Agent object is blocked, waiting for something. Its internal thread is sleeping on a Java monitor and will wake up when some condition is met (typically when a message arrives);
- *DELETED*: the Agent is definitely dead. The internal thread has terminated its execution and the Agent is no more registered with the AMS;
- *TRANSIT*: a mobile agent enters this state while it is migrating to the new location. The system continues to buffer messages that will then be sent to its new location.

2.4.4 Agent behaviours

The computational model of an agent is multi-task, where tasks (or behaviours) are executed concurrently. Each functionality/service provided by an agent should be implemented as one or more behaviours. A scheduler, internal to the base Agent class and hidden to the programmer, automatically manages the scheduling of behaviours. In the following, a brief description of the main behaviours is introduced:

- *SimpleBehaviour*: models simple atomic behaviours. Its reset() method does nothing by default, but it can be overridden by user defined subclasses;
- *OneShotBehaviour*: models atomic behaviours that must be executed only once and cannot be blocked. So, its done() method always returns true;
- *CyclicBehaviour*: models atomic behaviours that must be executed forever. So its done() method always returns false.
- *CompositeBehaviour*: models behaviours that are made up by composing a number of other behaviours (children). So the actual operations performed by executing this behaviour are not defined in the behaviour itself, but inside its children;
- *SequentialBehaviour*: this class is a CompositeBehaviour that executes its sub-behaviours sequentially and terminates when all sub-behaviours are done;
- *ParallelBehaviour*: this class is a CompositeBehaviour that executes its sub-behaviours concurrently and terminates when a particular condition on its sub-behaviours is met;
- *FSMBehaviour*: this class is a CompositeBehaviour that executes its children according to a Finite State Machine defined by the user.
- *WakerBehaviour*: implements a one-shot task that must be executed only once just after a given timeout is elapsed.
- *TickerBehaviour*: implements a cyclic task that must be executed periodically.

2.5 Implementation

A real implementation of the architecture introduced in 2.3 has been performed. The first step consisted in the implementation of a framework that gives the basic functionalities to all agents and manages the basic communication and coordination issues. The framework has been implemented in Java on JADE and most of classes are *abstract*. This because in a real or test scenario, the developer must subclass them in order to implement in detail the particular functionalities of the agents that reflects the coordination strategies adopted by a given country or region. Moreover, strategies may change over time and new agents can be developed and instantiated on the different devices or units they control. Besides the framework, a real system has been implemented for test reason and in absence of real devices, a set of custom user interfaces has been implemented to access the agent status and control their behaviour. All system classes that represent agents inherit from the framework classes. Moreover, some support classes (such as the Gps class) have been implemented to simulate the GPS coordination and location measurements.

2.5.1 Ubimedic2 layered overview

Ubimedic2 is implemented on top of the JADE platform and exploits its functionalities. JADE is written entirely in Java and that makes it node platform independent. Ubimedic2 can run on different nodes (such as personal computers or servers) having different operating systems (such as Windows, Linux or Mac OS). The architecture level in Figure 2.8 has been adopted as the Java byte code can be executed by the Java Virtual Machine (JVM) independently of the operating system and the architecture.

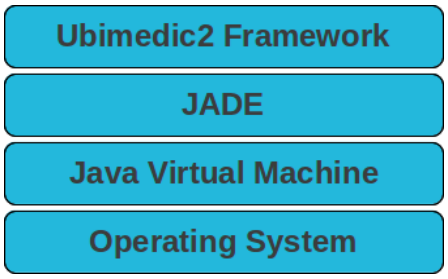

Figure 2.8: PC and server architecture

Operating systems on smart-phones and PDAs do not have the same JVM as on personal computers, so different add-ons to the basic JADE platform has been developed to support such operating systems such as Android and Widows Mobile (see Figure 2.9). The JADE community has developed a lighter version for the Android OS (written in Java) and Windows Mobile (written in C#), which enable the interoperability among smart-phones and PDAs devices with personal computers and servers.

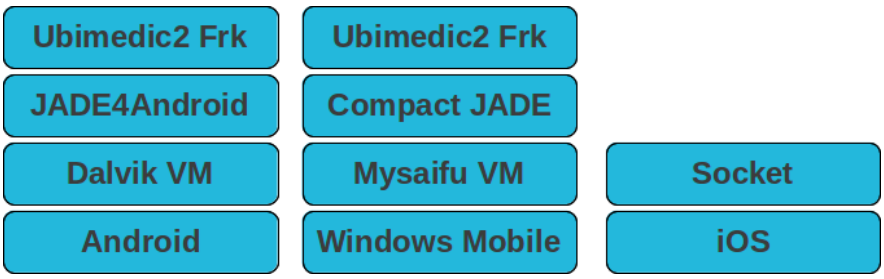

Figure 2.9: Smartphones and PDAs architecture

Since there is no such support for JADE in iOS, this gap has been filled by using sockets to exchange information between the mobile device with iOS and an Ubimedic2 agent. In particular, a MOA must have a socket always opened to receive new requests for communication and data. The requests for communication come from new devices that, when turned on, ask to establish a connection with it, and data come from the devices where a communication has been established.

The device must provide its IP address, the device type and the credentials. The socket port of the device can be fixed (the same to all devices, this decision must come in design time) or must be notified in the communication request. The MOA must create more sockets, one for each device connected to. This decision comes by the fact that continuous data (such as ECG trace) require always an opened channel between to hosts.

2.5.2 Framework

The **SuperAgent** class is the main class from all Ubimedic2 classes must inherit. It offers the main functionalities such as:

- *set/getName*: to set and to get the agent name;

- *set/getType*: to set and to get the agent type;
- *sendMessage*: to send messages to one or more agents;
- *receiveMessage*: to receive messages from agents;
- *searchByServiceName*: to search the agents that are registered with a given service name;
- *searchByServiceType*: to search the agents that are registered with a given service type;
- *registerService*: to registers new agent to the Directory Facilitator when they are created.

The **Activator** class extends the *SuperAgent* and represents an operative centre. It has one cyclic behaviour that retrieves the new requests for intervention and dispatches them to the available ambulances. It has one abstract method which requires to be implemented and determines what should be done if there are no available ambulances.

The **ServiceAgent (SA)** class extends the *SuperAgent* and represents the medical and hand-handle devices. It defines the agent names that allowed accessing the data. Since in this thesis policy issues have not been faced, predefined names are loaded by the service agent.

The **DeviceServiceAgent (DSA)** class extends the *ServiceAgent* class and represents medical devices. It has one cyclic behaviour divided into two steps. The first step checks for new messages coming from agents (called requesting agent in the following) that need access the data (see Figure 2.10) and sets the parameters that define if the device is in sending mode, which agent has requested the data and the sending type. The type can be *on-shot*, when only one message is sent to the requesting agent with the required data, or *continuous*, when data are sent with a predefined frequency to the agent till it sends a message requiring stopping the messages flow. At the second step, if the device is in sending mode, it sends the data to the requesting agent. If the requesting agent has required *on-shot* data, after sending them, DSA resets the request and does not send other messages until a new request has not been received, otherwise, it continues sending messages with the updated data.

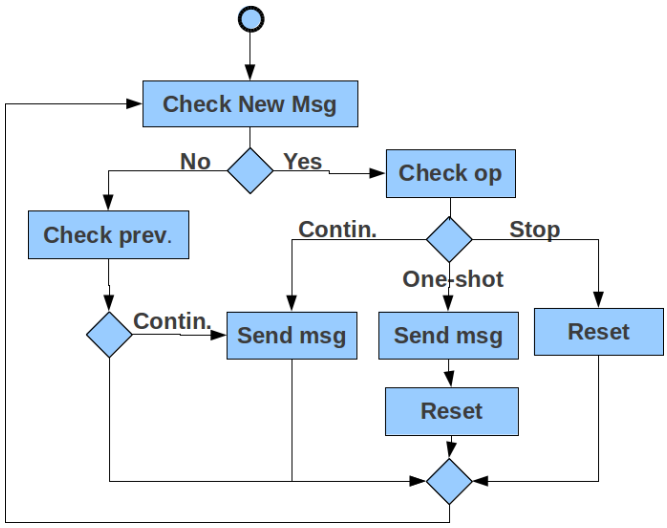

Figure 2.10: DSA life cycle

The **ClientServiceAgent (CSA)** class extends the *SuperAgent* and represents hand-handle devices used by medical staff. It has two cycle behaviours (see Figure 2.11). The first behaviour, continuously checks if there are new data or information to send to the MOA. The data messages represent the personal information or health parameters inserted by the medical staff. Meanwhile, the information message consists on the request of continuous/one-shot data collected from the DSA through the medical device. The second behaviour continuously checks for new messages received for the MOA and update the device data.

The **OperativeAgent (OA)** class extends the *SuperAgent* class and represents the operative units such as ambulances and hospitals. At the current implementation, this class retrieves from a preconfigured file the agent names of the different SA that the agent (instance of the class) can communicate with.

The **FixedOperativeAgent (FOA)** class extends the *OperativeAgent* class and represents the fixed units such as hospitals and first aid temporary camps. This class has only one behaviour (see Figure 2.12) which receives the requests from the MOAs to host a new patient, checks the availability for the pathology affecting the patient and responds to the request denying or accepting it.

The **MobileOperativeAgent (MOA)** class extends the *OperativeAgent* class and represents the mobile units such as ambulances and medical helicopters. This class im-

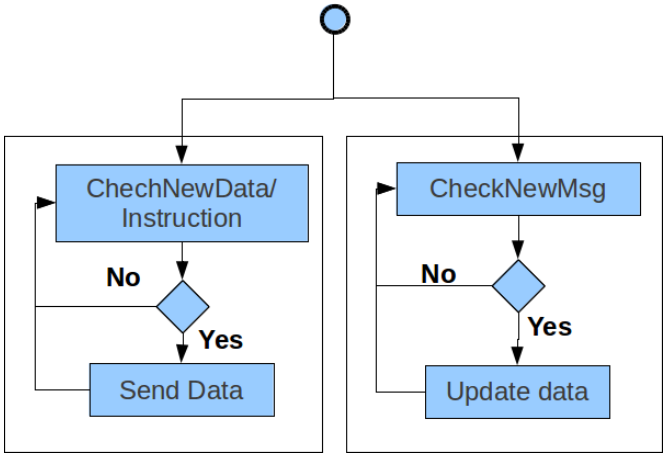

Figure 2.11: CSA life cycle

plements two main parallel behaviours (see Figure 2.13), the *State* behaviour and *Message Manager* behaviour. The former behaviour contains five finite state behaviours that correspond to the five mobile unit states and the latter parallel behaviour processes the asynchronous messages exchanged between the MOA and other agents.

In the state behaviour there are five states a mobile unit may have during a rescue operation such as reaching the place of accident and giving the first aid. Based on its real behaviour, five finite state behaviours are implemented in the MOA class:

- *Standby (5)* behaviour: the mobile unit is not performing any task and is ready to receive a new one;
- *ToPlace (1)* behaviour: the mobile unit, after receiving a new task, reaches the place where the accident occurred and this state represents the period of time the mobile unit takes to reach the place;
- *AtPlace (2)* behaviour: the mobile unit is at the place and the medical staff is giving the first aid;
- *ToHospital (3)* behaviour: the mobile unit is bringing the patient to the hospital;
- *AtHospital (4)* behaviour: the mobile unit is at the hospital and is delivering the patient to the triage medical staff;

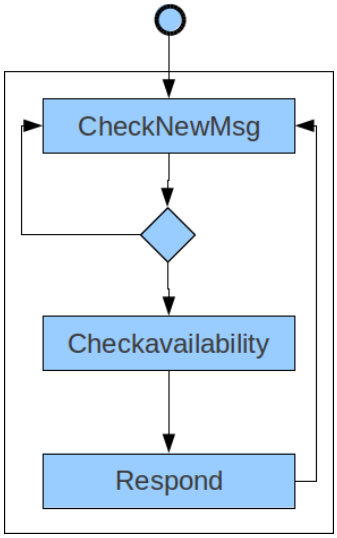

Figure 2.12: FOA life cycle

The code numbers are taken from the Italian First Aid department [1] standards. The behaviours follow cyclically each other, besides the state *AtPlace (2)* may move to *Standby (5)* if there is no patient to bring to the hospital (because it is not necessary or the patient himself refuses the transport) and the ambulance is free to perform any other task.

In Ubimedic2, MOA messages are divided into two categories: *synchronous* and *asynchronous*. A message is called *synchronous* when the receiver interrupts its process waiting for the message, meanwhile it is called *asynchronous* when the receiver is not waiting for a particular message and does not interrupt processing. In JADE there is not such distinction among messages. Messages may arrive at any time and is up to the agent to decide if waiting for any information or continues processing. Anyway, JADE defines message types such as *Reply* and *Information*. A *Reply* message, in this context, can be seen as a synchronous message and an *Information* message as an asynchronous one.

In the Ubimedic2 architecture, the distinction between *synchronous* and *asynchronous* was necessary to differentiate two data type, those coming from SAs that continuously provide health information about the patient and do not require any specific answer or action by the receiver (besides storing the data they hold that will be used if necessary)

[1] http://www.118italia.net/

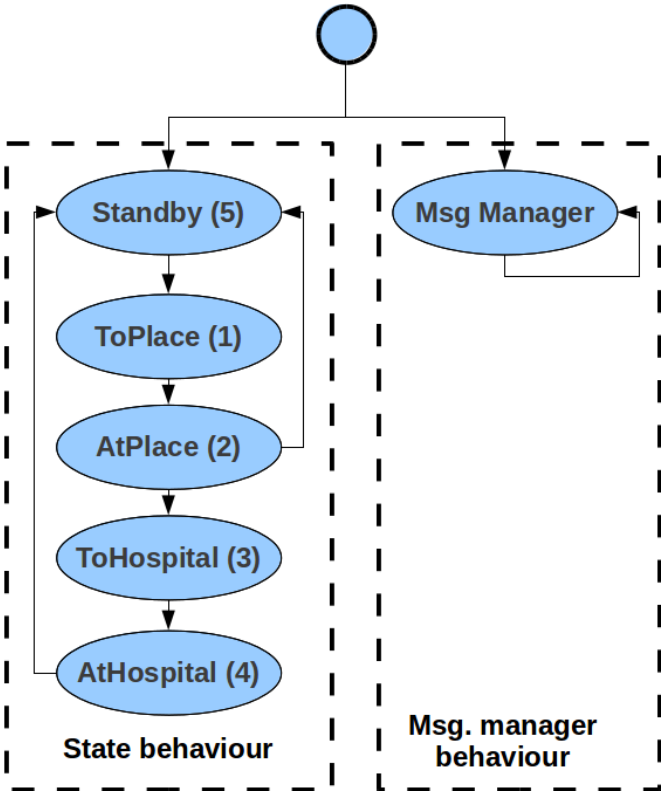

Figure 2.13: MOA life cycle

and those coming from OAs holding state information and require the particular attention of the agent, e.g. during intervention dispatch and patient dispatch. As the first category requires a process that continuously exchange information between a MOA and a SA, this need has been designed in parallel behaviour that continuously exchange these data. Other information is exchanged using the *SuperAgent* methods (previously introduced).

2.5.3 Data

Most of data are collected and stored by the MOAs. They collect information about the patient health and full name. In the following, the Entity Relational schema of the data managed by the MOA will be introduced. Besides the patient and intervention information, the MOA manages information about the hospitals and the wards it contains. This information is set up before each agent is activated and can change while it

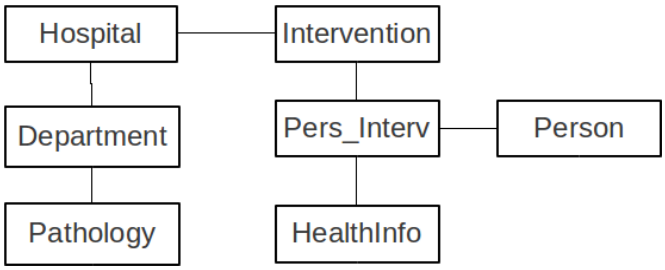

Figure 2.14: Reduced ER schema

is active. When a new temporary first aid camp is available, the Activator informs the agents about the new hospital, its location and the pathologies that can be treated by defining the wards it contains. In Figure 2.14 the reduced Entity Relational schema is shown. The `Intervention` table contains the data sent from the Activator to the MOA when a new task is assigned. The `Person` and `HealthInfo` tables are used to store information about the personal details and health information of the patients and the `Hospital` and `Department` tables contain information about the hospital and the ward they host. In the following, each table will be introduced describing each relational attribute.

Name	Type	Description
ID	longint	Table id (Primary Key)
NAME	varchar	Patient name
SURNAME	varchar	Patient surname
GENRE	char	Patient genre
BIRTHDATE	date	Patient date of birth
PERSONALCODE	varchar	Patient personal code

Table 2.1: Table PERSON

The `Person` table (see Table 2.1) contains the patient personal information such as name and surname. If this information is not know because the patient could not provide it or there is no identifying document.

The `Pathology` table (see Table 2.2) contains the pathology list codified by the health system. Such code will be used for data exchange.

The `Hospital` table (see Table 2.3) contains all the hospitals available in the territory where patients can be carried to receive furtherer treatments.

Name	Type	Description
COD	varchar	Pathology code (Primary key)
DES	varchar	Pathology description

Table 2.2: Table PATHOLOGY

Name	Type	Description
ID	longint	Table id (Primary Key)
NAME	varchar	Hospital name
GPS	varchar	Hospital GPS position

Table 2.3: Table HOSPITAL

Name	Type	Description
ID	longint	Table id (Primary Key)
NAME	varchar()	Department name/Ward name
PATHOLOGY_COD	varchar	Pathology code treated at the department/ward (FK to the PATHOLOGY table)
HOSPITAL_ID	longint	The hospital ID that hosts the department/word (FK to the HOSPITAL table)
MAX_NUM	int	The maximum number of patients that can be treated

Table 2.4: Table DEPARTMENT

The `Department` table (see Table 2.4) contains all the departments for each hospital and the maximum number of patients that can be treated.

Name	Type	Description
ID	longinteger	Table id (PK)
DATE	datetime	Date and time when the request was notified
ADDRESS	varchar	The address where the intervention was requested
PLACE_CODE	varchar	Describes the location type such as home, street or public place
PATHOLOGY_CODE	varchar	Describe the expected pathology effecting the patient/s
CODE	char	Describes the expected seriousness of the patient/s

Table 2.5: Table INTERVENTION

The `Intervention` table (see Table 2.5) contains the information about the interventions assigned to each MOA.

Name	Type	Description
INTERVENTION_ID	longinteger	The Intervention id (FK to the Intervention table)
PERSON_ID	longinteger	The patient id rescued in a intervention (FK to the Person table)

Table 2.6: Table PERSON_INTERVENTION

The `Person_Intervention` table (see Table 2.6) contains the relations between the Person table and Intervention table and defines the persons that have been rescued on a given intervention.

Name	Type	Description
PERSON_ID	longint	The patient id that is being treated (FK to the Person table)
INTERVENTION_ID	longint	The Intervention id (FK to the Intervention table)
DEVICE_TYPE	varchar	The device type that measure the health parameter
DEVICE_NAME	varchar	The device name/code that measure the health parameter
PARAMETER	varchar	The measured health parameter
VALUE_ASCII	varchar	The value of the health parameter if textual
VALUE_BYTE	byte	The value of the health parameter if saved in bytecode
TIME	datetime	The time when the health parameter has been measured

Table 2.7: Table HEALTH_INFO

The `Hearlth_info` table (see Table 2.7) contains all the health information collected for a given patient (defined by the PERSON table) rescued on a given intervention (defined by the INTERVENTION table). For each parameter detected, the time stamp is stored besides the value in order to analyse the evolution of the patient health condition. This information can be used for statistics too. For each record, only one of the two attributes VALUE_ASCII or VALUE_BYTE is populated based on how this data is represented. The necessity to store the device type and name comes with the necessity of tracing where the data come from and the necessity to give a priority about the correctness of the data collected. Some devices are more precise then others. For example, the hart-rate coming from the ECG device is more precise that the same measurement coming from the oximeter device that in particular conditions (such as cold hand) looses its precision.

2.5.4 User interfaces

The architecture introduced in Section 2.3 represents the framework of Ubimedic2 which classes implement the basic functionalities of the components they represent. Such classes can be extended in other classes in order to specialize their behaviour and adopt it to the real unit behaviour such as ambulances and medical cars. Therefore, an agent instance can be instantiated on each unit to better represents its peculiar characteristic.

In order to evaluate the feasibility of the Ubimedic2 approach, a set of interfaces has been implemented and the framework agent classes extended. The main framework classes have been extended implementing a set of policies and strategies. The devices they represent are simulated by the use of interfaces that will be described in the following.

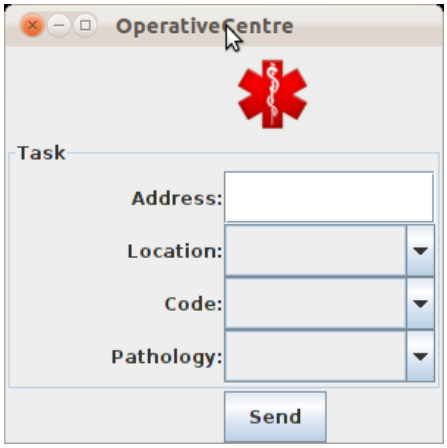

Figure 2.15: OperativeCentre GUI

The **OperativeCentre** is the class that extends the *Activator* class and represents the First Aid operative centre. It collects the new tasks and dispatches them to the MOAs. Its user interface has four input fields (see Figure 2.15): *Address* where the human operator of the operative centre inserts the address of the accident; *Location* which takes a value that represents type of the place such as Home (H), Street (S) and Public Building (P); *Code* that represents the expected seriousness of the patient health and can take three different values such as Green, Yellow and Red (when no information is available the Red code is assigned); and *Pathology* that represents the expected pathology affecting the patient.

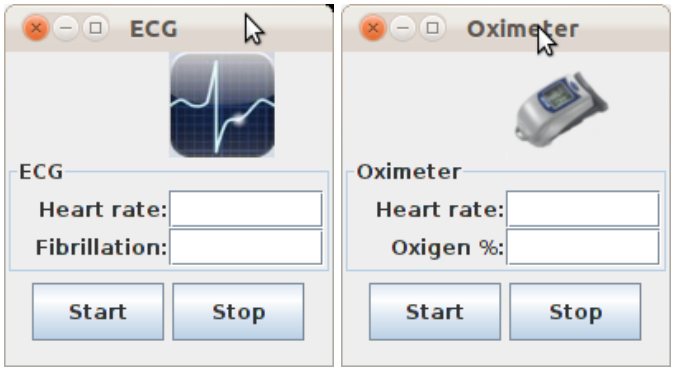

Figure 2.16: Ecg (left) and Oximeter (right) device interface.

The **Ecg** class extends the *DSA* class and represents an ECG medical device. This agent is provided with an interface containing two input fields, *Heart rate* and *Fibrillation* (see Figure 2.16, left). These two parameters will simulate the data retrieved from the medical device during experiments. Oximeter agent has a similar implementation (see Figure 2.16 right).

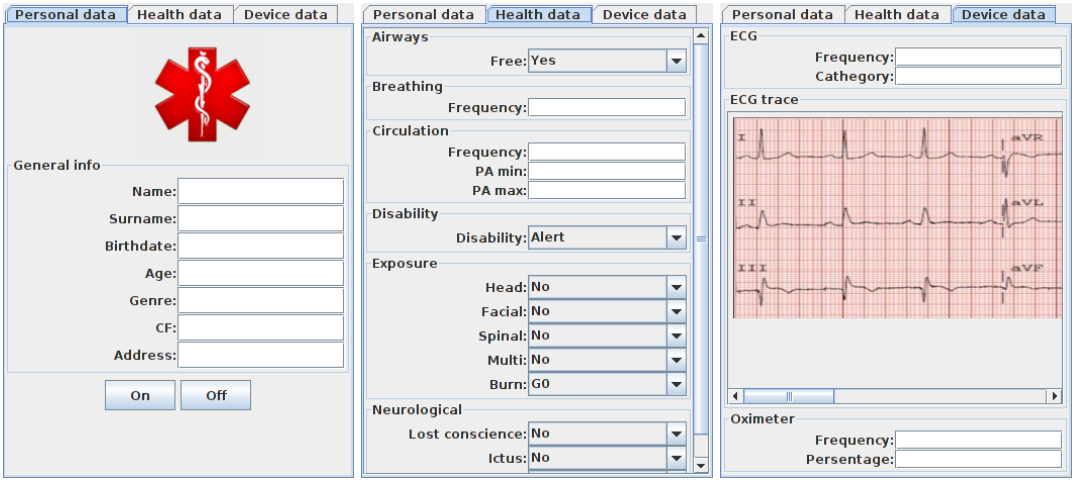

Figure 2.17: PDA user interface.

The **Pda** class extends the *CSA* class and represents a PDA device or a smart-phone. This device is used by medical staff to input information about the patient and visualize data from medical devices. The interface provides three screen tabs as shown in Figure 2.17. The first tab is used to input personal data such as patient name, surname and date of birth. The second tab is used to input health information. Data are structured

based on the IRC (Italian Resuscitation Council [1]) guidelines. The third tab is used to retrieve and display data from different medical devices (such as Ecg and Oximeter in the Figure 2.16).

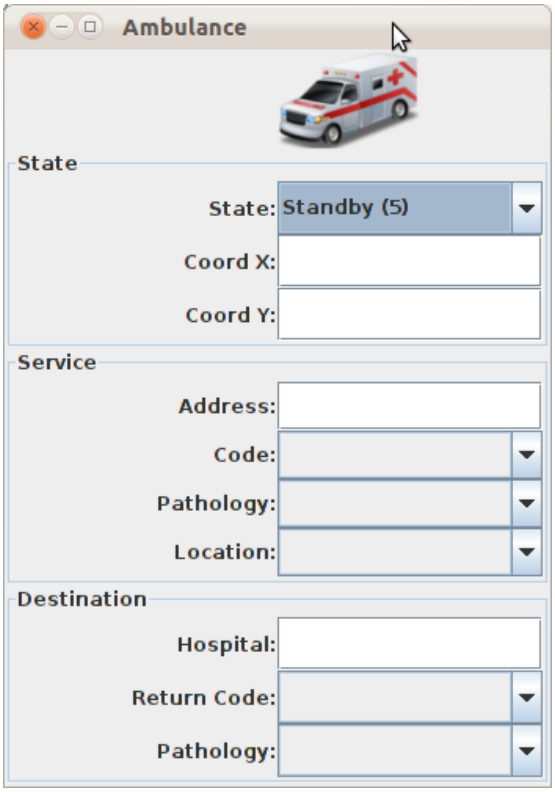

Figure 2.18: PDA GUI

The **Ambulance** class extends the *MOA* class and then its behaviour is based on the finite-state model with 5 predefined states (1 – reaching the place, 2 – on-site, 3 – reaching the hospital, 4 – at the hospital and 5 – standby). The passing from a state to another has been driven by a timer that simulates the time the ambulance takes to reach the place, to bring the patient to the hospital etc.

The interface, shown in Figure 2.18, is divided into three sections: 1) `State` which shows the status and the location of the ambulance, 2) `Service` which shows the task information sent from the OperativeCentre and 3) `Destination` which shows the hospital where the patient is going to be carried, the main pathology and the seriousness of it.

[1] http://www.ircouncil.it/

For experimental reasons, a simple implementation of the `getHospital()` and the `getPathologyAndGravity()` methods has been done, following the IRC guidelines, since the goal of this project is not to make health diagnosis. These two functions may be extended and improved with the help of medical staff without any change in the architecture.

2.5.5 Coordination workflow

Once all the agents of the architecture have been defined, the further step is to describe the interactions among these agents following the coordination approach introduces in Ubimedic2. For a better understanding of these interactions, the scenario introduced in the Motivating Example section will be described considering the Figure 2.19.

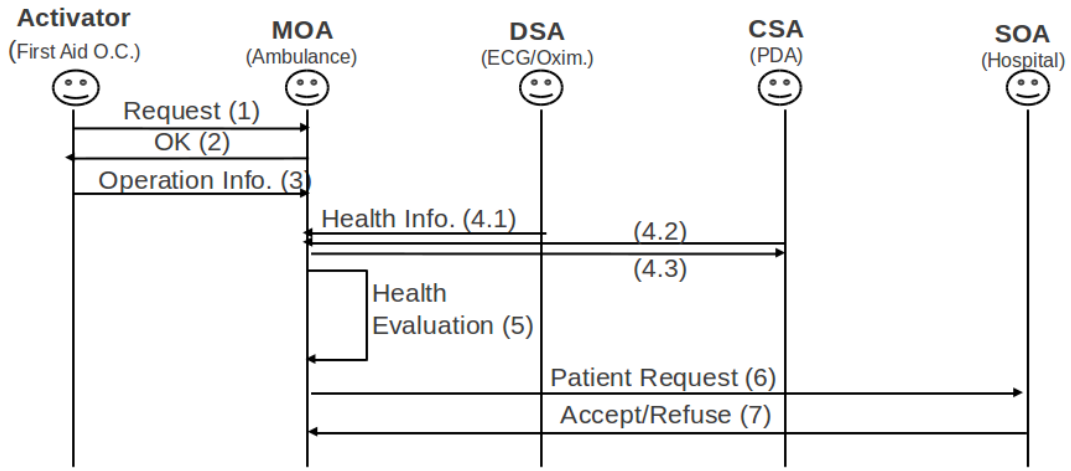

Figure 2.19: Coordination activity

Someone calls the emergency number after an accident has occurred. The human operator of the First Aid department, after collecting the necessary information, inserts a new task for intervention. The *OC* agent (an instance of the OperativeCentre class), after receiving the new task, searches for the first closest ambulance and sends it a request (step 1) asking if the ambulance is available to receive a new task. If the Ambulance agent (an instance of the Ambulance class) is not available because is carrying out another task, it responds declining the request. Otherwise, it accepts the request sending a positive message to the *OC* agent. If the *Ambulance* agent does not accept

the new task, the *OC* agent sends another message to another *Ambulance* agent representing the next closest ambulance till it finds an available one. If no ambulances are available as all respective agents decline the request, the *OC* agent will decide if to activate other units (staff for a new ambulance) or try again when one of the available ambulances is free. When an *Ambulance* agent accepts the task (step 2), the *OC* agent sends the necessary information about the place, the main pathology and its seriousness (step 3).

Once at the accident place, the medical staff collects the personal and health information of the patient using electronic medical devices or hand-handle devices. The *Ambulance* agent collects information from the *Ecg* and the *Oximeter* agents (instances respectively of the Ecg and Oximeter classes) (step 4.1), from the *Pda* agent (an instance of the Pda class) (step 4.2) and, if requested, sends the information collected from the devices to the *Pda* agent (step 4.3) if the medical staff needs to visualize these data on her hand-handle device. The *Ambulance* agent, with a pre-configured frequency, evaluates the patient health condition and the most proper hospital to bring the patient (step 5). The strategies used to decide which the appropriate hospital to host the patient is can be different, based on the territory where they are applied.

At the current implementation state the following strategy has been adopted. If the seriousness degree is represented by the red code (that means the patient needs an immediate hospitalization) the closest hospital is chosen. Moreover, this choice comes with the assumption that every hospital must provide the necessary equipment to treat the most critical health conditions. Otherwise, the Ambulance agent chooses the hospital with the largest number of hosting patient (calculated based on the free beds and the medical staff present at the hospital).

After deciding the most proper hospital, the Ambulance agent asks the Hospital agent (instance of the Hospital class) if there are free resources to host the patient for long treatment (step 6). If the Hospital agent does not accept the request (step 7), the Ambulance agent sends another request to another Hospital agent able to treat the previously defined patient pathology till it finds an available one. Otherwise, the medical staff brings the patient to the one of hospitals depending on the strategies implemented in the Ambulance class.

The evaluation process continues even on the way to the hospital. In fact, the Ambulance agent, after a new evaluation, can take another decision about the patient

pathology and its seriousness, and can decide that the most proper hospital is not the one previously chosen: then, steps 6 and 7 are processed asking the new Hospital agent if the patient can be hosted for treatment.

2.6 An Android application

In the medical field, with the help of information systems, data are becoming digital and the electronic patient sheet is not just bare idea any more. Smart-phones and tablets offer the appropriate technology to access these data remotely and at any time. In fact, smart-phones, PDAs and tablets have become more and more accessible due to their cost reduction. The performance growth, in terms of memory size and processing speed, makes them more and more suitable for many tasks. The access to the Web via Wi-Fi or GSM connection makes them tiny and useful portable computers able to access many services and to exchange data with other devices. JADE developers, aware of the usefulness of such portable technology, have extended the JADE platform in order to include these technologies that do not support the traditional Java byte code.

The Android operating system is an open source system developed by the Open Source Alliance led by Google corporation. It is a general purpose OS (in contrast to iOS, Symbian and BlackBerry) and is supported by many smart-phones device with different technologies and platforms. The wide spread and the open technology has brought the JADE developers to give a large support for the Android OS through an add-on called Jade4Android along with the appropriate tutorials.

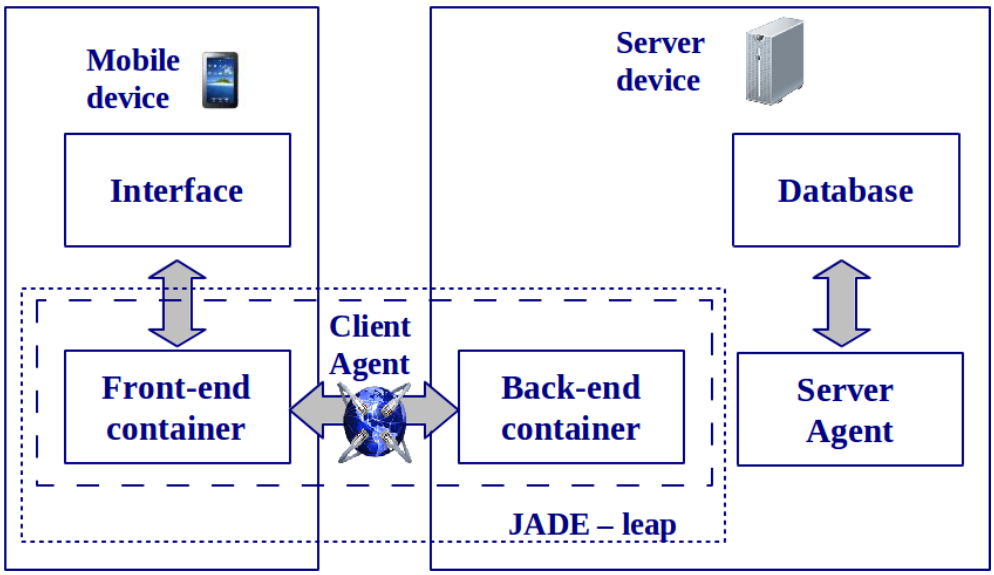

Figure 2.20: Split mode for JADE

To deal with the constraint of the limited computation and the reduced memory capacity that smart-phones may have, JADE offers two different operating modes for executing an agent on smart-phones: *stand-alone* and *split* mode. The first executes the agent entirely at the device making use of the device processor and memory. The second splits the agent life cycle into two different nodes: the front-end executed on the smart device and the back-end executed on another node such as personal computer. It allows the programmer to transfer all the hard computation to a much more performing node and leave to the smart device only the task to collect or to visualize the data and interact with the user. All this is transparent to the programmer who manages only one agent the life cycle. The front-end, when activated, connects to the predefined node and interacts with its respective back-end container. The JADE4Android architecture for this case is described in Figure 2.20.

As introduced in Section 2.5.1, an Andriod application has been developed using the Ubimedic2 framework. The custom user interface is similar to the one presented in Figure 2.17. The application has been developed over the Android 2.3 platform version and the interface optimized for the 8" display monitor. The test has been performed on a Galaxy Tab device.

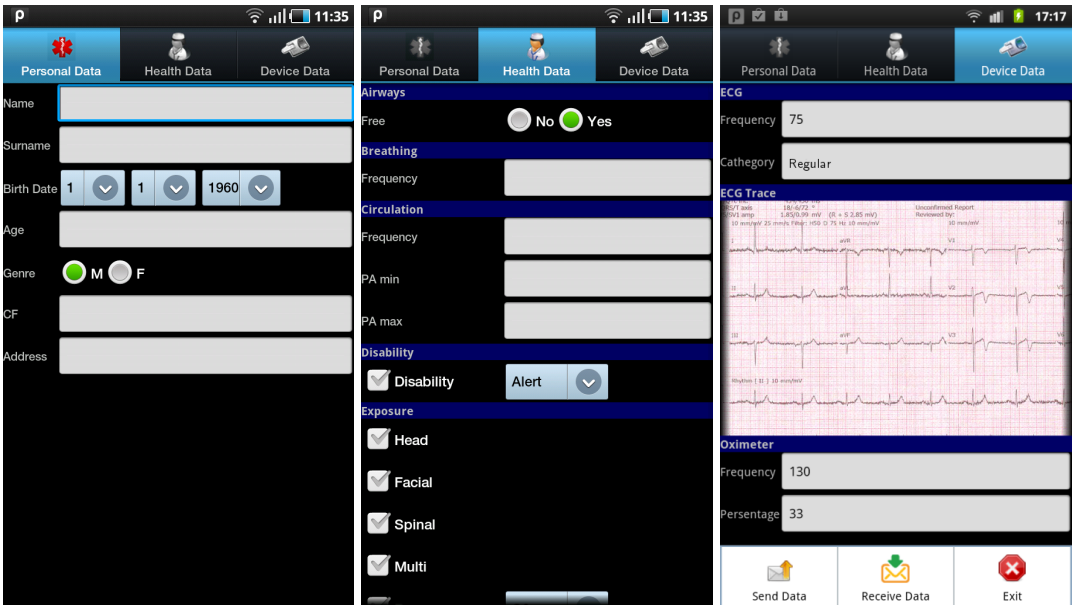

Figure 2.21: Android application GUI.

As shown in Figure 2.21, the application GUI has three tabs, which enable the medical staff to insert the patient personal information such as name, surname and birth date (first tab), health information based on the IRC protocols (second tab) and to visualize the data collected from medical devices (third tab), which has been simulated for test reasons. The "split mode" has been used to connect the application to a server node (consisting of a notebook Mac OSX 10.7 Lion).

One of the characteristics of the application on smart devices is the continuous exchange of data with the server and in particular the ECG trace, which requires higher performance rather than textual messages. For this reason, performance tests have been done, in order to evaluate the maximum number of messages that can be sent from and to the device and the limitation on message dimension. The first test evaluates the number of messages that can be sent over time and the second test evaluates the time it takes to send messages of different dimensions.

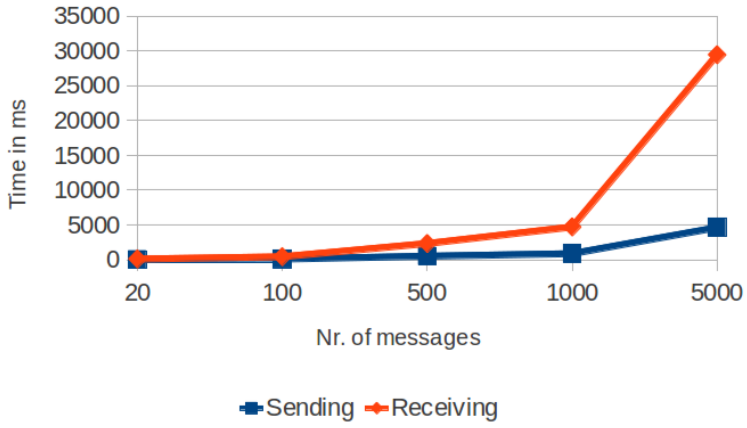

Figure 2.22: Time evaluation over number of messages

The first test evaluates the number of messages that can be sent over time. For this experiment a set of messages has been sequentially sent from the device to the server and from the server to the device and the overall time has been measured. In Figure 2.22 by the results of the first test, it can be observed that the time the device takes to send the messages is almost linear and it takes less than 5 second to send 5000 messages. While receiving, it takes much more time if the number of sent messages goes

over 1000 messages. This limitation is quite acceptable as the number of messages expected to be sent by the server with the updated data of the patient is much smaller.

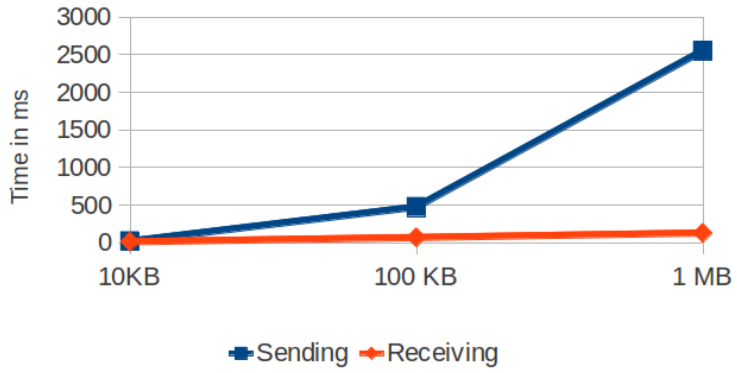

Figure 2.23: Time evaluation over message dimension

The second test evaluates the time it takes to send messages of different dimensions. This test aims to analyse the possibility of sending images representing the ECG trace or other visual information mainly from the server to a smart device. JADE implements two types of message content to be exchanged through agents: text and byte code. The method that assigns the textual content, takes a "String" data type in input. For byte code content, only serializable object can be sent. This requires the images to be processed before they can be sent through a JADE message. In Figure 2.23 it can be observed that the device is able to receive message up to 1MB within an average of 128 ms. while sending, the serialization process influences much more the device performance.

It can be observed that in both experiments, the performance of the device is satisfying as the number of messages that can be sent over time and the image exchange from the server to the device fulfil the project requirements. This makes Android and JADE a good technological solution in the integration of smart devices such as smartphones and tablets.

A similar work has been done for devices using iOS. An iOS application has been developed referring to the interface presented in Figure 2.17. Unfortunately, no test performance test has been done duo to Apple constraints on application installation on devices.

2.7 PIM

In the architecture introduced above, agents share information by exchanging messages with each other. Some update information is sent in broadcast such as the presence of a new hospital (which informs all the MOA to consider it during the patient dispatching process), alerts raised by the ambulances about traffic jump or unavailable roads (considerations that an ambulance must take into account during path calculation to the hospital), etc. Another issue in the above architecture is the use of a distributed algorithm in the patient dispatching process. In order to balance the hospital load, the number of patients must be proportional to the maximum load the hospital can hold. A distributed algorithm would require a considerable number of messages to be exchanged. Each node (MOA) must have information about the decision of every other node working not only in that intervention but also in others.

In the literature, there are many works on distributed algorithms that require a master node where a single node assumes the task to coordinate all the others. This is a centralized approach applied in a distributed system. To remain in the distributed approach, the PIM technology has been taken into account to be integrated into the Ubimedic2 architecture.

The PIM (Process Integrated Mechanism) [50] technology has been implemented by the Institute for Human and Machine Cognition (Florida, USA) and was born with the purpose of coordinating a set of robots. The idea that stays behind PIM is the use of a single process called Coordination Process (CP), which moves from one node to another exploiting the strong code mobility. It inverts the idea of multi-threading where more processes use one single computational unit, giving the impression that each process has the exclusive of the processor. Instead, PIM uses a single process that runs in many processing units cyclically moving from one node to another holding the whole state of the process such as data and program counter.

This technology does not aim to substitute the agent architecture as one of the main requirements of the MOAs is the ability to remain autonomous even during network problem and lack of communication. The CP must influence the decision of the agents about the patient distribution over the available hospitals, inform them about new hospitals or new first aid camps available, about roads and map paths to avoid due to traffic and other useful information that would require broadcast messages.

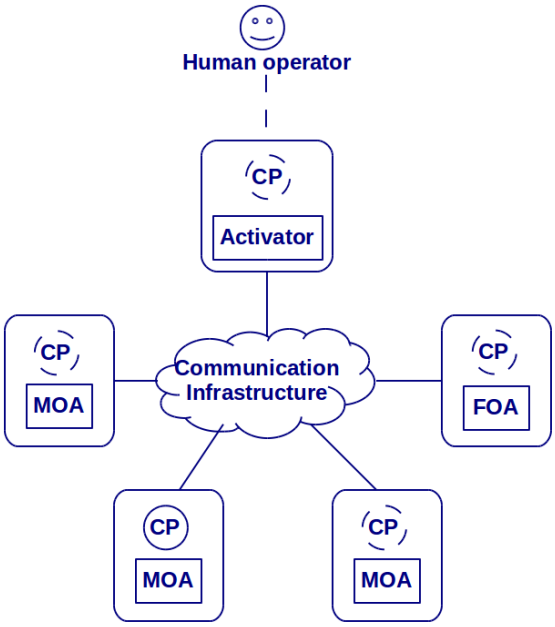

Figure 2.24: PIM integration

Each node must host the PIM platform and the same controller process. The process must be started from one node (in this case the depicted node will be the Activator) that is called the "bootstrap" node. For each new ambulance added to the intervention, the CP will update its node list appending the new node to the array of nodes (this means that the new node will be last to be reached by the CP).

The nodes interesting the CP are: the Activator, the MOAs (ambulances and other medical vehicles) and FOAs (hospitals and first aid camps). The SAs are not included in the PIM platform as these agents do not take part in the decision process. In Figure 2.24 is shown how the CP moves from one node to another, executing only in a single node per time (CP with the continuous cycle line) and all other nodes wait for their turn (CP with the dashed cycle line).

The communication infrastructure is represented as a cloud as it can be compound by different technologies as mentioned in Section 2.3.2 and these technologies are not being treated in details as Ubimedic2 focuses only on the coordination activity among different nodes in a P2P system. The test and experiments done so far uses the Wi-Fi communication technology as the most feasible to be set up in a laboratory area.

The CP is not an alternative to the agent approach but a support. One of the system

requirements is that agents must be always able to perform their decision and in absence of connection, the node would be unreachable by the CP compromising the agent activity. Besides complexity, distributed algorithms require a high interaction among distributed nodes and this would bring to a large number of messages exchanged by agents. To face such issue, a single process that manages all the nodes would be a feasible solution. The node excluded due to connection problems will still be self-sufficient as the agent running on it is independent but, will the decisions will not be improved by the overall overview of the controller process.

Conclusion

The work carried out during the three years of the Ph.D. activity had the aim to introduce intelligent approaches to support users during their activity. In particular, the two projects this work covered brought to two different approaches applied in keyword based search engines and in disaster management.

The first project follows a semantic approach to build keyword-based search engines. In contrast to traditional keyword search techniques that require access to the actual data stored in the database in order to build indexes, a novel approach, called Keymantic, has been proposed. This approach can be used in a wide range of applications, including databases on the web and information integration systems. As building and maintaining specialized indexes are not feasible options, an alternative solution has been introduced. Keymantic makes use of intensional knowledge. It exploits the semantics of the keywords provided by the user and builds the SQL queries to access the data. For the challenging task of interpreting the keyword queries and to rank the solutions from the most feasible to the less one, an extended version of the Hungarian algorithm has been used to map each keyword to the database terms.

The experiments introduced in Section 1.9 bring to an average recall (percentage of queries with answer) of 82% and an average precision of 71%. The average values derive from the different results achieved making use of different databases and of the different set of queries. In any case, the process time, even in critical cases, is in the order of seconds.

For future work, a real time analysis can be done to detect the meaning of keywords not mapped to any of the database terms. The Yahoo! similarity has been tested but the contribution given to map the keywords was not

satisfying. Different dictionaries may help increasing the system precision with the price of a decrease of the system performance in term of time consuming.

The second project follows instead the agent approach in order to support users in managing territorial emergencies. It is based on the Multi-Agent Systems (MAS) technology that offers a high flexibility in adapting in different scenarios. Each rescue operation is a complex process that requires high degree of coordination and cooperation among different entities in a short time. The weak points of the current approach adopted by the first aid operative centre are: communications are done via phone or radio, no digital information collection is done and there is no way to do statistics analysis, and communications are not error free.

To overcome these weaknesses we have designed the Ubimedic2 architecture as an evolution of the Ubimedic one. In particular, the Ubimedic2 architecture supports the human operators in their activities, offers a decision making support, and real-time data collection and communication. It is fully-distributed and scalable. Ubimedic2 is written in Java on JADE, is platform independent and is supported by most of the commercial digital devices for communication such as computers, thin clients and smartphones. The use of Ubimedi2 system reduces time and offers data integrity during propagation. The decision making process made by agents is possible due to the well defined operation roles which the human operators must follow. The operative centre is no more a bottleneck in communication. The time evolution of the events is in the order of minutes, so computation time of distributed agents is quite performing in our scenario.

To evaluate the usability and the feasibility of the Ubimedic2 a set of simulations has been performed. They are reliable with respect to an implementation with real devices as the agent behaviour does not change. Instead of using the custom user interface, the agents will collect the information from the digital devices and the management of these data remains the same we have already observed. In the testing, we have focussed in data represented using ASCII format such as text and numbers but JADE

handles messages with byte content so device agents are able to exchange images, videos or other byte data such as the ECG trace.

In order to improve the dispatching of the tasks to the ambulances and of the patients to the hospitals, the PIM technology has been adopted. This technology puts together the benefits of a centralized algorithm and the distributed approach exploiting the code mobility among nodes.

In conclusion, in different fields, activities processed by humans can be simplified offering specific user supports. The results achieved in these two different projects demonstrate that the use of high-level solutions such as semantic search engines and agent-based systems improves those processes that require artificial intelligence. The use of semantic search engines and agent-based system is not in its early stage but more efforts must be done in order to improve their efficiency and their use to face different technological issues.

Publication

- Sonia Bergamaschi, Elton Domnori, Francesco Guerra, Silvia Rota, Raquel Trillo Lado, and Yannis Velegrakis,"*Understanding the Semantics of Keyword Queries on Relational DataWithout Accessing the Instance in "Semantic Search Over the Web"*", pg 133-162, Book author: Roberto de Virgilio, Francesco Guerra, Yannis Velegrakis, Publisher Springer (IN PRESS);
- Elton Domnori, Giacomo Cabri, Letizia Leonardi, "*Multi-Agent approach for Disaster Management*", **SEDM** 2011, Barcelona (ES);
- Elton Domnori, Giacomo Cabri, Letizia Leonardi, "*A multi-agent approach for territorial emergency management*", **WAO 2011**: p74-80, Rende (IT);
- Sonia Bergamaschi, Elton Domnori, Francesco Guerra, Raquel Trillo Lado, Yannis Velegrakis: "*Keyword-based Search in Data Integration Systems*", **SEBD 2011**, Maratea (IT);
- Elton Domnori, Giacomo Cabri, Letizia Leonardi, "*Designing and implementing intelligent agents for e-health*", **EIDWT 2011**, Tirana (AL);
- Elton Domnori, Giacomo Cabri, Letizia Leonardi, "*Ubimedic2: An Agent-Based Approach in Territorial Emergence Management*", **PervasiveHealth 2011**: p176-183, Dublin (IR);
- Elton Domnori, Giacomo Cabri, Letizia Leonardi, "*Managing Territorial Emergences with Ubimedic2*", **PervasiveHealth 2011 - demo session**: p200-201, Dublin (IR);

- Giacomo Cabri, Elton Domnori, Davide Orlandini, "*Implementing Agent Interoperability Between Language-Heterogeneous Platforms*, **WETICE 2011**, Paris (FR);
- Sonia Bergamaschi, Elton Domnori, Francesco Guerra, Raquel Trillo Lado, Yannis Velegrakis: "*Keyword Querying over Relational Databases: an Intensional Knowledge-based Approach*", **SIGMOD Conference 2011**: 565-576, Athens (GR);
- Elton Domnori, Giacomo Cabri, Letizia Leonardi "*Coordination Issues in an Agent-Based Approach for Territorial Emergence Management*", **AINA Workshops 2011**: 184-189, Singapore (SG);
- Sonia Bergamaschi, Elton Domnori, Francesco Guerra, Mirko Orsini, Raquel Trillo Lado, Yannis Velegrakis: "*Keymantic: Semantic Keyword-based Searching in Data Integration Systems*", **PVLDB 3(2)**: 1637-1640 (2010), Singapore (SG);

Appendix A

A reasoned state of the art about Keyword based search engines

A.1 Introduction

Querying structured data sources addresses several critical issues, especially in large and complex sources that may not easily be known and managed by users. The query formulation needs the knowledge of the database contents, i.e. which data are stored in the data source and how they are represented. Let us consider, for example, relational databases, where expressing a query means to be able to select the right tables and attributes of a database, and to specify proper constraints on attributes. Such a process implies overcoming some critical tasks concerning structural and lexical aspects:

1. the user selects the tables and attributes of interest on the basis of their names, which may be misleading or not meaningful (lexical aspect);

2. the user expresses constraints on attribute without having accurate knowledge of the domain. Thus, s/he may define too selective or, vice-versa, too broad or illegal selection clauses (lexical aspect);

3. the user does not know the relationships between the tables and, consequently, it is hard to pose multi-table queries (structural and lexical aspect).

Therefore, there is a direct connection between the user's ability to express queries and the knowledge of what and how the data are stored. If the user knows the database

contents, the query process requires only the translation of the user's search criteria into a proper formalism. On the contrary, selective queries require an analysis of the data source, i.e. a complex and time consuming task, for users that do not know the database contents. This issue is very significant when you deal with real applications, where data structures and instances are heterogeneous, since they come from different and unknown sources. The same issue has to be addressed in querying an integrated data source, i.e. the unified view that results from the application of an integration methodology to a set of data sources [51]. In these cases, the same real world objects are represented in different sources with different data schemata, names of attributes and domains of values, and, consequently, synonym, polysemic, broader terms have to be taken into account in query formulations. The knowledge of the data collected in the "integrated" view is mandatory for expressing selective queries.

The research community has developed techniques for querying structured data sources that, from the user's perspective, can be roughly grouped into two categories:

1. query engines where the structures of the sources allow the users formulating selective queries by means of a specific query language;

2. keyword-based search engines, which exploit information retrieval techniques for selecting the instances closer to the terms provided by the users.

Query engines allow users to express complex queries with selection clauses defining constraints on the results. On the other hand, a user has to know the structure of the sources (i.e. names of the tables, the names and domains of attributes, the relationships between the tables) and a query language for writing effective queries. The research community has been involved in developing tools for supporting users in writing queries (according to the query by example approach [52]) and for visualizing data sources structures (see [53] for a survey).

Keyword-based search engines are more intuitive for the user, but they support less selective queries, since they detect instances in data sources that satisfy specified keywords. Typically, such tools integrate techniques from information retrieval and database systems to compute results of a user's query. This is a new challenging research issue, that only recently has been addressing.

The techniques actually implemented for keyword based searching on data sources may be divided in two categories: the first group includes systems that represent instances in the database with graphs or other data structures and apply to these representations information retrieval techniques and the second group that contains techniques for transforming keywords in queries to be executed by the database query engine. According to these categories, the report is divided in two parts: in section A.2 the main keyword based search engines developed in literature are reviewed on the basis of a common evaluation criteria.

A.2 Keyword based search engines

In the last few years, the research community developed several keyword based search engines. In this section, we evaluate these systems according to the following perspectives:

1. Representation model of the database: this perspective aims at evaluating how the data source is represented for further analysis. The following aspects are taken into account:
   - Adopted model for representing structure, instances, or both with a unique model;
   - Metadata taken into account (primary keys, foreign keys, data-types, regular expressions, other syntactic metadata);
   - Semantic enrichment of the model by means of external ontologies / taxonomies / knowledge bases;
2. User's input: by means of this criterium we evaluate how it is possible for a user to indicate the information he is searching for. The following aspects are taken into account:
   - query language: keywords, natural language, controlled terms (from an ontology / thesaurus), graphical interface, constraint definitions, ...;
   - querying metadata: possibility of searching for metadata;
   - supported operators: boolean, logical and arithmetical operators.

3. User's input analysis: by means of this perspective we evaluate the techniques (if any) exploited for analyzing the user's input. In particular:
    - disambiguation: which techniques are implemented for understanding the correct meanings of the terms used in the queries;
    - input enrichment and transformation, i.e. techniques for transforming the user's input into a similar one to obtain better results;
    - relaxing constraints, i.e. techniques for transforming the user's input into a less selective one to obtain more results.
4. Search algorithm: this perspective analyzes the techniques exploited for matching the user's input with the database. The techniques may takes into account:
    - syntax/structure: we evaluate if the way in which the information is given is relevant for the process;
    - semantics: we evaluate how the meaning associated to terms is relevant for the process;
    - ranking techniques: the technique exploited for setting a relevance to the data retrieved.
5. User feedback: we evaluate how the user obtains results and the interaction between tool and user. In particular:
    - interaction with the user in the process: we evaluate if the searching process is automatic, of if it requires user's feedbacks;
    - data visualization, i.e. which data are shown and in which way;
    - data cleaning, i.e. the technique implemented for avoiding duplicated answers.
6. Performance evaluations (technique and results): this perspective analyzes how the performances of the search engine are evaluated (in terms of datasets and indicators) and the obtained results:
    - datasets, i.e. data and queries used for evaluating the search engine

- indicators, i.e. the variables that are used for testing the results
- obtained results, i.e. some relevant evaluation of the prototype

7. Implementation: by means of this perspective we want to evaluate if the system is really implemented and available for testing. In particular:
   - website / tool available for download
   - fixed dataset / set of query / free evaluation
8. Interesting literature: the most relevant papers about the described systems are summarized.

A.2.1 DBXplorer

DBXplorer (DataBase eXplorer) is a system that enables keyword-based search in relational database. It as a full, efficient and scalable system which offers the possibility to find structured data in database. Given a set of keywords, matching rows may need to be obtained by joining several tables on the fly. DBXplorer has been implemented using a commercial relational database and web server and allows users to interact via front-end browser.

[Representation model]

The system has two core components: *publish* and *search*. The *publish* component is a preprocessing step that prepares the database for keyword search by building a symbol table that helps maping a keyword to a database structure (tables, columns and/or rows). There are two kinds of granularity for data representation: *column granularity* which associates each data with the table name and the attribute name where it is stored and *row granularity* that besides the table and attribute name associates the row id. The symbol table is populated with all the data that can be item of search. The *search* component aims to find if the keywords are present in the database by looking up the symbol table and build the SQL statement to retrieve the data and finally rank them before sending to the output.

[User input]

The user input has the following characteristics: *a)* The DBXplorer supports only exact keyword search. This limit is due to the performance of accessing the indexed symbol table. By using the B+ tree index search like "mi%soft%" can be done efficiently but search like name = "%micro" requires a sequential access and no index can be used. *b)* No metadata are used to describe the data. Only location information is present. *c)* The search is purely based on keywords and has no additional operators. *d)* Conjunctive query (AND semantic) only is supported.

[Keyword analysis]

As only exact keyword search is supported there is no analysis over the keywords. Proximity or query relaxation are not considered.

[Search algorithm]

The search algorithm is divided into three steps: search, build query and retrieve & rank.

Step 1: The symbol table is looked up to identify the set of tables, attributes (and eventually rows) of the database that contain the query keywords.

Step 2: These set of tables are called matched tables and it is assumed that contains all the keywords. If we view the database schema as an undirected graph G, a solution may be represent by a subgrapg such that: (a) the leaf nodes belong to the matched tables and (b) together, the all the leaf nodes contain all the keywords of the query. In this step all subgraph are discovered [54, 55] and enumerated [56].

Step 3: Finally, for every sub-tree a SQL statement is constructed and executed. The selected rows are retrieved and ranked before sent to the output. The approach used in this search engine is to rank the rows by the number of joins the query is involved. This approach in arbitrary and is based on the idea that the documents in which keywords occur close to one another are ranked higher.

[User feedback]

The process is automatic and no user feedback is required. The final output is visualized through a web browser. If there is any result the system returns the list of the databases and the user selects a specific database to explore. In the next step the system return the pairs table-attribute where the keywords occurs and finally the user choose one of the join trees to explore.

[Performance evaluation]

The performance evaluations are made over synthetic (TPC-Hx) and real database (USR - employee address book, ML - mailing list and KB - articles and help manuals). The performance are evaluated for system table granularity, publishing time, search performance and scalability. For more information see [12]. Some critical evaluation are the follows:
a) Pub-Col has superior performance when keywords are not very selective.
b) Pub-Col scales linearly with data size and is independent of data distribution both in publishing time and in system table size.
c) Search performance scales with data size and numbers of search keywords.

[Implementation]

The system is property of the Microsoft company and is not published.

[Related literature]

[12] is a full description of the system DBXplorer. A detailed description of the architecture, data representation, search algorithm and performance experiments.
[54, 55] show algorithms for tree discovery from a graph.
[56] is an algorithm for enumerating trees of a directed graph.

A.2.2 Discover

DISCOVER [11] is a keyword search engine over relational database. The challenges faced by this system are: a) formalize keyword search on relational database and provide intuitive semantic; b) present an efficient candidate network generation algorithm which creates a complete and non-redundant set of candidates networks; c) propose a cost model; and d) present a detail experimental evaluation.

[Representation model]

The DISCOVER system proceeds in three steps: 1) generates all candidate networks of relation, that is, join expression of tables that contain all the keywords given by the user (AND semantic); 2) builds plan for efficient evaluation of the set of candidate networks exploiting the opportunities to reuse common subexpressions of the candidate networks; 3) retrieves and ranks data in order to the number of joins. The DISCOVER system has a complex architecture made of five modules: master index,

tuple set post-processor, candidate network generator, plan generator, plan executer which we'll discus later.

[User input]

The user inputs only exact keywords and no additional operators are supported. The keywords must specify only words the user expects to find in the data. Any possible characterization of the data can only lead to wrong data discovery. For example, if we are looking for car model "Hammer H2" and the set of keywords is "car Hammer H2" the system can't find any solution because the keyword "car" does not appear in any table of the schema.

[Keyword analysis]

Supposing that only exact keywords are inserted by the user, there is no analyze on the keywords. Proximity or query relaxation are not considered.

[Search algorithm]

Consider the set of keywords K = $\{ k_i, \ldots, k_n \}$ given by the user.
The *Master Index* outputs a set of tuple S_i for i = 1..n containing all the tables where the keyword k_i occurs. The master index has been implemented by building full text indexes on single attributes of relation.
The *Tuple Set Post-Processor* takes the basic tuple sets S_i and produce tuple sets S^K that contains all the keywords K.
The non-empty tuple sets S_i with the schema graph are passed to the *Candidate Network Generator*. A *candidate network* is subgraph of the schema G that contains of: a) leaf node tables that contain all the keywords and b) in-between node tables that are necessary to join the leaf node tables.
The candidate networks are passed to the *Plan Generator* which optimizes the evaluation of the candidates.
The *Execution Plan* builds the query statements and retrieve the data. It returns first the data obtained with the smallest number of joins.

[User feedback]

There is no user feedback.

[Implementation]

For the implementation of Discover system TPC-H [1] database has been used. The Master Index has been implemented using Oracle9i interMedia Text extension [2]. The Discover has been implemented in JAVA and connects to DBMS through JDBC. The system is not available for download or on-line use.

[Performance evaluation]

Experiments has been done over the Discover system to observe the performance. Some of the performance experiments undertaken are:
a) Measure the efficiency of the candidate network generator.
b) Compare the seedup in runtime performance for generating and executing the execution plan using the greedy and the optimal algorithm compared to the naive method where no intermediate result are build.
c) compare the overall execution time of DISCOVER for some typical keywords query to the naive method and the optimal method.

Some consideration over these experiments are:
a) The quality of the plans produced by the greedy algorithm are very close to the quality of the plans produced by the optimal.
b) The optimal method is always worse than the naive due to the greater time overhead in discovering the optimal execution plan.
c) The time to build the tuple set takes form 2 to 4 seconds and can be reduced by using a more efficient master index implementation.

[Related literature]

[11] is a full description of the system DISCOVER. A detailed description of the architecture, data representation, search algorithm and performance experiments;
[57] Algorithms for answering queries on universal relations;
[58, 59, 60] Multi-query Optimization techniques to perform the Plan Generator;

[Future work]

The plan is to: a) apply new optimization techniques as dynamic optimization techniques in the evaluation of the candidate networks; b) to experiment new cost models that access the DBMS's optimizer; c) building a more efficient master index.

[1] www.tpc.org/tpch/
[2] http://technet.oracle.com/products/text/content.html/

A.2.3 SPARK

SPARK [61] is a top-k [62, 63] keyword query engine over relational database. The contribution of this new search engine system is the monotonic ranking function which enables to explore only the top-k results minimizing the database access and reducing the execution time. SPARK aims not to offer a new revolutionary system but to improve existing system as DISCOVER and BANKS by raising the efficiency of the algorithms which have the duty to build the queries statement, retrieve and rank data. The contributions of this search engine are: *a)* It introduces a novel and nontrivial ranking method that adapts the IR ones for ranking heterogeneous joined results of database tuples. *b)* Propose two algorithms, "Skyline sweeping" and "Block Pipeline" to provide efficient processing mechanism based on the new ranking methods. *c)* Conducts comprehensive experiments on large-scale real databases containing up to 10 milion tuples.

[Representation model]

Similar to previous systems, SPARK is designed to work on relational databases that supports full-text query and inverted index (for more info refer to DBXplorer [12]).

[User input]

Besides the list of the keywords, the user has to specify the maximum number of results his/she is interested in and optionally can specify the AND or OR semantic for the query. The default mode is the OR semantic for a more flexible result ranking.

[Keyword analysis]

There is no key analisis.

[Search algorithm]

Differently from the other systems, the steps of the algorithm are not well defined because of the strong relation between the candidate generator, the query statement execution and ranking. This relation is attributed to the fact that ranking is made over the candidate networks and not over result any more. In the former systems sequence was generate candidate-retrieve-rank but in SPARK it changes in generate candidate-rank-retrieve. This choice is due to the need to stop retrieving data after the first k results. The difference that SPARK introduces is the ranking function. The previous

serach engines (like: DBXplorer, BANKS, DISCOVER etc.) ranking funktions are based only in number of join of the queries the data are retrieved. In SPARK Introduces the IR techniques and introduce three different ranking parameters. What is really new and innovative is the candidate generator which allow building monotonically the candidate networks [11] from the higher ranked to the lower one. So we can stop generating and retrieving data when we achieved our goal, that is, retrieving only the top k ranking result. This performs drastically the system minimizing the execution time and the access time to the database. What remains unmodified if the data publishing implemented over full-text query inverted index. SPARK propose a solution based on the idea of modeling a JTT as a *virtual document*. Consequently, the entire results produced by the CN will be modeled as a document collection. A vary similar notion of virtual document was proposed in [64] but the SPARK's definition differs as it is query-specific and dynamic. The final scoring function is the result of the product of three different scoring function.

$$score(\mathrm{T,Q}) = score_a(\mathrm{T,Q}) \cdot score_b(\mathrm{T,Q}) \cdot score_c(\mathrm{T,Q})$$

Where:

IR ranking score $score_a$(T,Q) is the IR-style relevance score based on the model of virtual documents;

Completeness factor score $score_b$(T,Q), motivated by the believe that users usually prefer documents matching many keywords to those matching only few ones. Derives from an extended Boolean model [65];

Size Normalization Factor score $score_c$(T,Q) is a normalization factor that don't penalize too much the moderate-size CNs;

The new algorithm, called "Skyline sweeping", guarantee not to incur any unnecessary checking and thus has the minimal number of access to the database.

[User feedback]

There is no user feedback.

[Implementation]

The system work on relational databases that supports full-text query and inverted index. The implementation supports both Oracle and MySQL.

[Performance evaluation]

Evaluation over effectiveness and efficiency has been conducted over real databases such as: Internet Movie Database[1], DBLP[2] and Mondial[3]. The databases used are the Linux version of Oracle 10g Express and MySQL v5.0.18, both with their default configuration. Indexes were build on all primary key and foreign key attributes. Four algorithms where taken into study: **Spars** algorithm, the global pipeline algorithm (**GP**), the skyline sweeping algorithm (**SP**) and the block pipeline algorithm (**BP**). There were implemented in JAVA with JDK 1.5 and JDBC for database interaction. To measure the effectiveness two metrics were adopted (these metrics are used in [63]): a) number of top-1 answer that are relevant and b) reciprocal rank (**R-Rank**). Evaluating the effectiveness, the results show that the R-Rank is higher that other methods as it returned the relevant result as top-1 result for 16 out of the 18 DBPL queries while the method in [27] returs the relevant result within top-5 and [66] within Top-20 results. Evaluating the efficiency:

a) BP is usually the fastest algorithm in both DBLP and IMDB

b) SS usually outperforms Sparse and GP. When k is small, the SS shows more performance advantages.

c) Sparse and GP have almost the same performance. In general, while Sparse might lose for small k values or easy query, its performance doesn't deteriorate too much for large K or hard queries.

d) All algorithms are more responsive for smaller k vales. Since the Spark ranking function usually returns the relevant answer as the top-1 answer, the execution time is an important indicator of system performance.

[Related literature]

[62, 63]: Firts studies on top-k algorithms;

[67, 68, 69]: optimization techiniques for other variants of top-k queries;

[65]: Extended Boolean Model;

[Future work]

The system is to be considered complete and no future work is scheduled.

[1] http://www.imdb.com/interfaces/

[2] http://sites.wiwiss.fu-berlin.de/suhl/bizer/d2rq/benchmarks

[3] http://www.dbis.informatik.uni-goettingen.de/Mondial

A.2.4 PRECIS

PRÉCIS [14] is an unstructured keyword-based query search engine that generates a structured answer. In particular it generates an entire multi-relational database instead of the typical individual relation. This database is a *logical subset* of the original one. The search engine is based on a framework that supports Précis queries that contain multiple terms combined with the logical operators *AND*, *OR* and *NOT*. It provides novel algorithms for the creation of the logical database subset schema and its population. Present an extensive of experimental results.

[Representation model]

As the most search engine, Précis works on relational databases that supports full-text query and inverted index. Indexing data by using the inverted index is common in many search engine. The representation of the database is based on the graph concept but it defers from the other search engines as it considers even the attributes as nodes and give a weight to all the edges that connect to nodes.
A relational schema is denoted as R = $\{R_1,\ldots, R_n\}$. Each relation R_i has a set of attributes $A_i = \{A_j^i : 1 \leq i \leq k^i\}$. A *Database schema graph* **G(V,E)** is a directed graph corrisponig to the database **D**. There are two types of nodes in **V**: (a) relational nodes **R** for each relation in the schema and (b) attribute node **A** for each attribute of each relation in the schema. Consequently, there are two types of edges in **E**: (a) projection edges Π one for each attribute node emanating from its container relation node and ending at the attribute node and (b) join edges **J** emanating form e relational node and ending at another relational node. So the database schema is represented by the schema **G(V,E)** where $\mathbf{V} = \mathbf{R} \cup \mathbf{A}$ and $\mathbf{E} = \Pi \cup \mathbf{J}$. Each edges has a weight assigned which indicates the relation between the interconnected nodes. Π edges are undirected and has only one weight associated and **J** edges are directed so they have two weights, one for each direction. The range for the vales is [0,1]. For example, the weight of the edge W(“MOVIE”, “GENRE”) = 1 and W(“GENRE”, “MOVIE”) = 0.7. The reason for this choose is that if we are interested about movies we what to know also about the respective genre but if are interested in genre we are less interested in all the movies related. While navigating through the graph we multiply the weight of the edges and decide if the information is relevant for our search.

[User input]

The user is asked to input the keywords over which search will be execute. The keyword must be atomic. Moreover, the user can use logical operators such as *AND*($\wedge$), *OR*($\vee$) and *NOT*($\neg$) for more Précis query as the following example:
("*Clint Eastwood*" AND "*thriller*") OR ("*Gregory Peck*" AND NOT "*drama*")

[Keyword analysis]

There is no keyword analysis.

[Search algorithm] The architecture of the system consists of three steps: query parsing, logical subset schema creation, logical subset population.
Query parsing Give a query Q over a database D with schema graph G, this phase performs two tasks. First, it transforms Q in disjunctive normal form. For this task are used well known algorithms for DNF transformation [70]. Then in consults the an inverted index over the content of the database and for each disjunct X_i of Q and a set of initial relations IR is retrieved. If no initial relation are found subsequent steps are not executed.
Logical subset schema creator This phase creates the schema of the logical database subset containing the attributes where the keywords were found, the relative tables. The tables are joined using the schema graph G. The subgraph is enriched adding the relations R and the attributes A that satisfy the threshold.
Logical subset population This phase populates schema **G**' to create the logical database subset *L*. This contains initial tuples as well as tuples on the remaining relations of **G**', based on the query semantics and the constraints provided, all projected onto their attributes that appear as projection edges on **G**'.

[User feedback]

There is no implicit user feedback. The query are logged and later analyzed to define user profiles and redefine weights for schema navigation.

[Implementation]

A prototype system has been developed in C++. The RDBMS used is Oracle 9i release 2. The mechanism of the inverted index is implemented in Oracle PLSQL. The database used are IMDB (www.imdb.com) which consists in a small number of

relations and a large number of tuples and restaurant data (www.gastronomia.gr) which consists in large number of relational with relatively few tuples.

[Performance evaluation]

The performance are evaluates over the following parameters:
width(Q) that is the number of disjuncts in Q
span(Q) that is the maximum number of conjuncts in a disjuncts of Q.
C the relevance constraint that defines the threshold of the score on which two relation can be considered relevant.

For the experiments, the relevance constraint *C* is equal to 0.3, so that a large part of the database schema graph could be explored and logical subset of characteristics can be generated.

The parameters that denote the characteristics of the logical subset are:
#S: the number of subgraphs that comprise a logical database set;
#RS: the number of relations per subgraph;
#IRS: the number of initial relations per subgraph;
#TLS: the number of tuples in the logical subset;

The first step consists in evaluating the effects if query characteristics. The following illustrates the effect of width(Q) and span(Q) in the logical subset produced.
a) The trend of the initial relations per subgraph (#IRS) as function of span(Q) with width(Q) = 1 and C = 0.3. The number of #IRS increases with the number of query terms with e coefficient next to 1.
b) The trend of the initial relations per subgraph (#IRS) as function of width(Q) if span (Q) = 1 (a second experiment with span (Q) = 2) and C = 0.3; The number of #IRS remain constant while width(Q) increases;
c) The trend of the number of subgraphs #S as function of span(Q) for width(Q) = 1. Since more than one initial relation may be found for the same query term this increases the number of subgraph on the database that contain all the keyword;
d) The trend of the number of subgraphs #S as function of width(Q) for span(Q) = 1. Same consideration as the previous;
e) The trend of the number of tuples #TLS as function of span(Q) for width(Q) = 1. For span(Q) ¿ 1 we see that the #TLS is next to 0;

f) The trend of the number of tuples #TLS as function of width(Q) for span(Q) = 1. The #TLS increase constantly with the increase of width(Q);

The second step evaluates the effects of constraints and weights.

Two scenarios has been evaluated. At the first scenario random weight have been generated in a range [0.85,1] and in the second scenario the range was [0.6,1]. Impact of the constraint C has been evaluated over #S, #RS and #IRS for C that changes in the range [0.1,1.0] and for the two set of weights. Each result shown represents the average of 100 queries with width(Q)=1 and span(Q)=4. All the graphs has the same trend. For constraints >threshold values tend to 0. We can notice two regions in the graphics, the active constraint range and the inactive constraint range. At the first range the constraint strongly determine the values. The inactive range is the range were all subgraphs has been explore and lowing the constraint does not change the value the graphic.

[Related literature]

[14]: a detailed description of Précis system;
[71]: the basic idea of the join weights;
[72, 73, 74]: categorization and personalization of query results and query logs;

[Future work]

Future work aim to improve the system in three direction: a) selection of tuples to include in a logical subset under constraints on its size; b) incorporation of notions of tuple ranking; and c) incorporation of the logical subsets into the internals of an open source RDBMS such as POSTGRES.

A.2.5 SQAK

SQAK (SQL Aggregates using Keywords and pronounced squawk) is a framework that enables the using to pose aggregate queries using simple keywords with little or no knowledge of the schema. SQAK adds some expressive power giving the possibility to make more complex queries by the use of the keywords. Previous search engine over relation database allow only exact keyword search. The SQL language is much more expressive then the use of exact keywords in the search. This gives no change to the keyword base search engines to substitute in a satisfactory the SQL language by the

use of keywords. SQAK tries to fulfill this gap by introducing a framework that allow non exact keyword search. This requires an accurate keyword analysis which will be introduced further.

[Representation model]

SQAK operates with two kind of data: the DB content that in indexed using the inverted index techniques for a fast access and the structure of the schema. The content of the DB is used, like in all previous search engines, to check if a keyword resides in the DB. Information about schema are used in two different phases of the process: the keyword analysis and the exploration of possible path between two relations. Besides the information about the structure, each element, like table names and attribute names, goes with a set of synonyms that will be used at the keyword analysis.

[User input]

This framework aims to offer a search engine based on keywords capable to manage aggregate queries. The user must put a correct set of keywords so that the framework can interpret as a possible aggregate query. The user does not need to have any knowledge about the structure of the schema but anyway he/she has to know some of the aggregate function and the power of the SQL language. For example, if the user want to know the number of courses that John participates, a possible way to write the query keywords may be: John num courses. We see that only John is a possible instance of the database, the keyword num and courses represent, respectively, an aggregate function and a schema element.

[Keyword analysis]

The keywords are analyzed so they can be classified as schema element (like table name or attribute name), aggregate function (like: sum, count, avg) or value. The Parser/Analyzer detects if a keyword expresses a schema element or an aggregate function using approximate string matching algorithms. Here come in help the synonyms for each schema element and function. The remaining keywords are considered as values.

[Search algorithm]

The search algorithm is divided in three modules:

- *Parser-Analyzer:* parses and analyzes the keyword queries given by the user. The analysis try to find if a keyword represents a schema element, a aggregate function or a value. It creates the Candidate Interpreters by computing the cross product of each term's list.

- *SQN Builder:* tales the CI and computes the smallest valid SQN respect to the CI. It is similar to the Candidate Networks seen in DISCOVER and other previous works. Once identified the tables and attributes involved in, the builder find the minimum path that connects the tables is the relational schema.

- *Scorer:* The set of SQN are sent to the Scorer which ranks them. The score for an SQN is the sum of the weights of its nodes and edges. The weights of the nodes are determined using the match score of the Parser/Analyzer that gives a score from 1 (perfect mathc) to ∞ (no match). The edges has unit weights.

[User feedback]

The user interaction is needed when the Scorer finds SQN with the same score or with a score very close to the best one. This may happen when there are several semantically different schema elements that have a similar name and the query is not sufficiently precise to distinguish between them. If SQAK detects more SQN with a very close score it alerts the user and present an option to examine.

[Implementation]

SQAK algorithms are implemented in Java and were run using JVM 1.6.0_02. The inverted index was constructed using Lucene. The experiments are performed on a system with a dual-core 2.16 GHz Intel Processor with 2GB RAM running over Windows XP.

[Performance evaluation]

The first aspect of the algorithm analyzed was the effectiveness of creating the right SQL query. It was compared with the Steiner algorithm and the result is that SQAK algorithm is 93% accurate while Steiner algorithm is only 60% accurate. The second analysis was made over the two parameters of the edit distance. It has been analyzed the Accuracy and the Time of building the query. Is was noticed that the accuracy is stable with γ between 0.4 and 0.8 and increasing f increases the accuracy but the

influence is very small. The execution time increases linearly with γ between 0.1 and 0.6 and exponentially between 0.6 and 0.8. Changing f does not have significant influence to the execution time. A choice of $f = 2$ or 3 if a good one.

[Related literature]
[75, 76, 77]: algorithm for discovering functional dependencies and join key constraints;
[78]: describes a technique for selecting a database from a set of different databases based on the terms in a keyword query and how they are related in the context of the database;

A.3 Overview

Thousand of websites are available navigating the World Wide Web containing text, images, video and other multimedia files. This huge amount of data is distributed in many nodes connected to the net. If we need information about any topic is quite impossible to check all the websites, not even navigate randomly with the hope to find something useful. The search engines represent a powerful tool for information discovery, specially on the web. Crawling into web sites, they collect huge amount of data, index them and make available for search using information retrieval techniques. Another strong point of the search engines is the keyword-based search. They take as input a set of keywords representative of the information the user is interested in and return a list of ranked links to web pages or document that contain (totally or partially) the keywords. This is the most simple way for a user to express his request.

Actually, the search engines, restrict their search on web documents like static web pages (HTML) and text files (.doc, .txt, .pdf etc.). But these data represents only 10% of the whole amount of information available. The other 90% resides in relational databases and is called *deep web* or *hidden web*. Most of the web sites are dynamic and the data is stored in databases. Web sites of news, blogs, forums, booking services and many other are build on demand interacting with the users. The necessity of including this kind of information is growing up everyday. Structured data hold semantic relations and this enrich the information itself.

In the following table are list the most important differences between a web page and a database which will be explained more in detail in the next paragraph.

	Web page	**Database**
Type	Flat documents	Structured data
Access	Through URL	Through DMBS
Previous knowledge	None	Data structure
Relation between data	No	Yes
Output	Flat documents (partially/totally)	Tuples

Figure A.1: Differences between web pages and databases

The difficulty of integrating these data to the actual search engines are different:

- The crawling of structured data is not possible because querying databases require to have knowledge of the schema in advance in order to access any piece of data. All web resources, instead, are accessed using URL and no other information is required.
- The structure hides the relation between data and it is crucial for it's representation. Users are not interested on data of the same attribute but data of other attributes or other tables related to.
- The output of the result is a set of ranked tuples.

Web forms represent a simple way to interrogate databases. They usually refer to a single database view. Input data is structured and values in different fields have different semantic. All these values will enrich the WHERE clause usually in AND semantic. Forms are build ad-hoc for every database which data has to be accessible by the web.

The first search engines over database have been designed in the early '90. They are based on the same idea of the web search engines: collect and index, search for keywords in the content, retrieve the target documents. In a similar way search engines over database can be divided in the following steps: a) collect and index, b) look up for keywords and retrieve references such as table and attribute name, c) build all possible configuration creating the SQL code, and d) retrieve, rank and fuse result. Both approaches are based on the idea that data are previously known. For fast search in such

amount of data index comes to help. The keywords represent only expected value of the attributes and must not express characterization of the object. For example, if the user is looking for hotel Linz in Austria the keyword must be “Linz” and “Austria”. The word “hotel” introduces errors (OR semantic) or even no solution (AND semantic). That is because the keyword hotel does not occur in any of the attribute values in the relational schema. One of the major limitation that brings this approach is that the user can not search for generic object such as, for example, hotels in Austria.

A.3.1 Collection and index

For a fast look up process, the data extracted from the schema are stored are indexed. There are two techniques commonly used: the symbol table and the inverted index. The symbol table techniques creates a table containing all the data (in string format) and the references such as table name and attribute name. The attribute that contains the data is index and data are access very fast. If a data occurs in the table then the table name and the attribute name are retrieved for further processing. This technique exploits the indexing tool over relational databases. The other technique is the inverted index. Data are stored in documents and not in database. Every document represent the couple table-attribute of the relational schema and contains all the data represented by the attribute in a specific table. This techniques is the most used because inverted index maintenance has better performance than DB index.

A.3.2 Building configurations

For each keyword there is a set of nodes N(T, A) associated where T is the table and A the Attribute where the keyword was found. A Candidate Network is a subgraph (of the graph G representing the database schema) which contains leaf nodes (nodes containing all the keywords) and join nodes (nodes that are necessary to connect all the leaf nodes). In Boolean AND semantic a CN contains all the keywords and this approach is the most used. In Boolean OR semantic a CN can contain not all the keyword giving so partial answers. This approach is useful when does not exists a CN which contains all the keyword and instead of giving no answer the CN builder returns CN with partial ones. The Boolean AND is the only approach used in DBXplorer, BANKS and DISCOVER. In SPARK, the user can specify whenever wants to use the

AND or OR semantic. The AND semantic is used as default. For each keyword there is a set of nodes N(T, A) associated where T is the table and A the Attribute where the keyword was found. A Candidate Network is a subgraph (of the graph G representing the database schema) which contains leaf nodes (nodes containing all the keywords) and join nodes (nodes that are necessary to connect all the leaf nodes). In Boolean AND semantic a CN contains all the keywords and this approach is the most used. In Boolean OR semantic a CN can contain not all the keyword giving so partial answers. This approach is useful when does not exists a CN which contains all the keyword and instead of giving no answer the CN builder returns CN with partial ones. The Boolean AND is the only approach used in DBXplorer, BANKS and DISCOVER. In SPARK, the user can specify whenever wants to use the AND or OR semantic. The AND semantic is used as default. Later the CNs are translated in SQL queries so data can be retrieved and sent to the output.

Different algorithms has been developed to minimize the computation time and to increase the efficiency of building the CNs. The first algorithm approach were to find all the possible solution which will be ranked in a following step before retrieving data and sending the result to the output. Due to efficiency improvement this approach changes fusing the two algorithms, the CN building and ranking, in a single algorithm. It was not necessary retrieving all the possible solutions but the most significant ones, that is the solutions with the highest rank score. This means time and computation saving. The top-k approach would require at output no more than k result ordered by rank score.

In the following sections are listed the algorithms used in the for CN building and ranking.

[Naive algorith] The Naive algorithm issue a SQL query for each CN for a top-*k* query and union them to find the top-*k* results by their relevance scores. As we can see, this algorithm is not efficient because it retrieves all the results defines by the CNs and later use a sort-merge manner to identify the final top-*k* results of the query.

[Sparse algorith] It is an evolution of the Naive algorithm. With Sparse algorithm is possible to discard at any point and at any time any CN that is guaranteed not to produce a top-*k* match for the query. Specifically, the Sparse algorithm computes a bound MPSi (Maximum Possible Score) on the maximum possible score of a tuple tree derived from a CN C_i. If MPS_i does not exceed the actual score of *k* already

produced tuples tree, then CN C_i can be safely removed for further consideration. To calculate MPSi we apply the combing function to the top tuples of the non free tuple sets of C_i. That is, MPS_i is the score of a hypothetical join tree of tuples T that contains top tuples from every non-free tuples set in C_i. As a further optimization, the CNs for a query are evaluated in ascending size order. This way, the smallest CNs, which are the least expensive to process and the most likely to produce high score tuples trees using the combination function are evaluated first.

[Single Pipelined algorith] The Single Pipelined Algorithm receive as input a candidate network C and the non-free tuple sets (TS_1,...,TS_v). The tuples are sorted by their score calculated by the IR Engine. The output consists of a stream of joining trees of tuples T in descending order. The tuples that are sent to the output are those which has a score greater than the MPFS (Maximum Possible Future Score). The MPFS is calculated as the maximum of the $MPFS_i$ for every tuple set. The $MPFS_i$ is the maximum possible future score of any tuple tree that contains a tuple from TS_i that has not yet been retrieved.

A precise calculation of MPFSi would require multiple database queries. As an alternative to this expensive computation it is produced an overestimated MPFSi computed as the score of a hypothetical tree of tuples consisting of the next unprocessed tuple ti from TS_i and the top-ranked tuple t_j^{top} of each tuple set TS_j, for i $\neq$ j. This algorithm is not efficient when used separately for each CN is the main building block of the efficient Global Pipelined algorithm.

[Global Pipelined algorithm] It is built on the Single Pipelined algorithm to efficiently answer a top-*k* keyword query over multiple candidate networks. The algorithm receives as input a set of candidate networks together with their associated non-free tuple sets and produce as output a stream of joining trees of tuples ranked by their overall score for the query. The key idea of the algorithm is that all CNs of the keyword query are evaluated concurrently following an adaptation of a priority preemptive, round robin protocol [5], where the execution of each CN corresponds to a process. Each CN is evaluated using a modification of the Single Pipelined algorithm, with the priority of a process being the MPFS value of its associated CN. Initially, a "minimal" portion of the most promising CN C_c (i.e., Cc has the highest MPFS value) is evaluated. Specifically, this minimal portion corresponds to processing the next tuple from C_c. After this, the priority of C_c (i.e., MPFS c) is updated, and the CN with the next highest

MPFS value is picked. A tuple-tree result is output if its score is no lower than the current value of the Global MPFS, GMPFS, defined as the maximum MPFS among all the CNs for the query. Note that if the same tuple set TS is in two different CNs, it is processed as two separate (but identical) tuple sets. In practice, this is implemented by maintaining two open cursors for TS .

[Hybrid algorithm] The Hybrid algorithm critically relies on the accuracy of the result-size estimator. For queries with OR semantics, we can simply rely on the RDBMSs result-size estimates, which we have found to be reliable. In contrast, this estimation is more challenging for queries with AND semantics. We can obtain from the RDBMS an estimate S of the number of tuples derived from a CN (i.e., the number of tuples that match the associated join conditions), but we need to adjust this estimate so that we consider only tuple trees that contain all query keywords. To illustrate this simple adjustment, consider a two-keyword query W = w1, w2 with two non-free tuple sets TS_1 and TS_2. If we assume that the two keywords appear independently of each other in the tuples, we adjust the estimate S by multiplying by $\frac{|TS_1^{w_1}|\cdot|TS_2^{w_2}|+|TS_1^{w_2}|\cdot|TS_2^{w_1}|}{|TS_1|\cdot|TS_2|}$ formula, where TS_i^w is the subset of TS_i that contains keyword w. (An implicit simplifying as assumption in the computation of this adjustment factor is that no two keywords appear in the same tuple.)

A.3.3 Ranking function

The search engines use ranking functions to ranks data before sending to the output. This is to give to the user potentially the best answer first. Many approaches has been used to identify the best ranking function but it still remains an object of study. In free form keyword-based search engine systems the ranking function play an important role on the efficiency. The last approaches, stop the search processing after the first *k* answers that are considered the most relevant. A good ranking function bring benefits in user satisfaction, by giving the best answer first and in efficiency by reducing the execution time stopping the search process after the best answers are individuated.
The first ranking function were based on the idea of proximity. The documents in which keywords occur close to one another are ranked higher. The first search engine like DBXplorer and DISCOVERY ranked the tuples by their number of joins.

Score$(T, Q) = \frac{1}{size(T)}$ if T contains all the keywords Q. Alternatively, BANKS introduced another scoring scheme: *Score*$(T, Q) = f_r(T) + f_n(T) + f_p(T)$ if T contains all the keywords Q. where the f_r(T) measures how "related" the relations of the tuple T are, f_n(T) depends on the weight of the tuples of T (as determined by the PageRank-inspired technique) and f_p(T) is a function of the weight of the edges of T. Other ranking algorithms may use the multiplication for the terms above instead of the sum. This techniques are based on the structure of the query and do not leverage the relevance-ranking strategies developed by the IR community over years. Modern RDBMSs already include IR-style relevance ranking functionality over the individual text attributes. We consider two score functions: the single attribute score function and the combine score function.

Single-attribute IR-style relevance scores $Score(ai, Q)$ for each textual attribute a i $\in$ T and query Q, as determined by an IR engine at the RDBMS:

$$Score(a_i, Q) = \sum_{w \in Q \cap a_i} \frac{1 + \ln(1 + \ln(tf))}{(1 - s) + s\frac{df}{avdf}} \ln(\frac{N + 1}{df})$$

where, for a word *w*, *tf* is the frequency of *w* in a_i, *df* is the number of tuples in a_i's relation with word *w* in this attribute, *dl* is the size of ai in characters, *avdl* is the average attribute-value size, N is the total number of tuples in a_i's relation, and *s* is a constant (usually 0.2).

A function Combine, which combines the single-attribute scores into a final score for T.

$$Combine(Score(A, Q), size(T)) = \frac{\sum_{a_i \in A} Score(a_i, Q)}{size(T)}$$

Other combine function can be used instead of the one above.

A. A REASONED STATE OF THE ART ABOUT KEYWORD BASED SEARCH ENGINES

Appendix B

Keymantic

B.1 Configuration Manager

Defines the separator for the annotation of the attributes;

```
AttributeSeparator = .
```

Defines the value that will be add/subtract for attribute dependences during the cost Hungarian algorithm;

```
DeltaCostOfProximity = 25
```

Defines in percentage the minimum cost limit taken in the solution list;

```
CoefficientOfProximity = 0.9
```

Defines the schema filename to be imported;

```
SchemaFilename = src\files\schema_name.xml
```

Defines the schema graph filename to be imported;

```
GraphFilename = src\files\xmlgraph_schema_name.xml
```

Defines the log filename where the log will be saved;

```
LogFilename = src\files\log.txt
```

Defines in the generation of the CSV file containing the solution list is required;

```
CSVGenerateFile = false
```

Defines the filename where the resulys will be stored in CSV format;

```
CSVFilename = \path\test.csv
```

Defines the tab separator and the solution separator for the CSV file;

```
CSVTabSeparator = ,
CSVMapSeparator = ;
```

Defines if keywords are required to match exactly the schema element or similarity function can be applied for term matching (false - use similarity function / true - use exact matching);

```
ExactTermMatching = false
```

Defines the similarity coefficient if similarity function is used;

```
MinProximity = 0.7
```

Defines is external similarity functions are going to be used (0 - don't use external similarity / 1 - use external similarity)

```
ExternalSimilarity = 0
```

Defines the maximum value of weight for Hungarian algorithm;

```
MaxWeight = 100
```

The following rules define the proximity coefficients between two attributes;

		Same table ID	Same table	FK connected ID	FK connected
FW	d = 1	R1 (0.75)	R2 (0.50)	R7 (0.30)	R8 (0.18)
	d >1	-	R3 (0.25)	R9 (0.12)	R9 (0.12)
BW	d = 1	R4 (0.37)	R5 (0.25)	R10 (0.17)	R11 (0.09)
	d >1	-	R6 (0.12)	R12 (0.06)	R12 (0.06)

Figure B.1: Rule values

```
Rule_1 = 0.75
Rule_2 = 0.50
Rule_3 = 0.25
Rule_4 = 0.37
Rule_5 = 0.25
Rule_6 = 0.12
Rule_7 = 0.30
Rule_8 = 0.18
Rule_9 = 0.12
Rule_10 = 0.17
Rule_11 = 0.09
Rule_12 = 0.06
```

Defines the transaction properties

```
TransactionUsername = username
TransactionPassword = password
TransactionUrl = jdbc:mysql://localhost/keymantic
TransactionDriver = com.mysql.jdbc.Driver
```

Defines the folder where Lucene creates index

```
LuceneSrcFolder = src/files/lucene/src
LuceneIndexFolder = src/files/lucene/index
```

B.2 Meta-date file representation

The meta information used to evaluate the mapping between keywords and schema elements are collected in a XML file as in the following. The data are collected from the meta information table available in different DBMS-s. The wrapper in made in Java and runs offline before the system is available to the users.

```
<?xml version=``1.0'' encoding=``ISO-8859-1''?>
<schema name = ``'' >
```

For each schema, all the tables are listed as nodes. These nodes contain the meta information, in form of node attribute, that describe the schema element. The “name ” attribute contain the table name, the “synonyms” contains the different synonyms that the table name may have, the “hyponyms” and the “hypernyms” contain the respectively the hyponyms and the hypernyms of the table name, the “semanticid” contains the most representative attribute of the table and the “foreignKeys” the table list which have a foreign key relation.

```
<table name = ``'' synonyms = ``'' hyponyms = ``''
hypernyms = ``'' semanticid = ``'' foreignKeys =``'' >
```

For each table represented as nodes all the related attributes are list as sub-nodes. Beyond the attributes “name”, “synonyms”, “hyponyms” and “hypernyms”, similar to the previous one, it contains the “datatype” such as *string* and *integer* and the “regex”, a more refined regular expression of the possible values.

```
<attribute name = ‘‘’’ datatype = ‘‘’’ regex = ‘‘’’
synonyms = ‘‘’’ hyponyms = ‘‘’’ hypernyms = ‘‘’’ \>
...
<\table >
<\schema >
```

B.3 Graph file representation

The ER schema extracted from the DBMS meta information and is represented in a XML file. This is used to build the visual graph for the user interface, to calculate the joining tree by the Candidate Network builder and build the query string by the SQL builder.

```
<?xml version=‘‘1.0’’ encoding=‘‘ISO-8859-1’’ ?  >
<xmljgraph name="" >
```

The nodes represent the relations (tables) of the ER schema. Each node has an progressive “id”, the attribute “name” that represents the table name and the attribute “pk” that represents the primary key of the relation which is necessary to build the join condition of the SQL query string. Other attributes such as “x”, “y”, “width” and “length” are used by the visual graph builder when the ER schema is shown graphically in the user interface.

```
<nodes >
<node index=‘‘’’ name=‘‘’’ pk=‘‘’’
x=‘‘’’ y=‘‘’’ length=‘‘’’ width=‘‘’’/ >
</nodes >
```

The edges represent the foreign key connection between two relations. It contains a progressive attribute named “index”, the attribute “startindex” represents the index of the relation that contains the foreign key, “startname” its name and “startkeys” the key name list. The attribute “endindex” represents the index of the relation that contains the primary key, “endname” represents its name, “endkeys” the key name list. The attribute “jointable” contains the table name that is used as join table and the attributes “startlabel” and “endlabel” the labels representing the relation type such as “many-to-many”, “one-to-one” or “one-to-many”.

```
<edges >
<edge index=‘‘’’ startindex=‘‘’’ endindex=‘‘’’
startname=‘‘’’ endname=‘‘’’ startkeys=‘‘’’ endkeys=‘‘’’
jointable=‘‘’’ startlabel=‘‘’’ endlabel=‘‘’’ / >
</edges >
</xmljgraph >
```

Appendix C

Ubimedic2

C.1 Packages

- **agents** contains all the framework class implementation (such as SuperAgent, Activator, SA and OA class);
- **agents.operative.mobile** contains the classes representing ambulances;
- **agents.operative.fixed** contains the classes representing hospitals;
- **agents.service.device** contains the classes representing medical devices such as Ecg and Oximeter;
- **agents.service.client** contains the classes representing PDAs and smart-phones;
- **conf** contains the ConfigurationManager class and properties file;
- **files** contains the xml file;
- **jar** contains the necessary Jar libraries;
- **scripts** contains the scripts necessary to launch the agents during test;
- **utils** contains utility classes such as database manager and Gps manager;
- **doc** contains the java documentation;

C.2 Script

In Ubimedic2, agents representing Activators and Operative Agents must be executed separately. Each of these agents must create its own container and Service Agents can be added on it.

An Activator can be launched with the following script:

```
java -cp *.jar jade.Boot -name project_name
-container -container-name container_name
agent_name@agent_type:agent_class
```

where

- **.jar* are all the necessary Jar file containing the necessary classes;
- *project_name* is the name of the project;
- *container_name* is the name of the container that will be created;
- *agent_name* the unique name of that identifies the agent;
- *agent_type* the agent type such as OperativeCentre and Ambulance;
- *agent_class* the agent class name;

An code example can be the following,

```
java -cp *.jar jade.Boot -name Ubimedic
-container -container-name Amb01Cont
Amb01@Ambulance:agents.moa.Ambulance
```

C.3 Configuration files

```
<ubimedic name = “”>
    <Activator name = “” class = “” args = “”\>
    <OperativeAgent name = “” class = “” args = “” repository=“” >
        <DeviceAgent name = “” class = “” args = “”\>
        ...
    <\OperativeAgent >
    ...
<\ubimedic >
```

Ubimedic2 test can be activated using an XML file that generates all the necessary agents. There are three node type in defined in XML file: *Activator*, *OperativeAgent* and *ServiceAgent*. All nodes has three main attributes: *name* containing the unique agent name, *class* containing the class name inside the project and *args* containing argument list necessary to activate the agent. The Operative Agent has one more attribute such as *repository* containing the repository name where the Operative agent will be created and all the respective Service Agents.

```
<Hospitals >
    <Hospital name = “” gps = “” >
        <Ward name = “” pathology = “” availability = “” \>
        ...
    <\Hospital >
    ...
<\Hospitals >
```

When the MOA is activated, it loads all the hospital available in the territory, their location and the wards it contains. *Hospital* node contains two attributes: *name* containing the hospital name and *gps* containing the location representation. The child node of the Hospital node is the Ward node that for each hospital contains the information of the wards such as the treated pathology and the maximum number of patient it can treat.

References

[1] MICHEAL K. BERGMAN. **The Deep Web: surfacing hidden value**. *Journal of Electronic Publishing*, **Vol 7**, 2011. 6

[2] FRANÇOIS BOURGEOIS AND JOHN-CLAUDE LASSALLE. **An Extension of the Munkres Algorithm for the Assignment Problem to Rectangular Matrices**. *Communications of ACM*, **14**(12):802–804, 1971. 9, 24, 38, 39

[3] AMIT SINGHAL, CHRIS BUCKLEY, AND MANDAR MITRA. **Pivoted Document Length Normalization**. In *SIGIR*, pages 21–29, 1996. 10

[4] SERGEY BRIN AND LAWRENCE PAGE. **The Anatomy of a Large-Scale Hypertextual Web Search Engine**. *Computer Networks*, **30**(1-7):107–117, 1998. 10

[5] DANIELA FLORESCU, DONALD KOSSMANN, AND IOANA MANOLESCU. **Integrating Keyword Search into XML Query Processing**. In *BDA*, 2000. 10

[6] YUNYAO LI, CONG YU, AND H. V. JAGADISH. **Schema-free XQuery**. In *VLDB*, pages 72–83, 2004. 10

[7] MARTIN THEOBALD, HOLGER BAST, DEBAPRIYO MAJUMDAR, RALF SCHENKEL, AND GERHARD WEIKUM. **TopX: efficient and versatile top-k query processing for semistructured data**. *VLDB Journal*, **17**(1):81–115, 2008. 10

[8] SANDEEP TATA AND GUY M. LOHMAN. **SQAK: doing more with keywords**. In *SIGMOD*, pages 889–902. ACM, 2008. 11, 16, 42

[9] JEFFREY XU YU, LU QIN, AND LIJUN CHANG. *Keyword Search in Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010. 11

[10] SOUMEN CHAKRABARTI, SUNITA SARAWAGI, AND S. SUDARSHAN. **Enhancing Search with Structure**. *IEEE Data Eng. Bull.*, **33**(1):3–24, 2010. 11

[11] VAGELIS HRISTIDIS AND YANNIS PAPAKONSTANTINOU. **DISCOVER: Keyword Search in Relational Databases**. In *VLDB*, pages 670–681, 2002. 11, 16, 27, 47, 107, 109, 111

[12] S. AGRAWAL, S. CHAUDHURI, AND G. DAS. **DBXplorer: A System for Keyword-Based Search over Relational Databases**. In *ICDE*, pages 5–16. IEEE Computer Society, 2002. 11, 16, 47, 107, 110

[13] B. ADITYA, GAURAV BHALOTIA, SOUMEN CHAKRABARTI, ARVIND HULGERI, CHARUTA NAKHE, PARAG, AND S. SUDARSHAN. **BANKS: Browsing and Keyword Searching in Relational Databases**. In *VLDB*, pages 1083–1086, 2002. 11

[14] ALKIS SIMITSIS, GEORGIA KOUTRIKA, AND YANNIS E. IOANNIDIS. **Précis: from unstructured keywords as queries to structured databases as answers**. *VLDB Journal*, **17**(1):117–149, 2008. 11, 113, 116

[15] LU QIN, JEFFREY XU YU, AND LIJUN CHANG. **Keyword search in databases: the power of RDBMS**. In *SIGMOD*, pages 681–694. ACM, 2009. 11

[16] R. KUMAR AND A. TOMKINS. **A Characterization of Online Search Behavior**. *IEEE Data Engineering Bulletin*, **32**(2):3–11, 2009. 14, 16, 24, 26

[17] E. RAHM AND P. A. BERNSTEIN. **A survey of approaches to automatic schema matching**. *VLDB Journal*, **10**(4):334–350, 2001. 23, 25, 29

[18] RAINER BURKARD, MAURO DELL'AMICO, AND SILVANO MARTELLO. *Assignment Problems*. SIAM Society for Industrial and Applied Mathematics, 2009. 23, 38

[19] SERGEY MELNIK, HECTOR GARCIA-MOLINA, AND ERHARD RAHM. **Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching**. In *ICDE*, pages 117–128. IEEE Computer Society, 2002. 24

[20] RUDI CILIBRASI AND PAUL M. B. VITÁNYI. **The Google Similarity Distance**. *IEEE TKDE*, **19**(3):370–383, 2007. 25, 33

[21] THANH TRAN, HAOFEN WANG, SEBASTIAN RUDOLPH, AND PHILIPP CIMIANO. **Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data**. In *ICDE*, pages 405–416. IEEE Computer Society, 2009. 26

[22] Y. KOTIDIS, A. MARIAN, AND D. SRIVASTAVA. **Circumventing Data Quality Problems Using Multiple Join Paths**. In *CleanDB*, 2006. 27

[23] JENS BLEIHOLDER AND FELIX NAUMANN. **Data fusion**. *ACM Comput. Surv.*, **41**(1), 2008. 27

[24] SONIA BERGAMASCHI, CLAUDIO SARTORI, FRANCESCO GUERRA, AND MIRKO ORSINI. **Extracting Relevant Attribute Values for Improved Search**. *IEEE Internet Computing*, **11**(5):26–35, 2007. 29, 33

[25] WILLIAM W. COHEN, PRADEEP D. RAVIKUMAR, AND STEPHEN E. FIENBERG. **A Comparison of String Distance Metrics for Name-Matching Tasks**. In *IIWeb*, 2003. 30

[26] WILLIAM WEBBER. **Evaluating the Effectiveness of Keyword Search**. *IEEE Data Engineering Bulletin*, **33**(1):55–60, 2010. 42

[27] FANG LIU, CLEMENT T. YU, WEIYI MENG, AND ABDUR CHOWDHURY. **Effective keyword search in relational databases**. In *SIGMOD*, pages 563–574, 2006. 47, 112

[28] MICHAEL WOOLDRIDGE. *An Introduction to Multiagent Systems*. Wiley, 2. edition, 2009. 55

[29] GERHARD WEISS, editor. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, Cambridge, MA, USA, 1999. 55

[30] A. MORENO AND CATHERINE GARBAY. **Software agents in health care**. *Artificial Intelligence in Medicine*, **27**(3):229–232, 2003. 55

[31] JOHN NEALON AND ANTONIO MORENO. **Agent-Based Applications in Health Care**. In *Applications of Software Agent Technology in the Health Care Domain, Whitestein Series in Software Agent Technologies, Birkhuser Verlag*, pages 3–18. Verlag, 2003. 56

[32] DAVID ISERN, DAVID SÁNCHEZ, AND ANTONIO MORENO. **Agents applied in health care: A review**. *I. J. Medical Informatics*, **79**(3):145–166, 2010. 56

[33] KEITH DECKER AND JINJIANG LI. **Coordinated Hospital Patient Scheduling**. In YVES DEMAZEAU, editor, *ICMAS*, pages 104–111. IEEE Computer Society, 1998. 56

[34] MARKUS HANNEBAUER AND SEBASTIAN MÜLLER. **Distributed constraint optimization for medical appointment scheduling**. In *Agents*, pages 139–140, 2001. 56

[35] CHRISTIAN HEINE AND STEFAN KIRN. **Adapt at agent.hospital - agent based support of clinical processes**. In *ECIS*, 2004. 56

[36] JAN ERIC LARSSON AND BARBARA HAYES-ROTH. **Guardian: An Intelligent Autonomous Agent for Medical Monitoring and Diagnosis**. *IEEE Intelligent Systems*, **13**(1):58–64, 1998. 56

[37] GOKCE LALECI, ASUMAN DOGAC, MEHMET OLDUZ, IBRAHIM TASYURT, MUSTAFA YUKSEL, AND ALPER OKCAN. **SAPHIRE: A MultiAgent System for Remote Healthcare Monitoring through Computerized Clinical Guidelines**. 56

[38] MIREN I. BAGÜÉS, JESÚS BERMÚDEZ, ALFREDO BURGOS, ALFREDO GOÑI, ARANTZA ILLARRAMENDI, JIMENA RODRÍGUEZ, AND ALBERTO TABLADO. **An Innovative System that Runs on a PDA for a Continuous Monitoring of People**. In *CBMS*, pages 151–156. IEEE Computer Society, 2006. 56

[39] SHAILENDRA SINGH 0002, BUKHARY IKHWAN ISMAIL, FAZILAH HARON, AND CHAN HUAH YONG. **Architecture of Agent-Based Healthcare Intelligent Assistant on Grid Environment**. In KIM-MEOW LIEW, HONG SHEN, SIMON SEE, WENTONG CAI, PINGZHI FAN, AND SUSUMU HORIGUCHI, editors, *PDCAT*, **3320** of *LNCS*, pages 58–61. Springer, 2004. 56

[40] HORACIO GONZÁLEZ-VÉLEZ, MARIOLA MIER, MARGARIDA JULIÀ-SAPÉ, THEODOROS N. ARVANITIS, JUAN MIGUEL GARCÍA-GÓMEZ, MONTSERRAT ROBLES, PAUL H. LEWIS, SRINANDAN DASMAHAPATRA, DAVID DUPPLAW, ANDREW PEET, CARLES ARÚS, BERNARDO CELDA, SABINE VAN HUFFEL, AND MAGÍ LLUCH I ARIET. **HealthAgents: distributed multi-agent brain tumor diagnosis and prognosis**. *Appl. Intell.*, **30**(3):191–202, 2009. 56

[41] DAVID ISERN, DAVID SÁNCHEZ, AND ANTONIO MORENO. **HeCaSe2: A Multi-agent Ontology-Driven Guideline Enactment Engine**. In HANS-DIETER BURKHARD, GABRIELA LINDEMANN, RINEKE VERBRUGGE, AND LÁSZLÓ ZSOLT VARGA, editors, *CEEMAS*, **4696** of *Lecture Notes in Computer Science*, pages 322–324. Springer, 2007. 56

[42] ZAFAR IQBAL HASHMI, SYED SIBTE RAZA ABIDI, AND YU-N CHEAH. **An Intelligent Agent-based Knowledge Broker for Enterprise-wide Healthcare Knowledge Procurement**. In *CBMS*, pages 173–178. IEEE Computer Society, 2002. 56

[43] MADALINA CROITORU, BO HU, SRINANDAN DASMAHAPATRA, PAUL H. LEWIS, DAVID DUPPLAW, ALEX GIBB, MARGARIDA JULIÀ-SAPÉ, JAVIER VICENTE, CARLOS SÁEZ, JUAN MIGUEL GARCÍA-GÓMEZ, ROMAN ROSET, FRANCESC ESTANYOL, XAVIER RAFAEL PALOU, AND MARIOLA MIER. **Conceptual Graphs Based Information Retrieval in HealthAgents**. In *CBMS*, pages 618–623. IEEE Computer Society, 2007. 56

[44] TIA GAO, TAMMARA MASSEY, LEO SELAVO, DAVID CRAWFORD, BOR RONG CHEN, KONRAD LORINCZ, VICTOR SHNAYDER, LOGAN HAUENSTEIN, FOAD DABIRI, JAMES JENG, ARJUN CHANMUGAM, DAVIDWHITE,

MAJID SARRAFZADEH, AND MATT WELSH. **The Advanced Health and Disaster Aid Network: A Light-WeightWirelessMedical System for Triage**. In *Transactions on Biomedical Circuits and Systems*, page 203, 2007. 56

[45] NATHAN SCHURR, JANUSZ MARECKI, JOHN P. LEWIS, MILIND TAMBE, AND PAUL SCERRI. **The DEFACTO System: Training Tool for Incident Commanders**. In MANUELA M. VELOSO AND SUBBARAO KAMBHAMPATI, editors, *AAAI*, pages 1555–1562. AAAI Press / The MIT Press, 2005. 56

[46] VIDHYA BALASUBRAMANIAN, DANIEL MASSAGUER, SHARAD MEHROTRA, AND NALINI VENKATASUBRAMANIAN. **DrillSim: A Simulation Framework for Emergency Response Drills**. In SHARAD MEHROTRA, DANIEL DAJUN ZENG, HSINCHUN CHEN, BHAVANI M. THURAISINGHAM, AND FEI-YUE WANG, editors, *ISI*, **3975** of *Lecture Notes in Computer Science*, pages 237–248. Springer, 2006. 56

[47] GIACOMO CABRI, FRANCESCO DE MOLA, AND RAFFAELE QUITADAMO. **Supporting a Territorial Emergency Scenario with Services and Agents: A Case Study Comparison**. In *WETICE*, pages 35–40, 2006. 57

[48] FRANCESCO DE MOLA, GIACOMO CABRI, NICOLA MURATORI, RAFFAELE QUITADAMO, AND FRANCO ZAMBONELLI. **The UbiMedic Framework to Support Medical Emergencies by Ubiquitous Computing**. *ITSSA*, **1**(1):15–26, 2006. 57

[49] ANAND S. RAO AND MICHAEL P. GEORGEFF. **BDI Agents: From Theory to Practice**. In VICTOR R. LESSER AND LES GASSER, editors, *ICMAS*, pages 312–319. The MIT Press, 1995. 61

[50] RAFFAELE QUITADAMO, DANILO ANSALONI, NIRANJAN SURI, KENNETH M. FORD, JAMES F. ALLEN, AND GIACOMO CABRI. **The PIM: an innovative robot coordination model based on Java thread migration**. In LUÍS VEIGA, VASCO AMARAL, R. NIGEL HORSPOOL, AND GIACOMO CABRI, editors, *PPPJ*, **347** of *ACM International Conference Proceeding Series*, pages 43–51. ACM, 2008. 93

[51] MAURIZIO LENZERINI. **Data Integration: A Theoretical Perspective**. In *PODS*, pages 233–246. ACM, 2002. 102

[52] MOSHÉ M. ZLOOF. **Query-by-Example: the Invocation and Definition of Tables and Forms**. In DOUGLAS S. KERR, editor, *VLDB*, pages 1–24. ACM, 1975. 102

[53] AKRIVI KATIFORI, CONSTANTIN HALATSIS, GEORGE LEPOURAS, COSTAS VASSILAKIS, AND EUGENIA G. GIANNOPOULOU. **Ontology visualization methods - a survey**. *ACM Comput. Surv.*, **39**(4), 2007. 102

[54] ROBERT ENDRE TARJAN. **Depth-First Search and Linear Graph Algorithms**. *SIAM J. Comput.*, **1**(2):146–160, 1972. 106, 107

[55] G. J. MINTY. **A Simple Algorithm for Listing all Trees of a Graph**. *IEEE Trans. on Circuit Theory*, 1965. 106, 107

[56] SANJIV KAPOOR AND H. RAMESH. **An Algorithm for Enumerating All Spanning Trees of a Directed Graph**. *Algorithmica*, **27**(2):120–130, 2000. 106, 107

[57] JEFFREY D. ULLMAN. *Principles of Database and Knowledge-Base Systems, Volume II*. Computer Science Press, 1989. 109

[58] SHELDON J. FINKELSTEIN. **Common Subexpression Analysis in Database Applications**. In MARIO SCHKOLNICK, editor, *SIGMOD Conference*, pages 235–245. ACM Press, 1982. 109

[59] TIMOS K. SELLIS. **Multiple-Query Optimization**. *ACM Trans. Database Syst.*, **13**(1):23–52, 1988. 109

[60] PRASAN ROY, S. SESHADRI, S. SUDARSHAN, AND SIDDHESH BHOBE. **Efficient and Extensible Algorithms for Multi Query Optimization**. In WEIDONG CHEN, JEFFREY F. NAUGHTON, AND PHILIP A. BERNSTEIN, editors, *SIGMOD Conference*, pages 249–260. ACM, 2000. 109

[61] YI LUO, XUEMIN LIN, WEI WANG, AND XIAOFANG ZHOU. **Spark: top-k keyword query in relational databases**. In *SIGMOD*, pages 115–126. ACM, 2007. 110

[62] BENNY KIMELFELD AND YEHOSHUA SAGIV. **Efficient Engines for Keyword Proximity Search**. In ANHAI DOAN, FRANK NEVEN, ROBERT MCCANN, AND GEERT JAN BEX, editors, *WebDB*, pages 67–72, 2005. 110, 112

[63] BENNY KIMELFELD AND YEHOSHUA SAGIV. **Finding and approximating top-k answers in keyword proximity search**. In STIJN VANSUMMEREN, editor, *PODS*, pages 173–182. ACM, 2006. 110, 112

[64] QI SU AND JENNIFER WIDOM. **Indexing Relational Database Content Offline for Efficient Keyword-Based Search**. In *IDEAS*, pages 297–306. IEEE Computer Society, 2005. 111

[65] GERARD SALTON, EDWARD A. FOX, AND HARRY WU. **Extended Boolean Information Retrieval**. *Commun. ACM*, **26**(11):1022–1036, 1983. 111, 112

[66] VAGELIS HRISTIDIS, LUIS GRAVANO, AND YANNIS PAPAKONSTANTINOU. **Efficient IR-Style Keyword Search over Relational Databases**. In *VLDB*, pages 850–861, 2003. 112

[67] KEVIN CHEN-CHUAN CHANG AND SEUNG WON HWANG. **Minimal probing: supporting expensive predicates for top-k queries**. In MICHAEL J. FRANKLIN, BONGKI MOON, AND ANASTASSIA AILAMAKI, editors, *SIGMOD Conference*, pages 346–357. ACM, 2002. 112

[68] RONALD FAGIN, AMNON LOTEM, AND MONI NAOR. **Optimal Aggregation Algorithms for Middleware**. In *PODS*. ACM, 2001. 112

[69] APOSTOL NATSEV, YUAN-CHI CHANG, JOHN R. SMITH, CHUNG-SHENG LI, AND JEFFREY SCOTT VITTER. **Supporting Incremental Join Queries on Ranked Inputs**. In PETER M. G. APERS, PAOLO ATZENI, STEFANO CERI, STEFANO PARABOSCHI, KOTAGIRI RAMAMOHANARAO, AND RICHARD T. SNODGRASS, editors, *VLDB*, pages 281–290. Morgan Kaufmann, 2001. 112

[70] EDWARD. J. MCCLUSKEY. **Logical Design Principles**, 1986. 114

[71] ALLAN M. COLLINS AND M. ROSS QUILLIAN. **Retrieval time from semantic memory**. pages 191–201, 1995. 116

[72] KAUSHIK CHAKRABARTI, SURAJIT CHAUDHURI, AND SEUNG WON HWANG. **Automatic Categorization of Query Results**. In Weikum et al. [79], pages 755–766. 116

[73] HANG CUI, JI-RONG WEN, JIAN-YUN NIE, AND WEI-YING MA. **Probabilistic query expansion using query logs**. In *WWW*, pages 325–332, 2002. 116

[74] GEORGIA KOUTRIKA AND YANNIS E. IOANNIDIS. **Personalized Queries under a Generalized Preference Model**. In *ICDE*, pages 841–852. IEEE Computer Society, 2005. 116

[75] PERIKLIS ANDRITSOS, RENÉE J. MILLER, AND PANAYIOTIS TSAPARAS. **Information-Theoretic Tools for Mining Database Structure from Large Data Sets**. In Weikum et al. [79], pages 731–742. 119

[76] PAUL BROWN AND PETER J. HAAS. **BHUNT: Automatic Discovery of Fuzzy Algebraic Constraints in Relational Data**. In *VLDB*, pages 668–679, 2003. 119

[77] YKÄ HUHTALA, JUHA KÄRKKÄINEN, PASI PORKKA, AND HANNU TOIVONEN. **TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies**. *Comput. J.*, **42**(2):100–111, 1999. 119

[78] BEI YU, GUOLIANG LI, KAREN R. SOLLINS, AND ANTHONY K. H. TUNG. **Effective keyword-based selection of relational databases**. In CHEE YONG CHAN, BENG CHIN OOI, AND AOYING ZHOU, editors, *SIGMOD Conference*, pages 139–150. ACM, 2007. 119

[79] GERHARD WEIKUM, ARND CHRISTIAN KÖNIG, AND STEFAN DESSLOCH, editors. *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*. ACM, 2004. 145

Declaration

I herewith declare that I have produced this work without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This thesis has not previously been presented in identical or similar form to any other Italian or foreign examination board.

The thesis work was conducted from January 2009 to December 2011 under the supervision of Prof. Letizia Leonardi and Francesco Guerra.

Modena, February 3, 2012