Travelers, there is no path, paths are made by walking.

– Antonio Machado.

University of Alberta

SociQL: A Query Language for the Social Web

by

Diego Fernando Serrano Suarez

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Examining Committee

To all my social network,
specially the first order zone.

Abstract

Social network sites are becoming increasingly popular and useful as well as relevant means for serious social research. However, despite their user appeal and wide adoption, the current generation of sites are hard to query and explore, offering limited views of local network neighbourhoods. Moreover these sites are disconnected islands of information due to application and interface differences. We describe SociQL: a query language along with a prototype implementation that enables for the representation, querying and exploration of disparate social networks. Unlike generic web query languages, SociQL is designed to support the examination of sociological questions, incorporating social theory and integration of networks that form a single unified source of information. The thesis discusses the design and rationale for the elements in the language, and reports on our experiences in querying real social network sites with it.

Acknowledgements

It is hard to write in two lines, what an experience of two years has stamped in my mind. And it is even harder when it allows me to thank people who taught me that knowledge is not transmitted from head to head, but from heart to heart. My supervisors, Eleni Stroulia and Denilson Barbosa.

Besides, I would also like to express my infinite gratitude to my ambassador of Kwan, Adriana Serrano, for being by my side all the time.

Thanks to my almost-perfect family for their emotional support.

Many thanks to the Department of Computing Science of University of Alberta for the generous financial support provided during my graduate studies.

And last but not least, to my dog, Luna. For being the best friend a man can have.

Table of Contents

List of Tables

List of Figures

List of Plates

...

List of Symbols

...

Chapter 1

Introduction

In this chapter we first explain the motivation and objectives of the work presented in this thesis. Then, we give a summary of the main features of the language and the system we have developed, and describe our contributions. Finally, we present an outline of the rest of the thesis.

1.1 Motivation and Background

In recent years, participation in social networking sites has increased dramatically, and similarly, online databases, such as DBpedia[1] and Freebase[2], have been emerging and collecting huge amounts of data representing relations between individuals. This increase in adoption was accompanied by a progressive diversification and specialization on the purpose and manner of use of such networks. Social tagging, blog contributions, instant messaging, events and fan clubs are just some of the different ways in which individuals interact in social networks.

In spite of the proliferation of social-network platforms, at a high level, social networks share a similar structure to model social interactions. Essentially, social networks are built around objects. A research network, for example, turns organizations, conferences and papers into objects of sociality among researchers. This observation is refered to as *Object-Centered Sociality* by Knor-Cetina [21], denoting the hypothesis that people are connected by a shared object, and is the reason why

[1]`http://dbpedia.org/`
[2]`http://www.freebase.com/`

some sociologists prefer to talk about “Socio-Material Networks”. This distinctive feature of social networks, allows the creation of an abstraction for modelling the record-based data across the various sites.

However, current online social-network sites and services are fairly “ego-centric”, focused on a respondent (ego), and a set of immediate neighbouring actors, thus hiding from the users most of the network. Furthermore, despite their similar underlying conceptual models, online social networks currently are silos of information, thus failing to take advantage of the synergies that could arise by sharing their data.

Under these premises, in this thesis we introduce SociQL, a query language designed from the ground up to answer relevant queries in social-network analysis as well as to provide a declarative means to integrate data from disparate networks.

1.2 The Research Problem

The benefits of linking social networks are invaluable. Users would be able to have their unique profiles exposed across sites and effortlessly become part of multiple communities. One could easily receive information from connections across platforms, and interests expressed in one community could easily sip through to others. This integration, in addition to increased information dissemination, could also lead to further community differentiation; instead of having similar information replicated in multiple places just to start participating, users could invest their efforts in taking advantage of the unique features of each platform, providing and accessing more platform-specific information and services.

Such possibilities are even more interesting and relevant in our area of interest, namely research networks. The ReaSoN (REseArcher SOcial Networks) project [28], developed by the Service Systems Research Group at the University of Alberta, studies the collaboration of researchers, focusing on computing scientists, with two long-term objectives. The first objective is to examine the co-authorship and citation relations among publications and authors, in order to understand typical research-collaboration patterns in the field. The second objective is to develop tools to support the established collaboration patterns and enable novel communication channels among authors, in order to advance the quality and productivity of collaborative research. Towards these goals, SociQL supports many of the abstractions necessary to represent the elements and relations within a typical social network. Moreover, SociQL is expressive and readable enough to enable querying and exploring the underlying social networks in ReaSoN in ad-hoc and complex ways with little effort.

Our goal with SociQL differs from previous works first, and foremost, because our design decisions are founded in sociological theories, as opposed to computer artifacts (e.g., files) that are used to represent networks. SociQL embraces the notion of *Object-Centered Sociality*, which implies that objects tend to displace human beings as relationship partners and embedding environments, or that they increasingly mediate human relationships, making the latter dependant on the former [21]. The design of the query language also recognizes that social actors coexist and interact in multiple overlapping networks simultaneously (e.g., two colleagues co-authoring papers and citing each other's work). Also, we seek to balance expresiveness and readability in the language, while keeping performance and query execution costs in mind. Finally, SociQL aims to exploit the importance of the actors in the networks when computing query results, as explained later in our model.

In this thesis, we discuss the design principles behind SociQL, based on our data model which is grounded on an object-centered theory of networks which is a very natural framework for the problem at hand. We discuss an implementation of SociQL over ReaSoN, which is also capable of exploiting links across platforms. We have experimented with it through several exploratory queries in our database, in concert with information available in other on-line resources, such as DBpedia[3] [7] and Facebook. Finally, we discuss the architecture of our current implementation and our query processing strategy. In a nutshell, we compile SociQL expressions into highly optimizable SQL (to be executed locally) and also in the REST APIs to fetch data from the external resources.

Although our experience with SociQL was encouraging that our paradigm might

[3]A queriable data source of facts extracted from Wikipedia.

be effective in helping users and social analysts search and explore large, interconnected social networks, a full usability study to confirm this hypothesis must be among the immediate future work.

1.3 Contributions

The original motivation of this work is to overcome the limitations observed in current query languages, namely, the lack of support for exploiting the structure and importance of actors in a social network. This led us to study web query languages and other academic and industrial approaches for structured query languages that can be applied to social networks.

Based on the theory of object-centered sociality, we define a model for social networks as a set of objects, relations, properties of objects and properties of relations, as described in Chapter 4. This simple model captures the semantics contained and extracted from social networks. Starting from this model, in Chapter 5, we use an SQL-like notation to define SociQL, our specialized query language for social networks. SociQL provides primitives for importance and path structure, which are essential in management and analysis of social networks. Additionally, the language incorporates the idea of querying a social network at different levels, by means of a layered architecture that enable aggregations and derivations of elements in the network. The language is introduced by examples that show the connection between sociological theories with the syntax and semantic supported by the language.

Furthermore, in Chapter 6, we present an implementation of SociQL on top of a conventional relational database system, which integrates external data sources, through REST APIs that access the external data in a declarative way. In order to interlink data across networks, we use a repository for identifiers, enabling the user to query several databases as a single integrated database.

Describing the implementation of SociQL, we also explain all the stages in the processing and execution of a query. During the first phase of the execution, the query is parsed and an internal representation of the query is created. Then, the system checks the syntactic validity, and if the query is valid, the query planner formulates an order of execution of the subqueries composing a SociQL query. Finally, during the execution phase, queries are sent to the proper data source and reified in the relational database, in order to translate the SociQL query into a single SQL statement.

Once we have executed a query, our implementation also offers a variety of options to visualize the results. We present a generic visualization in tabular form, in a similar way in which typical query languages show results. The map visualization shows the geographical position of objects in the result, as markers plotted in a map. Finally, graph visualization presents the information using nodes to represent objects, and links to represent relations, with the possibility of using this visualization as a network browser.

1.4 Outline

In the next chapter, we present a summary of the query languages and sociological theory related to our work. In Chapter 3, we introduce SociQL and its associated data model by means of an extensive series of examples. In Chapter 4, we formally define the data model and the relational algebra of the query language. In Chapter 5, we present the semantics of the language, including the special query features. In Chapter 6, we describe the implementation of SociQL over a relational database with links to external APIs. Finally, in Chapter 7, we present our conclusions and suggest possible directions of future work.

Chapter 2

Related Work

The work presented in this thesis is related to recently developed projects from diverse research areas such as Web query languages and social networks analysis. In fact, SociQL incorporates, generalizes and adapt ideas that are already present, although perhaps under different implementations and for a different purpose, on systems and languages from these projects. In the rest of this chapter, we present a summary of the most relevant work in query languages for web technologies and Social Networks.

2.1 Social Networks

The notion of social network and the methods of social network analysis have caught the attention and curiosity from various academic fields in recent decades. Along with the growing interest and increased use of network analysis has come to a consensus about the principles underlying the network perspective [47]. In this principles, we can identify two elements as being important: *actors*, representing autonomous social units, and *relational ties* between actors as channels for transfer or flow of resources.

Social network theories provide analysis of the causes of social correlation, e.g.: correlation between the behavior of affliated agents in a social network. Formally, this means that for two nodes u and v that are adjacent in graph G, the events in which u becomes active are correlated with v becoming active in such events. There are two primary (roughly categorized) explanations for social correlation: influence, and homophily [3].

Influence refers to the change induced by the social context to a person's behaviour. The influence phenomenon in a social network can be recognized as the tendency of a group of actors to exhibit similar behaviour to the source of influence. An example of this scenario is when an author uses a keyword because one of his/her colleagues/friends has recently adopted it. The importance of identifying influence as the type of correlation lies in its possible use for recognizing the creation of research paradigms and fads. Homophily is the tendency of individuals to associate and bond with similar others. Individuals in homophilic relationships share common characteristics (beliefs, values, behaviours, etc.) that make communication and relationship formation easier. Homophily also takes into account the external influence of elements in the environment, such as geography and family ties. The homophily hypothesis is straightforward for individuals [31]. At the individual level, persons are more likely to have a connection, friendship or association, if they have common attributes [12]. And while common norms are promoted through common attributes, so are common attributes likely when association or friendship occurs as a result of co-location and commonly situated activities [2].

2.2 Analysis and Importance metrics

The idea of importance was introduced by Bavelas [10] in 1948, when he hypothesized about the relation between group structure and centrality applied to human communication. Since then, graph theory researchers have been trying to identify the "most important" actors in a social network.

Over the years, many measures of centrality have been proposed, all of them attempting to define and measure properties of actor location in a social network. In this section, we will review the most noteworthy and substantively interesting definitions of importance, according to Wasserman [47]: *degree*, *closeness* and *betweenness*.

The simplest definition of actor centrality is *degree* centrality, which considers an actor as central depending on the number of ties to other actors in the network. Degree is often interpreted in terms of the immediate risk of an actor for catching whatever is flowing through the network, like a virus, or information. If the network is directed, then the definition of centrality may be separated in two measures of degree centrality, namely indegree and outdegree. *Indegree* is a count of the number of ties directed to the actor, and *outdegree* is the number of ties that the actor directs to others.

The second definition of actor centrality is focused on *closeness* or distance. The closeness measure is calculated based on how close an actor is to all the other actors in the network. The idea is that an actor is central if it can quickly interact with all others. In 1965, Beauchamp [11] noted that actors occupying central locations with respect to closeness can be very productive in communicating information to other actors. The simplest measure of closeness centrality was proposed by Sabidussi [43], who stated that closeness should be measured as a function of geodesic distances.

The third of the centrality measures is *betweenness*. This centrality measure recognizes that interactions between two nonadjacent actors depend on other actors in the network, and those other actors potentially might have some control over the interactions between the two nonadjacent actors. The idea is that an actor is central if it lies between other actors on their geodesics. The first attempt to quantify the betweenness centrality was made by Anthonisse [4] in 1971, suggesting that the locations of actors on the geodesics should be examined.

Finally, there is one more definition of centrality that has been adopted by current search engines, namely *eigenvector* centrality [15]. Eigenvector centrality is a measure of the importance of an actor in a network. It assigns relative scores to all actors in the network based on the principle that connections to high-scoring actors contribute more to the score of the actor in question than equal connections to low-scoring actors. Google's PageRank [40] is a variant of the Eigenvector centrality measure.

2.3 Frameworks for Social Network Analysis

Social network analysts sometimes use standard statistical procedures in examining and uncovering the ties that link social actors. Network analysis tools allow researchers to investigate representations of networks of different size. The various tools provide mathematical and statistical routines that can be applied to the network model. In this section, we provide a review of three programs, regarded by Carrington [20] as general and well known, such as UCINet, Pajek and NetMiner.

UCINet [16] is a program for the analysis of social networks and other proximity data. It is probably the best known and most frequently used software package for the analysis of network data and contains a large number of network analytic routines. UCINet is a menu-driven software that can read and write several kinds of text files, as well as Excel files, and it can handle a maximum of 32,767 nodes. Social network analysis methods include centrality measures, subgroup identification, role analysis, elementary graph theory, and permutation-based statistical analysis.

Pajek [9] is also a menu based software for network analysis and visualization, but specifically designed to handle large datasets (more than one million nodes). Among the main goals of this software are, first, to facilitate the reduction of a large network into several smaller networks that can be treated further using more sophisticated methods, and, second, to implement a selection of efficient network algorithms. Pajek network data can be defined inside the program or defined by a text file, which can also be in UCINet format. The software counts with a good number of descriptive methods for networks, for instance, computation of degrees, depths, cores or cliques, centrality, paths, structural holes and more.

Finally, NetMiner [24] is a software that combines social network analysis and visual exploration techniques. It allows users to explore network data visually and interactively, and helps to detect underlying patterns and structures of the network. It can handle a maximum of one million nodes. The data in NetMiner can be entered directly in a built-in matrix editor or by formatted files, like Excel datasheets, UCINet files, or NetMiner files. The network statistics available in NetMiner include methods to analyze the connection and neighborhood structure of the network and subgraph configurations, to calculate centrality measures and to analyze subgroup structures.

Although all these software tools provide advanced techniques for social analysis, they lack the flexibility of a declarative language to select and analyze portions of the network based on the data included in the nodes. Moreover, the representation of the network is restricted to nodes and links, without properties in the nodes or links, and also restrained only to uni-modal and two-modal networks, which subsequently means they do not include high level abstractions of the social network.

2.4 Representation Models for Communities

Various approaches for representing social networks have been presented. Considering the domain of ReaSoN, for instance, the network can be modeled as an amalgamation of a generic and a bibliographic social network, as the domain of knowledge. In the following we present several models, relevant to our domain of knowledge.

FOAF [18], the Friend of a Friend project that describes people, the links between them and the things they create and do. This model attempts to represent the persons and the interactions with documents and the multiple accounts linked to a person.

SIOC [14], the Semantically-Interlinked Online Communities initiative aims to enable the integration of online community information, such as forums. This model is built on top of FOAF, and among its distinctive features are the possibility to group interaction objects, for instance, posts are contained by forums.

Bibo [25], the Bibliographic Ontology describe bibliographic elements on the semantic Web, in order to be used as a citation ontology, as a document classification ontology, or simply as a way to describe any kind of document. The ontology also models the interactions between publications and events where publications are presented.

CERIF [19], the Common European Research Information Format is a formal model to setup research information systems and to enable their interoperation. The conceptual structure of the CERIF model is categorized into entity types: base, result and second level entities. The base entities represent the actors in a social network, such as persons and organizations. The result entities represent the output of the research, for instance publications or patents. The second level entities allow for the representation of the research context by linking to them from the base and result entities.

2.5 Web Query Languages

The problem of web query language design has been of long-term interest to the academic community and industry. Several research projects have investigated the idea of viewing the Web as a database that can be queried with a declarative language: WebSQL [38], WebOQL [5] and WebDB [36]. WebSQL's most salient features are its simple formal semantics and the powerful notation of path regular expressions for expressing graph searches. However, it assumes a rather inflexible data model, relying on a schema for restricted representation of web pages exclusively, that does not allow for the representation of the various types of objects and relations in social networks today. WebOQL provides a slightly more flexible model than WebSQL, giving support for exploiting the internal structure of documents, but as WebSQL, only supports unimode networks. WebDB allows access to document level information, intra-document structures (like tables and forms) and inter-document linkage.

These early Web query languages share two common features. First, they model the web as a graph over which queries can be expressed using familiar (essentially relational) constructs. Second, they provide special-purpose syntax for specifying how to traverse the Web graph, and produce node collections, using path expressions.

However, unlike our own SociQL, these Web query languages provide very little or no support for modeling the semantics of heterogeneous nodes and for social-network analyses, since their models adopt an impoverished view of the Web as a graph of interlinked documents.

With the emergence of the Semantic-Web research agenda, the Resource Description Framework [33] (RDF) has become the standard format for representing machine-readable information [13]. RDF databases can be viewed as labeled directed graphs, representing statements about resources in the form of subject-predicate-object expressions, where the object denotes the value of the subject's property predicate. Along with RDF, the W3C has recommended the declarative SPARQL [41] query language, which can be used to extract information from RDF graphs. SPARQL relies on a powerful graph-matching facility to bind variables to components in the input RDF graph, which is then matched against the RDF repository. However, SPARQL adopts a low-level abstraction for its underlying model, with very little domain knowledge; this is, in principle, undesirable for querying social networks since such generally expressive queries become quite difficult to optimize.

2.6 Social Network Query Languages

Recently, social-networking sites like Yahoo! and Facebook have started to support APIs, which are akin to simple application-specific query languages since they are meant to enable third-party systems to access their data. In 2007, Facebook introduced FQL, the Facebook Query Language [26]. FQL provides a way to query the same Facebook data that can be accessed through API functions, but with a SQL-style interface. The clauses are of the form select-from-where, with a single *from* table. Joins are not explicitly supported in the language, but a similar effect can be achieved, using composed queries or multiple queries. YQL, the Yahoo! Query Language [48], is grammatically similar to FQL, with additional support for the statements *show* and *desc* with which the user can request to access metadata from the source. Both languages, FQL and YQL, offer a rather restricted set of queries, in order to achieve good performance. Moreover, unlike the earlier web-query languages, FQL and YQL, are designed for ego-centric queries, in which a small portion of the networks, around a focal node, is visited during the query.

Finally, Ronen and Shmueli [42] proposed SoQL, a new language for querying and creating data in social networks. The language is designed to manage the increasing volumes of data, and includes advanced query features for social networks. SoQL identifies two basic structures: *path* and *group*. A path is an ordered set of network participants in which every consecutive two are friends, and a group is essentially a set of participants. The main element of a query is either a path or a group, with subpaths, subgroups and paths within a group defined in the query. The network model underlying SoQL is rather oversimplified, assuming unimodal networks and bidirectional relationships, which is not realistic in modern social networks. Moreover, there is no approach for queries across social networks.

2.7 Querying Linked Data

The amount of semantically structured data available on the Semantic Web has recently grown considerably, and projects such as DBpedia, Yago and LinkedMDB are providing harmonization of identifiers over vast amounts of large databases. Along with the data, the projects also give the user interfaces to post queries over the knowledge contained in the database.

DBpedia [7] is a community project that aims at extracting information from Wikipedia and make this information available on the Web as Linked Data[1]. Furthermore, over the years, an increasing number of data publishers have begun to set data-level links to DBpedia resources, making DBpedia a central interlinking hub for Web data. This project counts with two special query interfaces: DBpedia Query Builder[2] and Relationship Finder[3]. In the Query Builder, queries are expressed by means of a graph pattern consisting of multiple triple patterns, and for each triple pattern three form fields capture variables, identifiers of filters for the

[1]Linked Data describes a method of publishing structured data, so that it can be interlinked and become more useful

[2]`http://querybuilder.dbpedia.org/`

[3]`http://relfinder.dbpedia.org/`

subject, predicate and object of a triple, suggesting possible values for each field based on the information entered. The Relationship Finder presents a visual interface that allows the user to explore and find connections between two different entities in Wikipedia. The interface initially contains a simple form to enter two entities, and after submitting the query, the user can view the connections between two entities in a graph visualization.

Yago [44] is a semantic knowledge base containing information harvested from Wikipedia and linked to Wordnet. Currently, Yago provides a web interface similar to DBpedia Query Builder with form fields for the triples in the pattern.

Finally, the LinkedMDB (stands for Linked Movie DataBase) project [29] provides connection to several existing movie web resources. LinkedMDB as DBpedia and Yago provide access to SPARQL clients, where the user can pose queries in SPARQL format.

In SociQL, we also provide a way to ask queries in a SociQL client, with an interface similar to existing SPARQL clients, and we give the possibility of different views, like a map visualization, and alternative ways to browse the data, like in the graph visualization.

Chapter 3

SociQL by Examples

In this chapter, we illustrate SociQL for querying social-network data by reviewing a set of questions, regarded as important by sociologists, and showing how they can be expressed in our declarative language, SociQL.

Additionally, we provide an introduction to SociQL's data model and query language. However, this presentation is deliberately informal, in order to facilitate an intuitive understanding of the language. We give formal definitions in Chapter 4.

3.1 Sociological Questions in terms of SociQL

Our examples are based on a research social network that captures information about collaboration among computer science researchers. Throughout the chapter, we use the example social network illustrated in Figure 3.1 composed of the objects and relations in a small researchers' network. The network shows four authors with their publications (books, papers and posters) and affiliation to institutions. At the same time, the publications are linked to the conferences where they were presented and the keywords contained in the publication. In the figure, different icons of the nodes represent the variety of types of objects in the example network. In the objects, the name of the object represent the id, and the icon represent the label. For the relations, the ids are the ones of the relating objects and the label is shown next to the line.

The following examples illustrate typical questions that users and social scien-

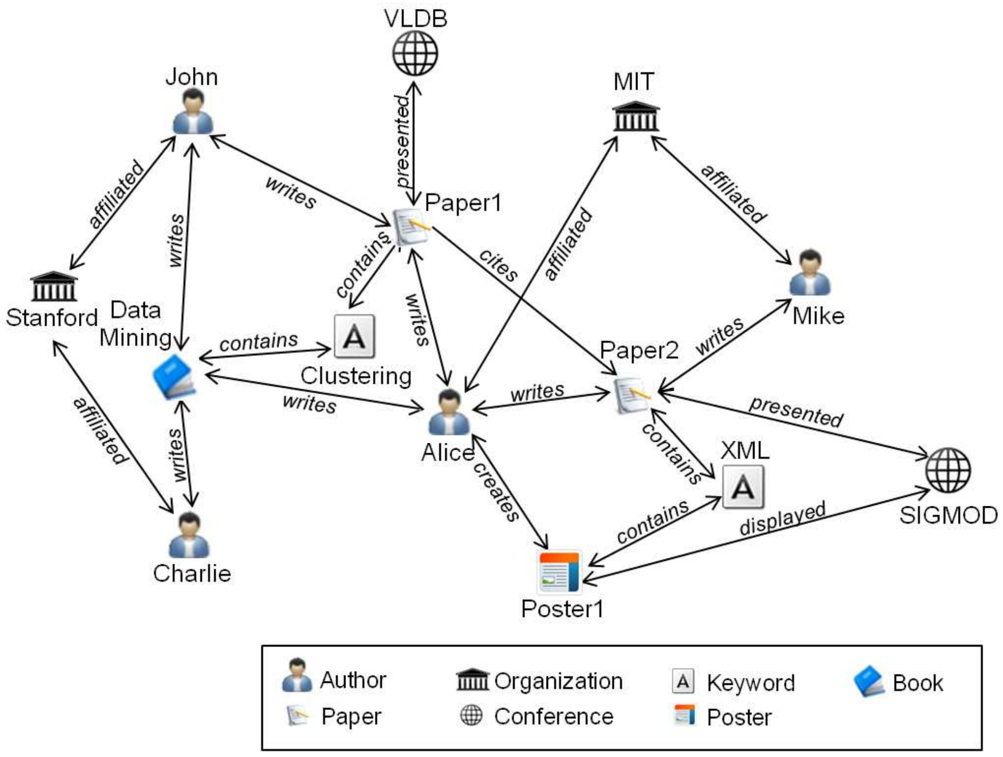


Figure 3.1: Example of a researcher social network.

tists may want to ask, while at the same time, highlight the limitations of current query languages, which motatived our work.

These examples demonstrate mainly two interesting innovations of SociQL. First is the fact that all *selections* return networks, in contrast with existing languages that return data in tabular form, regardless of the conceptual type of data they handle. This convention enables us to recursively nest *selections*. Once the desired network subset has been selected, a set of post-processing manipulations, expressed with additional syntax discussed in Section 6.7, can be applied to show the results as sorted tables, interactive graphs, or spatially mapped data. Second, it is important to note that some queries, such as queries involving centrality measures, cannot be easily expressed in existing query languages, since they do not incorporate the notion of importance as part of the language. In SociQL, however, these centrality measures are treated as primitives that can be used to filter networks.

3.1.1 Order Zones

In the 1960s, Milgram published the results of a now-famous experiment to estimate the minimum number of steps through which any two persons could be connected [39]. The results determined that the actual number of persons intervening are about five, hence the folkloric "six degrees of separation". The research was groundbreaking in that it suggested that human society is a small world type network characterized by short path lengths. The small world question is still a popular research topic today, with many experiments still being conducted [8, 34]. This notion of separation is represented in SociQL in terms of *order zones*. The region of nodes directly linked to a focal node is the first-order-zone; the nodes two steps removed from a focal node constitute the second-order-zone, and so on.

Thus in SociQL, we can retrieve the authors who have coauthored papers with a given author and the corresponding papers, as shown in the following example, which retrieves co-authors of researcher *Alice* and their papers which contain the word *"XML"* in the title:

Q1: `SELECT writes(r1, p1)`

```
FROM paper p1, author r1, author r2
     writes(r1, p1), writes(r2, p1)
WHERE r2.name='Alice' AND
     p1.title >< 'XML'
```

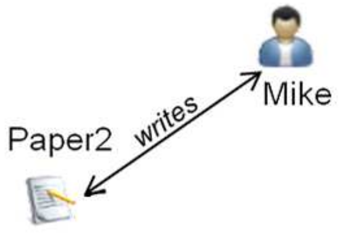


Figure 3.2: Result network for *Q1*.

The main construct provided by the query language is the familiar *select-from-where*. In the example, we use $r2$ to denote *Alice* as our focal actor, and then we form a network in the `FROM` with two researchers ($r1$ and $r2$) and a paper in common ($p1$). The result of the query is composed by the network formed by papers ($p1$) and coauthors ($r1$), linked by the *writes* relationship, after applying all the predicates in the `WHERE`. In other words, the previous query essentially retrieves a subset of the first-order zone of *Alice*. The result can be seen in Figure 3.2, presenting only Mike and Paper2 (assuming that Paper2 contains the word 'XML' in the title)

3.1.2 Influence

Consider the example in which John wants to find paths to authors with papers in SIGMOD, in order to analyze the structural cohesion of him with respect to the community of researchers that have published in such conference.

The above situation can be better approached taking into account social influence. Social influence refers to the impact in the behavior induced by a person's social context. In other words, influence can be defined as power, understanding power as the ability to influence other people's will. Anagnostopoulos [3] regards influence as one of the main causes for social correlation, and furthermore, many researchers are interested in the connection between social influence and behaviors, like technology adoption or consumer trends [46, 32].

While in principle there can be an infinite number of zones (third, fourth, fifth, ..., n), Kadushin stated that the influence of each zone on an individual node declines exponentially [31]. He also affirms that, for most purposes, the number of effectively consequential zones is between two and three; that is, whatever is being studied, individuals or objects, past the third zone, or at most the fourth zone, have relatively small effects on the focal individual or structure. Some experiments found that, for social events like loneliness, if a direct connection in your social network is lonely, you are 52 percent more likely to be lonely. At the second order zone (a friend of a friend) it is 25 percent. At the third order zone, someone who knows your friend's friend, it is 15 percent [23]. SociQL allows the discovery of zones of influence between two actors, using *path* relations. These relations define the two participating objects or individuals, a maximum number of objectual relationships necessary to connect objects, and optionally, the kind of participating relationships.

Returning to our initial problem, we assume John considers short paths (of length less than four) irrespective of the nature of their connections, since impact decreases exponentially and longer paths are assumed to have insignificant influence for the cohesion. Summarizing, John would like a path in which the length of the path is no longer than four, and the person at the end has a paper published in SIGMOD.

Q2:
```
SELECT PATH(r1,r2)
FROM author r1, author r2,
     conference c1, paper p1
     writes(r2,p1), presented(p1,c1),
     NEIGHBORHOOD(r1,r2,4)
WHERE r1.name='John' AND
      c1.name='SIGMOD'
```

Assuming the query is executed on the network shown in Figure 3.1, it will return the network depicted in Figure 3.3.

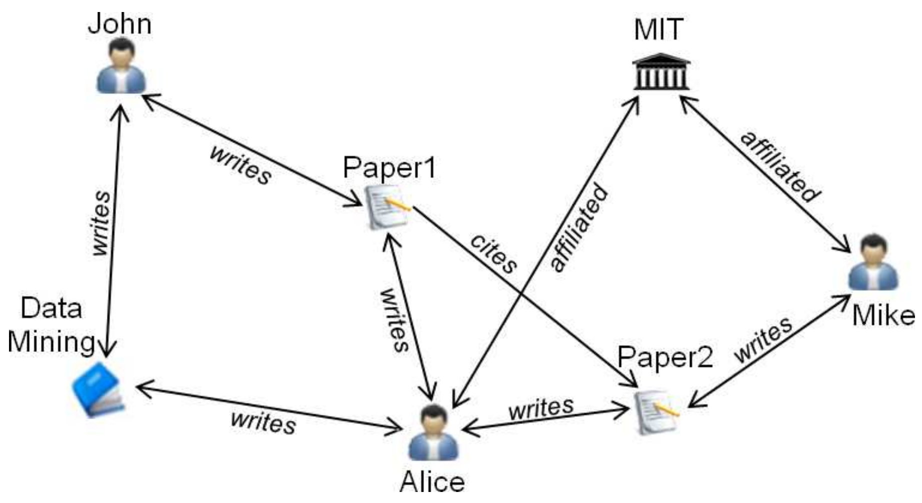


Figure 3.3: Result network for query in *Q2*.

The query employs two *author* objects $r1$ and $r2$ that should be related through a *path* predicate, where $r2$ writes paper $p1$ which is presented in conference $c1$. The *where* condition restricts the name of one of the *authors* to 'John' and the *conference* to'SIGMOD'. By default, the result of a path includes the objects and relations that form the path, connecting its two ends. The query, in this case, returns the network generated by the paths between the two researchers. The word `PATH` in the `SELECT` clause indicates that in the network will be included the set of elements (objects and relations) that connect the two authors and satisfies the conditions in the query.

3.1.3 Identification

Identification is a variety of social influence, where people are influenced by someone who is liked and respected. An example of an identification phenomenon is when a researcher writes about a particular topic because his colleagues are interested in the same topic. SociQL allows the inclusion of a `FILTER BY` clause that, as its name suggests, filters the results according to a centrality measure calculated on the graph pattern being queried.

Consider the following example in which John wants to work on Clustering, but first, he wants to know what other researchers, also interested in Clustering, have been doing, especially the ones with the most important papers. The "importance" in this case will be based in the importance of the citing papers (using the Pagerank score), and the number of citations.

```
Q3: SELECT writes(r1, p1)
    FROM
      (SELECT writes(r1, p1), cites(p2, p1)
      FROM author r1, paper p1, paper p2,
           keyword k1, writes(r1, p1),
           cites(p2, p1), contains(p1, k1)
      WHERE k1.keyword='Clustering'
      FILTER BY (PAGERANK OF p1 ON cites) > 0.8)
    FILTER BY (INDEGREE OF p1 ON cites) > 100
```

This example demonstrates the composability of SociQL queries. A `SELECT` query returns a network that, as can be seen in the query, can go in the `FROM` of another query. In the above query, the inner query returns a network of authors, papers and citing papers, containing the papers about Clustering with a Pagerank score greater than 0.8. The outer query takes this result and returns only the network of authors and papers where the paper has more than 100 citations, fulfilling our requisites for 'importance' in this example.

Moreover, it is important to note that, the previous example cannot be easily expressed in other existing query languages, since they are not designed to handle explicitly the centrality measures in a network.

3.1.4 Genealogies

In addition to discovering relations between actors, the *path* relations can also be used to find genealogies. The path expressions allow the specification of relation types that will restrict the kind of paths. Consider a network that specifies the *supervises* relationship, and a query in which we want to get the research genealogy with John as focal actor. The query can be written as follows.

```
Q4: SELECT PATH(r1,r2)
    FROM author r1, author a2,
         NEIGHBORHOOD(r1,r2,6,supervises)
    WHERE r1.name='John'
```

The research genealogy is defined by the *supervises* relation. In the result, we get the network of authors supervised directly or indirectly by John up to the sixth order zone.

3.1.5 Homophily

The objectual relationships can also define certain expected behaviour between individuals in the network, motivated by homophily, the tendency of individuals to associate and bond with similar others. Anagnostopoulos [3] also regards homophily as one of the main causes for social correlation. In our model, we take objects and properties as the elements that define the similarity of a given individual.

In the following example, we get the network of papers and authors who write about *XML* and have been affiliated to *MIT*. The assumption in this case is that there is a high probability that the authors in the result set may be associated, since they share some similarities (interests and affiliation).

```
Q5: SELECT writes(r1.p1)
    FROM paper p1, author r1,
         keyword k1, organization o1,
         affiliated(r1,o1), writes(r1.p1),
         contains(p1,k1)
    WHERE o1.name='MIT'
         AND k1.keyword='XML'
```

The result is shown in Figure 3.4, presenting Alice and Mike as the authors affiliated to MIT and the paper they wrote about 'XML'.

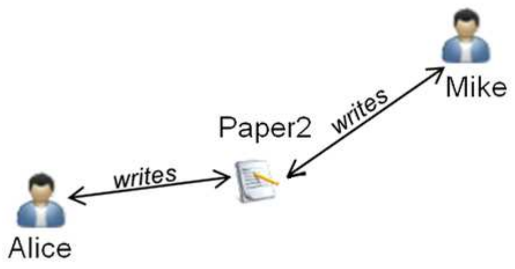


Figure 3.4: Result network for *Q5*.

3.1.6 Propinquity

Geographical proximity or co-location is perceived as a fundamental factor in sociological phenomena. Aronson et al. [6] highlight the effectiveness of the propinquity effect, noting the “mere exposure” effect, where being exposed to a person affects our liking for them since the more we see the person, even though there is no communication, the more familiar they become and thus the more we like them. According to the propinquity principle, at all levels of analysis, nodes are more likely to be connected with one another, other conditions being equal, if they are geographically near to one another [37].

The SociQL system identifies the properties specifying location as special properties. When the location properties are available, we can visualize the result of a query through an interface that includes a map visualization, which provides the geographic location of the individual or object. Nevertheless, SociQL does not provide functionality to calculate distances between the objects contained in the result. To build a map query, the line containing the projected properties must be modified. Basically, the keyword `SELECT` must be replaced with `MAP`, following the name of the object and the properties that the marker will contain, like in the following query.

```
Q6: MAP r2:name, url
    FROM paper p1, author r1, author r2
         writes(r1,p1), writes(r2,p1)
    WHERE r1.name='Alice'
```

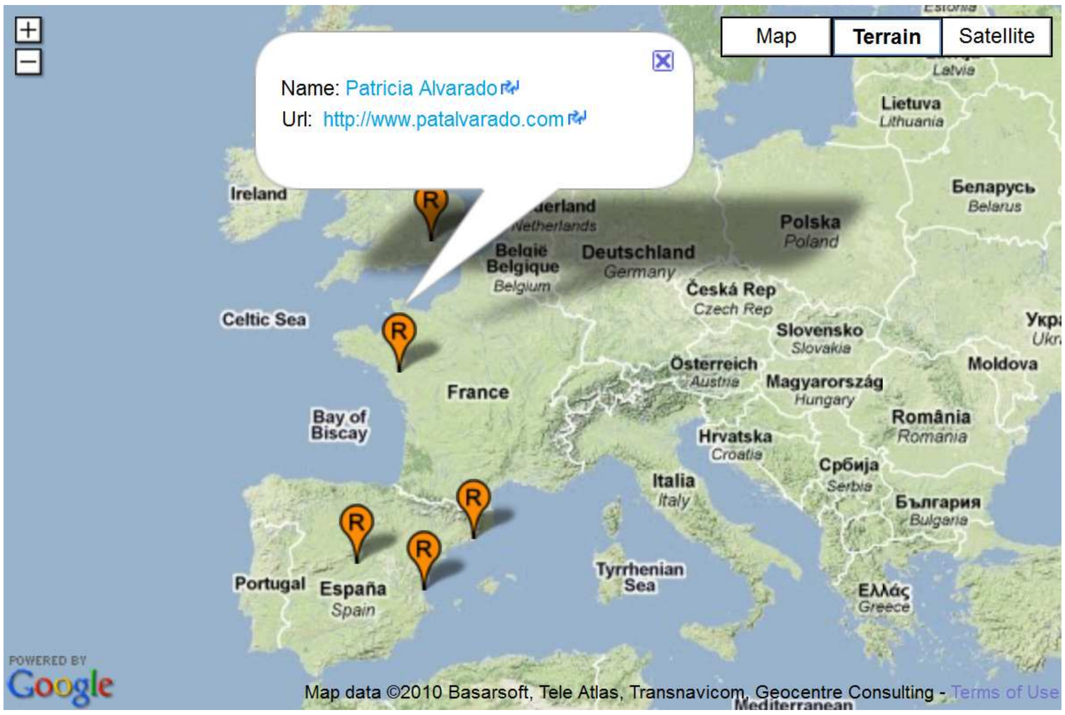


Figure 3.5: Result of Map query in *Q5*.

The result of *Q6* is shown in Figure 3.5, which shows the geographical distribution of coauthors of Alice.

3.2 SociQL for Systems Integration

Let us now revisit our original motivation for interoperation of social networking platforms. SociQL queries can also be augmented using external sources as Facebook or DBpedia. An extensive experimental literature in social psychology established that attitude, abilities, belief and value similarity lead to attraction and interaction. Homophily on traits like intelligence was one of the first phenomena studied in the early network literature [37]. So, it is highly probable, for example, that among the friends in Facebook, the user has added people working in similar topics as him. In this way, listing the papers of friends in Facebook could return a network of papers relevant to the interests of the user, due to the homophily phenomena. This query is shown in Q7.

Q7:
```
SELECT writes(r1,p1), sameAsUser(r1,u1)
FROM paper p1, author r1, user u1
     writes(r1,p1),
     sameAsUser(r1,u1)
WHERE p1.year=2009
```

3.3 The Social Network Model

In this section, we provide the theoretical foundations for our SociQL language. Starting from the graph structure of social networks mentioned in the Introduction, we define the necessary graph elements to construct a query language that provide easy and effective access to social network analysis functionalities.

3.3.1 Objectual Social Networks

The social network perspective encompasses theories, models and applications that are expressed in terms of relational concepts [47]. In other words, relations defined by linkages among units are fundamental component of network theories. Those studies led to the identification of important concepts:

- *Actors* and their actions are viewed as interdependant rather than independent, autonomous.
- *Relational ties* between actors are channels for transfer of resources.

In 1997, Knorr-Cetina begins to develop an analysis of *objectualization* in social networks [21]. In simple words, the thesis of objectualization imply that specific objects substitute and become constitutive of social relations. Essentially, while recognizing the social interaction between individuals, this theory exalts the role of specific objects as the reason why social actors affiliate with each other. Knorr-Cetina also states that this objectualization is caused by the dispersion of knowledge processes and knowledge structures in social life, since now the knowledge processes are heavily centered on objects of knowledge. Considering the objectualization, we define SociQL around the concept of an **object**. For example, in the context of ReaSoN, we have that a paper (an object) mediates the relation among co-authors; similarly, a publication venue (an object) connects authors who publish their work in it.

Objects are linked to one another by **relationships**. The defining feature of a relationship is that it establishes a linkage between a pair of objects. The type and nature of the relationships may vary, ranging from affiliations to behavioral interactions.

In our model, both, objects and relationships may be described by **properties** (actual data), such as the name of an author or the date in which an author affiliates to an organization. We also distinguish the **context** in which objects are circumscribed to contain properties and relationships. For instance, an author might return different email addresses for the same individual depending on the context in which the query is asked (professional or personal). In practice, each context will correspond to different social network systems, thus, each context may have its unique data access methods and privacy restrictions, which complicates query processing to a great extent.

3.3.2 Interlinks across Social Networks

Current online social networks are 'islands' of information, where each one has their own data that eventually could be complementary to some data in other 'island'. As more and more social communities emerge, the lack of interoperation seems more evident.

In order to interlink the different communities, the different online social networks describing an entity or resource must refer to the entity with a consistent identifier. However, the developers of such social networks are using identifiers only valid in a particular context, which results in a proliferation of identifiers that prevents the merging of social networks.

In 2007, Bortoli et al. [17] introduce the *Pirandello's identity problem* as a metaphor to explain the identifiers problem. Luigi Pirandello was an Italian novelist awarded the Nobel Prize in literature in 1934. In one of his novels, "One, no one and one hundred thousand", the protagonist discovers that everyone he knows, everyone he has ever met, has constructed an image of him in their own imagination and that none of these persons corresponds to the image that he himself has constructed and believes himself to be.

The identity problem in today's social networking sites is inevitable. Every site uses only a local identifier, which certainly promoted the proliferation of different identifiers across the social networks and has contributed to the construction of different 'images' of the same resource. As it turns out, this problem is extremely hard to solve in practice. In order to correctly interlink the different communities, different social network sites describing the same object would have to refer to it with a globally consistent identifier. In practice, however, each site has its own local identifier, unique only in its particular context. In practical terms, the main approaches for linking data across networks include approximate joins and social tagging, but we will return to this discussion later when presenting our current implementation.

Chapter 4

The SociQL Data Model

In this chaper, we introduce a formal foundation for SociQL. Starting from the graph structure for social networks presented in the previous chapter, we formally define the notion of social network and layered architecture, in order to construct a query language.

4.1 Data Model

In designing SociQL, we have developed a simple model for capturing the semantics of the information contained in, and extracted from, social networks. We define a "social network" as a 4-tuple (O, R, P_O, P_R). The object set O is a set of **social objects**, in what we call socio-material networks[1], for example, *authors* or *papers*. The **relation** set R is a set of links between the objects in O that represent the flow of information or materials, like the relation *writes* or *affiliates*. Every object and relation have **properties** that define them, represented by P_O and P_R respectively, such as the *title* and the *year-of-publication* for a *paper*, or the *starting-year* in the *affiliated-with* relation. Every property, be it an object property or a relation property, contains a *value* of a given **type** (e.g., integer, Boolean and string). In our setting, values are objects whose associated interpretations are universally agreed upon in the application domain. We define the types as a finite non empty set T; and the domain D associated to each type t of T, as $dom(t) \subseteq D$.

[1]Some sociologists prefer to talk about "Socio-Material Networks", given the hypothesis that people are connected by a shared object.

In contrast to values, social *objects* represent real or conceptual objects in the world, which are defined in terms of their properties. In the representation of objects, we distinguish a type $Oid \in T$ of object identifiers. We assume an infinite set O-dom of Oids, and a disjoint set V-dom of T. The object can be formally defined as a tuple:

$$O = \ [id : Oid, label : t]$$

where id is a unique identifier for the object, with a $label$ of type t identifying the type of object.

Each object is characterized by a set of properties, depending on its type. The properties of the objects P_O are defined through the object identifier of the associated object, as follows:

$$P_O = \ [id : Oid, label : t_1, value : t_2]$$

where id represents the object id, $label$ describes the property name and $value$ contains the value of the property.

Similarly, we can define the relations as:

$$R = \ [id1 : Oid, id2 : Oid, label : t]$$

where $id1$ and $id2$ are object identifiers that represent the origin and the end of the relation link, and $label$ identifies the type of the relation.

Finally, the properties of relations P_R are defined through the two object ids that uniquely identify a relation instance:

$$P_R = \ [id1 : Oid, id2 : Oid, label : t_1, value : t_2]$$

where $id1$ and $id2$ together represent the relation instance, $label$ describes the property name and $value$ contains the value of the property, just as in the object definition.

We will illustrate the representation of the data model, using the example network shown in Figure 4.1. We will assume a relational database schema, based on the data model, as shown in Figure 4.2. The tuples of Objects and Relations are

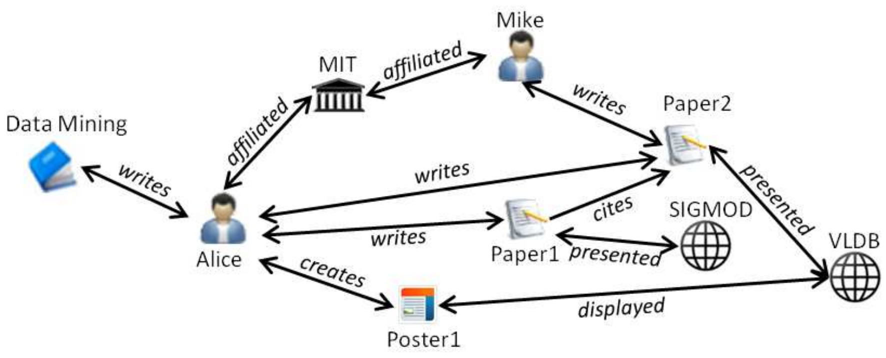


Figure 4.1: Simplified example of a research social network.

O

<u>id</u>	label

R

<u>id1</u>	<u>id2</u>	label

PO

<u>id</u>	<u>label</u>	value

PR

<u>id1</u>	<u>id2</u>	<u>label</u>	value

Figure 4.2: Schema model for a social network. The underlined attributes represent primary keys.

defined as:

$$\begin{aligned}
O = \{&[\text{‘}\mathit{DataMining}\text{’}, \text{‘}\mathit{Book}\text{’}], [\text{‘}\mathit{MIT}\text{’}, \text{‘}\mathit{Organization}\text{’}], [\text{‘}\mathit{Alice}\text{’}, \text{‘}\mathit{Author}\text{’}],\\
&[\text{‘}\mathit{Mike}\text{’}, \text{‘}\mathit{Author}\text{’}], [\text{‘}\mathit{VLDB}\text{’}, \text{‘}\mathit{Conference}\text{’}], [\text{‘}\mathit{SIGMOD}\text{’}, \text{‘}\mathit{Conference}\text{’}],\\
&[\text{‘}\mathit{Poster1}\text{’}, \text{‘}\mathit{Poster}\text{’}], [\text{‘}\mathit{Paper1}\text{’}, \text{‘}\mathit{Paper}\text{’}], [\text{‘}\mathit{Paper2}\text{’}, \text{‘}\mathit{Paper}\text{’}]\}\\
R = \{&[\text{‘}\mathit{Alice}\text{’}, \text{‘}\mathit{DataMining}\text{’}, \text{‘}\mathit{writes}\text{’}], [\text{‘}\mathit{Alice}\text{’}, \text{‘}\mathit{MIT}\text{’}, \text{‘}\mathit{affiliated}\text{’}],\\
&[\text{‘}\mathit{Mike}\text{’}, \text{‘}\mathit{MIT}\text{’}, \text{‘}\mathit{affiliated}\text{’}], [\text{‘}\mathit{Alice}\text{’}, \text{‘}\mathit{Paper1}\text{’}, \text{‘}\mathit{writes}\text{’}],\\
&[\text{‘}\mathit{Alice}\text{’}, \text{‘}\mathit{Paper2}\text{’}, \text{‘}\mathit{writes}\text{’}], [\text{‘}\mathit{Mike}\text{’}, \text{‘}\mathit{Paper2}\text{’}, \text{‘}\mathit{writes}\text{’}],\\
&[\text{‘}\mathit{Alice}\text{’}, \text{‘}\mathit{Poster1}\text{’}, \text{‘}\mathit{creates}\text{’}], [\text{‘}\mathit{Paper1}\text{’}, \text{‘}\mathit{SIGMOD}\text{’}, \text{‘}\mathit{presented}\text{’}],\\
&[\text{‘}\mathit{Paper2}\text{’}, \text{‘}\mathit{VLDB}\text{’}, \text{‘}\mathit{presented}\text{’}], [\text{‘}\mathit{Poster1}\text{’}, \text{‘}\mathit{VLDB}\text{’}, \text{‘}\mathit{displayed}\text{’}],\\
&[\text{‘}\mathit{Paper1}\text{’}, \text{‘}\mathit{Paper2}\text{’}, \text{‘}\mathit{cites}\text{’}]\}
\end{aligned}$$

And with the only purpose of exemplify the representation of properties, we will assume that *Papers* have *title*, *year* and *pages* (for number of the pages) as the properties of the object, and similarly, *Books* and *Poster* will have only *title* and *year* for their properties.

$$\begin{aligned}PO = \{&[\text{‘}\mathit{Paper1}\text{’}, \text{‘}\mathit{title}\text{’}, \text{‘}\textit{Constrained locally weighted clustering}\text{’}],\\ &[\text{‘}\mathit{Paper1}\text{’}, \text{‘}\mathit{year}\text{’}, \text{‘}\mathit{2008}\text{’}], [\text{‘}\mathit{Paper1}\text{’}, \text{‘}\mathit{pages}\text{’}, \text{‘}\mathit{90-101}\text{’}],\\ &[\text{‘}\mathit{Paper2}\text{’}, \text{‘}\mathit{title}\text{’}, \text{‘}\textit{Structured Search Result Differentiation}\text{’}],\\ &[\text{‘}\mathit{Paper2}\text{’}, \text{‘}\mathit{year}\text{’}, \text{‘}\mathit{2009}\text{’}], [\text{‘}\mathit{Paper2}\text{’}, \text{‘}\mathit{pages}\text{’}, \text{‘}\mathit{313-324}\text{’}],\\ &[\text{‘}\mathit{DataMining}\text{’}, \text{‘}\mathit{title}\text{’}, \text{‘}\textit{Data Mining}\text{’}], [\text{‘}\mathit{DataMining}\text{’}, \text{‘}\mathit{year}\text{’}, \text{‘}\mathit{2007}\text{’}],\\ &[\text{‘}\mathit{Poster1}\text{’}, \text{‘}\mathit{title}\text{’}, \text{‘}\textit{NN-Queries over Dynamic Graph Data}\text{’}],\\ &[\text{‘}\mathit{Poster1}\text{’}, \text{‘}\mathit{year}\text{’}, \text{‘}\mathit{2008}\text{’}], ...\}\end{aligned}$$

For properties of relations, we will use *start_year* in the *affiliated* relation.

$$\begin{aligned}PR = \{&[\text{‘}\mathit{John}\text{’}, \text{‘}\mathit{Stanford}\text{’}, \text{‘}\mathit{start\_year}\text{’}, \text{‘}\mathit{1999}\text{’}],\\ &[\text{‘}\mathit{Charlie}\text{’}, \text{‘}\mathit{Stanford}\text{’}, \text{‘}\mathit{start\_year}\text{’}, \text{‘}\mathit{1999}\text{’}],\\ &[\text{‘}\mathit{Alice}\text{’}, \text{‘}\mathit{MIT}\text{’}, \text{‘}\mathit{start\_year}\text{’}, \text{‘}\mathit{2003}\text{’}],\\ &[\text{‘}\mathit{Mike}\text{’}, \text{‘}\mathit{MIT}\text{’}, \text{‘}\mathit{start\_year}\text{’}, \text{‘}\mathit{1995}\text{’}], ...\}\end{aligned}$$

In order to interlink the different *contexts*, the different online social networks describing an entity or resource must refer to it with a consistent identifier. In our model, we assume a central repository for object identifiers. This repository can be imagined like a thesaurus that links different object identifiers across networks. This repository can be modeled as a function mapping.

$$f : Oid \times targetContext \rightarrow Oid$$

This function takes the object id and the target context, and returns the object id in the target context. The details on how this repository can be implemented will be discussed in the following chapters.

4.2 Three-Layered Architecture

Social-network analysis is interested in examining different types of questions, sometimes inspecting specific types of relations and others aggregating relations focusing on general issues of "connectivity". Jung and Euzenat [30] introduce what they call semantic social network, a structure made of three superimposed networks that are assumed to be strongly linked:

Data layer explicitly relating instances of actors and objects at the lowest level of abstraction;

Concept layer relating domain (in our case, research network) concepts on the basis of explicit domain relations and implicit relations, derived based or our adoption of the socio-material networks view; and

Network layer relating networks on the basis of explicit high-level relationships, assumed to be common across all social networks.

In SociQL, we adapt the ideas of semantic social networks, and apply them to our query language. Figure 4.3 shows a graphic representation of the multi-layer architecture in SociQL, however, in order to improve readability, only a portion of the Concept layer is shown in the figure and the mappings between relations are not shown. In the figure, the orange dashed arrows represent the mapping between the data and concept layer, the red arrows represent the derived relationships within the concept layer, and the purple dotted lines depict the mapping between the concept and network layer. The characteristics of each adapted layer and the relationships between layers are described below.

4.2.1 Data layer

In this layer, nodes represent objects at the lowest level, like *authors*, *conferences* or *papers*, and relations are the explicit connections between the objects, like *writes*, as defined in the data model.

In our example of research social networks, consider a query in which we want to get the papers written by John. In this case, we are asking for elements at the data

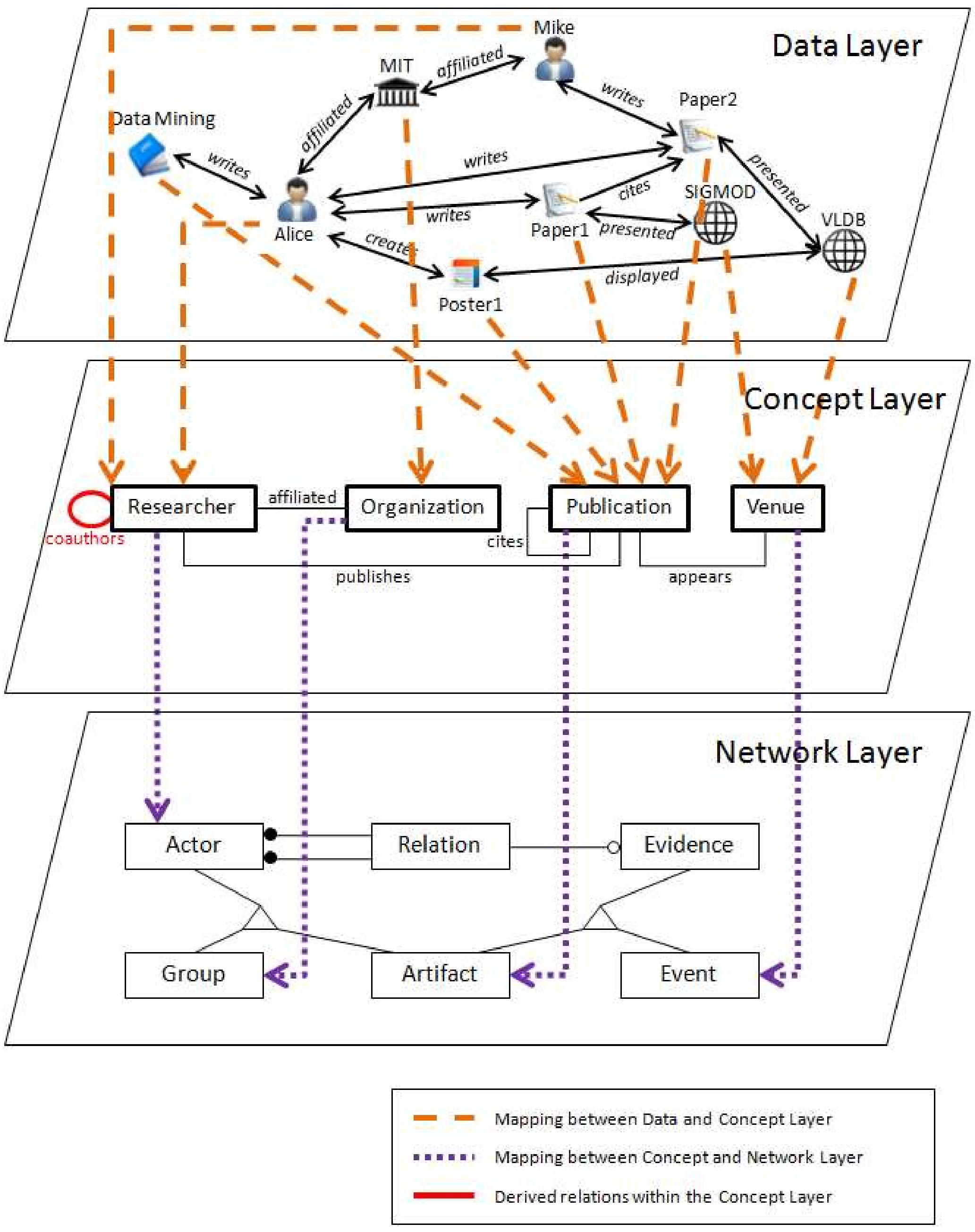


Figure 4.3: Three layer architecture in SociQL.

layer (*author* and *paper*), which can be expresed in the following SociQL query:

Q8:
```
SELECT writes(r1,p1)
FROM author r1, paper p1, writes(r1,p1)
WHERE a1.name='John'
```

As can be seen, the query only uses elements declared in the model at the lowest level. We will continue expanding this example, to demonstrate how, using SociQL, one can ask more general queries (at higher levels of abstraction).

4.2.2 Concept layer

This layer provides a level of abstraction at which to define a domain of knowledge. Basically, this layer deals with issues concerning the grouping of entities (or classes), related within a hierarchy, and subdivided according to differences and similarities encountered between the entities within the domain of knowledge.

This layer also allows the definition of derived relationships. As stated by the objectualization theory, many relations are mediated by an object, but sometimes it is convenient to abstract the relation and obviate the object in between, such as in the coauthorship relation that associates coauthors but without defining explicitly the paper in common.

The materialization of a derived relation $R(A, B)$ is defined as a relation between two objects such that:

- there is a finite path $o_0, r_1, ..., r_n, o_n$, where $r_1, ..., r_n$ represent the relations and $o_1, ..., o_n$ denote the objects.
- for each i, there is a relation r_i between o_{i-1} and o_i
- $o_0 = A$ and $o_n = B$

In other words, a derived relationship is the representation of a path that hides the complexity, simplifying multiple relations into a single virtual relation.

The concept network C is a network $\langle O_C, R_C \rangle$, in which O_C is a set of entities and $R_C \subseteq O_C \times O_C$ the relationships between the entities. There can be two main kinds of relationships in this network.

Domain-specific associations are used to store specific kinds of facts or to answer particular types of questions. For example, in the domain of bibliographic information, we might need a *publishes* type of relationship which tells us that a publication is published by certain researcher(s).

Derived relations, as described before, simplify multiple relations into a single virtual relation.

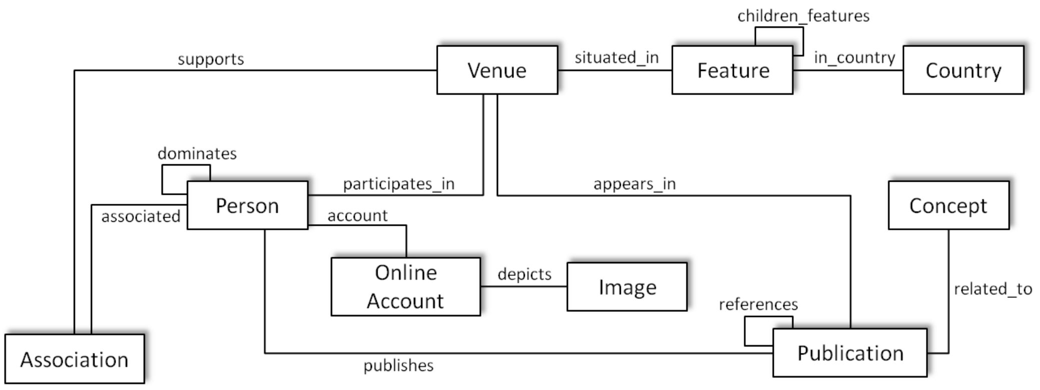


Figure 4.4: Concept Layer.

Figure 4.4 shows the entities as well as their recursive and linking relationhips. The core entities are *Person* and *Association* which allow the representation of scientific actors and their different kind of interactions, including the *Associates* relation, linking a *Person* to an *Association*. Another important entity is *Publication*, which represents the result or research output. This entity exhibits the typical behavior of a publication, linking to *Concepts* or topics they are related to, linking to other publications for citations or *References*, and relating the *Persons* who authored it. The entity *Venue* groups together the previously presented entities around an event, in order to relate the *Association* that *Sponsors* a *Venue*, and also links the *Persons* and *Publications* that *Participates* or *Appear*, respectively, in a *Venue*. Finally, the entities *Country* and *Features* are used to define geographic location, and *OnlineAccount* and *Image* represent the different accounts for social networks that a *Person* can have in this context.

In the example presented for the Data layer, we make a query in order to obtain the papers written by John. For the Concept layer, we make a generalization of the query. In this case, we want to get all the publications written by John, in order to evaluate his academic productivity in general. The term “publications” in this case refers to papers, books and posters. In this query, we are asking for elements at the concept layer (*individual* and *publication*), which will be translated into the following SociQL query:

Q9:
```
SELECT CON.PUBLISHES(r1,p1)
FROM CON.PERSON r1, CON.PUBLICATION p1,
     CON.PUBLISHES(r1,p1)
WHERE r1.name='John'
```
[2]

This query is translated into an aggregation of data objects, simulating the effects of a view in the social network. Further details on the translation will be discussed in the following sections.

4.2.3 Network layer

In this layer, objects and relations are interpreted at the highest level of abstraction. The network defines the basic elements that compose a social network and its interactions.

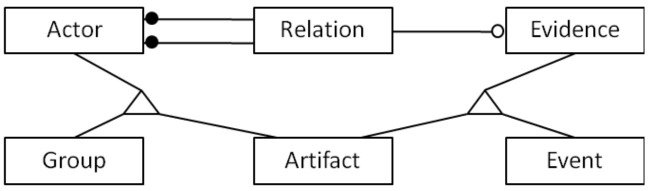


Figure 4.5: Network Layer.

The design of the network layer is based in the notion of “objectualization”, distinguishing between active and passive objects. Figure 4.5 shows a diagrammatic representation of the SociQL’s network layer. Active objects, called *Actors*, are the

[2]Note that the prefix “CON.” and “NET” in the query elements defines the scope of the query as the Concept and Network layer respectively.

objects which have the power of acting, causing change, motion or communication. Passive objects, called *Evidence*, are the ones that receive the actions from the Actors, and mediate the relations between active objects. Actors, at the same time, may be specialized in *Groups* and *Artifacts* (denoted by the triangle in the relation), where groups aggregate individual actors acting as a single social unit, for instance, a project team. Passive objects or Evidence, in turn, are specialized in *Artifacts* and *Events*. Events refer to an occurrence happening at a determinable time and place, with or without the participation of Actors. As can be seen in the diagram, *Artifacts* may be *Actor*s or *Evidence*. The difference lies in the way the Artifact participates in the relation, in other words, if the Artifact is merely a mediator, then it is an Evidence, otherwise, it is an Actor. In this definition, the term Artifacts encompasses all the objects produced by manual or intellectual labor.

The general purpose *Relation* between the network elements represent the basic interactions in a social network, expressed at an abstract level. Most of the relations are mediated by an evidence, reinforcing our notion of objectualization, but the relation can also obviate the mediator, in which case, the relation is constituted by two Actors. A social network will always have relations between Actors (represented by the filled circles), and optionally, relations between Actors and Evidence (represented by the empty cirle).

In the example presented previously, in the Concept Layer section, we make a query in order to obtain the publications written by John. In this case, we are going to relax the query in order to analyze only John's associations to objects or artifacts in general. The results of associations in this example comprises papers, books, posters and organizations. In this query, we are asking for elements at the network layer (*actor* and *artifact*), which will be translated into the following SociQL query:

Q10:
```
SELECT NET.RELATION(r1,p1)
FROM NET.ACTOR r1, NET.ARTIFACT p1,
     NET.RELATION(r1,p1)
WHERE r1.name='John'
```

It is important to note that if we include more contexts in our network, such as Facebook, then we would be able to include other objects, for instance status messages and photo albums.

Similarly to queries at the concept layer, this query is translated into an aggregation of concept objects, and recursively to data objects, simulating the effects of a view in the social network. Further details on the translation will be discussed in the following sections.

4.3 Mappings between Layers

The mapping between elements in the data layer and the concept layer is established by semantic links between a O_D and a O_C for objects, and R_D and R_C for relations[3]. One O_C object may correspond to many O_D objects, and one R_C relation may correspond to many R_D objects. Objects and relations in C can be considered as aggregations of objects and relations in D, or in other words, a mapping that aggregates and abstracts multiple elements. Similarly, the mapping between elements in the network layer and the concept layer is established by semantic links between O_C and O_N for objects, and R_C and R_N for relations. One O_N object may correspond to many O_C objects, and one R_N relation may correspond to many R_C objects. In the same way, objects and relations in N can be considered as aggregations of objects and relations in C.

In the following sections, we will illustrate the mapping for the example presented in Figure 4.3, using Datalog [1] queries.

4.3.1 Data to Concept Layer

The following mappings generate the entities *Person*, *Association* and *Venue* respectively, using the objects declared as a relation with an id and a label denoting the type of object. In the first case, *Person* is formed only by the objects of type *Author*; in the second case, *Association* is constituted by objects of type *Organization*; and in the third case, *Event* is made of objects with label *Conference*.

[3]The subscripts D, C and N denote the Data, Concept and Network layer respectively

$O(id, \text{‘}CON.Person\text{’})$:- $O(id, \text{‘}Author\text{’})$.
$O(id, \text{‘}CON.Association\text{’})$:- $O(id, \text{‘}Organization\text{’})$.
$O(id, \text{‘}CON.Venue\text{’})$:- $O(id, \text{‘}Conference\text{’})$.

Publications in the Concept layer is a high level definition that encloses all the documents that have been made available to the public, like *Books*, *Papers* and *Posters*, in our example. The following mapping depicts this aggregation.

$O(id, \text{‘}CON.Publication\text{’})$:- $O(id, \text{‘}Book\text{’})$.
$O(id, \text{‘}CON.Publication\text{’})$:- $O(id, \text{‘}Paper\text{’})$.
$O(id, \text{‘}CON.Publication\text{’})$:- $O(id, \text{‘}Poster\text{’})$.

And to exemplify the mapping of properties, we assume that the name and the semantics of the properties will be the same, then we will have the following mappings for *Paper*, *Book* and *Poster* with respect to *Publication*. Note that as *Books* and *Posters* do not have the property *pages*, then the pages for the mentioned objects are *NULL*.

$PO(id, \text{‘}CON.title\text{’}, val)$:- $PO(id, \text{‘}title\text{’}, val), O(id, \text{‘}Paper\text{’})$.
$PO(id, \text{‘}CON.title\text{’}, val)$:- $PO(id, \text{‘}title\text{’}, val), O(id, \text{‘}Book\text{’})$.
$PO(id, \text{‘}CON.title\text{’}, val)$:- $PO(id, \text{‘}title\text{’}, val), O(id, \text{‘}Poster\text{’})$.

$PO(id, \text{‘}CON.year\text{’}, val)$:- $PO(id, \text{‘}year\text{’}, val), O(id, \text{‘}Paper\text{’})$.
$PO(id, \text{‘}CON.year\text{’}, val)$:- $PO(id, \text{‘}year\text{’}, val), O(id, \text{‘}Book\text{’})$.
$PO(id, \text{‘}CON.year\text{’}, val)$:- $PO(id, \text{‘}year\text{’}, val), O(id, \text{‘}Poster\text{’})$.

$PO(id, \text{‘}CON.pages\text{’}, val)$:- $PO(id, \text{‘}pages\text{’}, val), O(id, \text{‘}Paper\text{’})$.
$PO(id, \text{‘}CON.pages\text{’}, NULL)$:- $O(id, \text{‘}Book\text{’})$.
$PO(id, \text{‘}CON.pages\text{’}, NULL)$:- $O(id, \text{‘}Poster\text{’})$.

Publishes relates a *Person* and his *Publications*, and as the *Publications* may be formed from many objects, then *Publishes* is formed by aggregating the relations used to connect *Authors* with *Books* and *Papers* (*writes*), and the relation between *Author* and *Posters* (*creates*).

$$R(id1, id2, `CON.Publishes') \text{ :- } O(id1, `CON.Person'),\\ O(id2, `CON.Publication'), R(id1, id2, `writes').$$
$$R(id1, id2, `CON.Publishes') \text{ :- } O(id1, `CON.Person'),\\ O(id2, `CON.Publication'), R(id1, id2, `creates').$$

Appears relates, at the Concept layer, the *Publications* and the *Event* in which they appear. As the relations at the Data layer are defined by *presented* (for *Papers*) and *displayed* (for *Posters*), then *Appears* is the union of both relations.

$$R(id1, id2, `CON.Appears') \text{ :- } O(id1, `CON.Publication'),\\ O(id2, `CON.Event'), R(id1, id2, `presented').$$
$$R(id1, id2, `CON.Appears') \text{ :- } O(id1, `CON.Publication'),\\ O(id2, `CON.Event'), R(id1, id2, `displayed').$$

The following relations are defined similarly to the previous relationships.

$$R(id1, id2, `CON.Associates') \text{ :- } O(id1, `CON.Person'),\\ O(id2, `CON.Association'), R(id1, id2, `affiliated').$$
$$R(id1, id2, `CON.References') \text{ :- } O(id1, `CON.Publication'),\\ O(id2, `CON.Publication'), R(id1, id2, `cites').$$

And finally, the properties of the relations are mapped. In our case, only the *start_year* in *Associates*.

$$PR(id1, id2, `CON.startYear', val) \text{ :- } O(id1, `CON.Person'),\\ O(id2, `CON.Association'), R(id1, id2, `affiliated'),$$

$$PR(id1, id2, \text{‘}start\_year\text{’}, val).$$

4.3.2 Concept to Network Layer

The objects and relations in the Network layer are defined analogously to the elements in the Concept layer, according to the mapping specified in Figure 4.3. Note that *Publication*, from the Concept layer, in this Network layer is represented as *Actor* and *Evidence*, since it can be the mediator of a relation, like for *coauthor*, but can also be an active part of the relation, like in *cites*.

The following set of mapping are the ones that depend directly on the Concept layer.

$O(id, \text{‘}NET.Group\text{’})$:- $O(id, \text{‘}CON.Association\text{’})$.
$O(id, \text{‘}NET.Artifact\text{’})$:- $O(id, \text{‘}CON.Publication\text{’})$.
$O(id, \text{‘}NET.Actor\text{’})$:- $O(id, \text{‘}CON.Person\text{’})$.
$O(id, \text{‘}NET.Event\text{’})$:- $O(id, \text{‘}CON.Venue\text{’})$.

And the rules expressed with the relations and specializations/generalizations are represented by the following set of mappings.

$O(id, \text{‘}NET.Actor\text{’})$:- $O(id, \text{‘}NET.Group\text{’})$.
$O(id, \text{‘}NET.Actor\text{’})$:- $O(id, \text{‘}NET.Artifact\text{’})$.
$O(id, \text{‘}NET.Evidence\text{’})$:- $O(id, \text{‘}NET.Event\text{’})$.
$O(id, \text{‘}NET.Evidence\text{’})$:- $O(id, \text{‘}NET.Artifact\text{’})$.

And all the relations are grouped into one single *Relation* in the Network layer.

$R(id1, id2, \text{‘}NET.Relation\text{’})$:- $O(id1, \text{‘}NET.Actor\text{’}), O(id2, \text{‘}NET.Actor\text{’}),$
$R(id1, id2, \text{‘}CON.Coauthors\text{’})$
$R(id1, id2, \text{‘}NET.Relation\text{’})$:- $O(id1, \text{‘}NET.Actor\text{’}), O(id2, \text{‘}NET.Artifact\text{’}),$
$R(id1, id2, \text{‘}CON.Publishes\text{’})$

$$R(id1, id2, \text{‘}NET.Relation\text{’}) \text{ :- } O(id1, \text{‘}NET.Group\text{’}), O(id2, \text{‘}NET.Group\text{’}),$$
$$R(id1, id2, \text{‘}CON.Associates\text{’})$$
$$R(id1, id2, \text{‘}NET.Relation\text{’}) \text{ :- } O(id1, \text{‘}NET.Artifact\text{’}),$$
$$O(id2, \text{‘}NET.Artifact\text{’}), R(id1, id2, \text{‘}CON.References\text{’})$$
$$R(id1, id2, \text{‘}NET.Relation\text{’}) \text{ :- } O(id1, \text{‘}NET.Evidence\text{’}),$$
$$O(id2, \text{‘}NET.Evidence\text{’}), R(id1, id2, \text{‘}CON.Appears\text{’})$$

The mapping of properties is done in a similar way as in the Concept layer. Let us assume that *Artifacts* define a property *name* that can be mapped to the *title* of the *Publication*, just to illustrate how the mapping is produced at this layer.

$$O(id, \text{‘}NET.name\text{’}, val) \text{ :- } O(id, \text{‘}CON.Publication\text{’}),$$
$$PO(id, \text{‘}CON.title\text{’}, val)$$

Let us now consider the query presented in the previous section at the Network layer, in which we want to know all the objects associated to John.

Q11:
```
SELECT NET.RELATION(r1,p1)
 FROM NET.ACTOR r1, NET.ARTIFACT p1,
      NET.RELATION(r1,p1)
WHERE r1.name=‘John’
```

In this case, to simplify, the *Artifacts* are only *Papers*, *Books* and *Posters*. That query can be interpreted as a series of simple queries, one for every valid mapping of object at the Data layer. The SociQL query shown before is equivalent to the aggregation of the following simple SociQL queries:

Q12:
```
SELECT writes(i1,p1)
 FROM author i1,
      paper p1, writes(i1,p1)
 WHERE i1.name="John"
```

```
SELECT creates(i1,p1)
FROM author i1,
     poster p1, creates(i1,p1)
WHERE i1.name="John"
```

```
SELECT writes(i1,p1)
FROM author i1,
     book p1, writes(i1,p1)
WHERE i1.name="John"
```

The specific translation of the query depends on the model of the social network. In this case we assume there are only three different *Artifacts*: *Paper*, *Poster* and *Book*; and only one kind of *Actor*: *Author*.

4.4 Mappings within Layers for Derived Relationships

Concept layer also contemplates the definition of derived relationships. In this example, we declare the *Coauthors* relation, which abstracts the evidence (i.e., the paper) between the various authors, and pairwise relates its authors.

$$\begin{aligned} R(id1, id2, \text{`}CON.Coauthors\text{'}) \text{ :- } & O(id1, \text{`}CON.Person\text{'}), \\ & O(id3, \text{`}CON.Publication\text{'}), \\ & O(id2, \text{`}CON.Person\text{'}), \\ & R(id1, id3, \text{`}CON.Publishes\text{'}), \\ & R(id2, id3, \text{`}CON.Publishes\text{'}). \end{aligned}$$

The definition of derived relationships also allows the description of even more complex relationships, like for example, if we want to declare the second order zone[4] of coauthors, then we would have a mapping like the following.

[4]The region of nodes directly linked to a focal node is the first-order zone; the nodes two steps removed from a focal node constitute the second order zone, and so on

$$
\begin{aligned}
R(id1, id2, \text{`}\mathit{CON.Coauthors2ndZone}\text{'}) \text{ :- } & O(id1, \text{`}\mathit{CON.Person}\text{'}), \\
& O(id3, \text{`}\mathit{CON.Publication}\text{'}), \\
& O(id4, \text{`}\mathit{CON.Person}\text{'}), \\
& R(id1, id3, \text{`}\mathit{CON.Publishes}\text{'}), \\
& R(id4, id3, \text{`}\mathit{CON.Publishes}\text{'}), \\
& O(id5, \text{`}\mathit{CON.Publication}\text{'}), \\
& O(id2, \text{`}\mathit{CON.Person}\text{'}), \\
& R(id4, id5, \text{`}\mathit{CON.Publishes}\text{'}), \\
& R(id2, id5, \text{`}\mathit{CON.Publishes}\text{'}).
\end{aligned}
$$

Chapter 5

The Language Specification

This chapter introduces our SQL-like language for social networks, by describing the semantics involved in SociQL statements.

In the examples presented through the chapter, we shall use use the database described in Figure 5.1, based in the network depicted in Figure 3.1. The database schema is derived from the formal definition on Chapter 4, where the sets constituting the social network are tables, and their elements are the attributes[1]. We will call this database $\mathrm{SOCIAL\_NETWORK}$.

In this setting, the objects are represented by tuples in the table O, which contains two attributes: *id*, representing a unique identifier for the object, and *label* specifying the type of the object. For example, when we refer to the objects of type *author*, we refer to the tuples in O which *label* attribute is equals to *author*. The relational algebra equivalent to refer to objects of type *author* is

$$\sigma_{label=`author'}(O)$$

The properties of objects are represented by tuples in the table PO, which contains three attributes: *id* referencing an object in the table O, *label* representing the name of the property, and *value* representing the value of the property. In some occasions in the query execution we want to refer to objects with certain properties, like *papers* written in *2007* and with title equals to *X*. Then, in those cases we are refering to an object in table O that is referenced by a series of tuples in table

[1]We will reserve the word *relation* to refer to the element in our data model. Whenever we refer to a relation in the context of a relational database, we will use the word *table*

O	
id	label
MIT	Organization
Stanford	Organization
John	Author
Charlie	Author
Alice	Author
Mike	Author
Paper1	Paper
Paper2	Paper
Data Mining	Book
Poster1	Poster
VLDB	Conference
SIGMOD	Conference
Clustering	Keyword
XML	Keyword

PO		
id	label	value
John	name	John
Charlie	name	Charlie
Alice	name	Alice
Mike	name	Mike
Paper1	title	Structured search...
Paper1	year	2010
Paper1	pages	313-324
Paper2	title	Implementation...
Paper2	year	1995
Paper2	pages	9-14
Poster1	title	NN-Queries...
Poster1	year	2008
Data Mining	title	Data Mining
Data Mining	year	1999
Data Mining	pages	315
MIT	name	MIT
Stanford	name	Stanford
VLDB	name	VLDB
VLDB	fullname	Very Large...
SIGMOD	name	SIGMOD
SIGMOD	fullname	Special Interest...
XML	keyword	XML
Clustering	keyword	Clustering

R		
id1	id2	label
Mike	MIT	Affiliated
Alice	MIT	Affiliated
Charlie	Stanford	Affiliated
John	Stanford	Affiliated
Mike	Paper2	Writes
Alice	Paper1	Writes
Alice	Paper2	Writes
John	Paper1	Writes
John	Data Mining	Writes
Alice	Data Mining	Writes
Charlie	Data Mining	Writes
Alice	Poster1	Creates
Paper1	VLDB	Presented
Paper2	SIGMOD	Presented
Poster1	SIGMOD	Presented
Paper1	Clustering	Contains
Paper2	XML	Contains
Poster1	XML	Contains
Paper1	Paper2	Cites

PR			
id1	id2	label	value
Mike	MIT	start_year	1995
Alice	MIT	start_year	2003
Charlie	Stanford	start_year	1999
John	Stanford	start_year	1999

Figure 5.1: Database state for the SOCIAL_NETWORK relational database schema. The underlined attributes denote the primary keys of the tables. The data in the tables is based in the network shown in Figure 3.1.

PO with some specific values in the attributes for *label* and *value*. The relational algebra equivalent for our example is represented by O' in the following expression

$$\begin{aligned} O1 &= \sigma_{\text{label='paper'}}(O) \\ PO1 &= \sigma_{\text{label='year' AND value=2007}}(PO) \\ PO2 &= \sigma_{\text{label='title' AND value='X'}}(PO) \\ O' &= \pi_{\text{O1.id,O1.label}}(O1 \bowtie_{\text{O1.id=PO1.id}} (PO1 \bowtie_{\text{PO1.id=PO2.id}} PO2)) \end{aligned}$$

Similar to the representation of objects, the relations are represented by tuples in the table R, which contains three attributes: *id1* and *id2* representing the objects linked in the relation, and *label* specifying the type of the relation. For example, when we refer to the relation of type *writes*, we refer to the tuples in R which *label* attribute is equals to *writes*. The relational algebra equivalent to refer to relations of type *writes* is

$$\sigma_{label='writes'}(R)$$

The properties of relations are represented in a similar way to properties of objects, by tuples in the table PR, which contain four attributes: *id1* and *id2* referencing a relation in the table R, *label* representing the name of the property, and *value* representing the value of the property. In some steps of the query execution we refer to relations with certain properties, like *affiliated* relations with start year greater than *2003*. Then, in those cases we are refering to a relation in table R that is referenced by a series of tuples in table PR with some specific values in the attributes for *label* and *value*. The relational algebra equivalent for our example is represented by R' in the following expression

$$\begin{aligned} R1 &= \sigma_{\text{label='affiliated'}}(O) \\ PR1 &= \sigma_{\text{label='start\_year' AND value>2003}}(PR) \\ R' &= \pi_{\text{R1.id1,R1.id2,R1.label}}(R1 \bowtie_{\text{R1.id1=PR1.id1 AND R1.id2=PR1.id2}} PO1) \end{aligned}$$

In the rest of this chapter, we introduce the different clauses of SociQL, defining the semantics and showing strategies for evaluating statements in our language.

5.1 Basic Queries in SociQL

Perhaps the simplest form of query in SociQL asks for networks that satisfy a predicate. Queries in SociQL can be complex, including path relationships and filtering clauses, but in this section we will start with simple queries, and then progress to more complex ones in a step-by-step manner. The basic SociQL statement is formed of three clauses, `SELECT`, `FROM` and `WHERE` and has the following form:

`SELECT` $Relation R_1, ..., Relation R_n$

`FROM` $Object\ O_1, ..., Object\ O_k, Relation\ R_1, ..., Relation\ R_t$

`WHERE` $Predicate P_1$ `AND` ... `AND` $Predicate P_m$

Through the presentation of the semantics and execution strategy for basic queries, we will use an example query to illustrate the meaning of each clause and the execution of every step in the strategy. The example query will attempt to get the network of *authors* and *papers*, in which authors are affiliated to *MIT*. The SociQL query is:

Q13:
```
SELECT writes(r1,p1)
FROM author r1, organization o1, paper p1
     affiliated(r1,o1), writes(r1,p1)
WHERE o1.name='MIT'
```

The main construct in SociQL is the familiar *select-from-where*.

- The `FROM` clause gives the objects and relations involved in the search pattern the query refers. In our example, the query is about the objects *Paper*, *Author* and *Organization*, and the relations *Writes* and *Affiliated*.

  A $Relation$, in this context, establishes an association between pairs of objects. A relation $R(A, B)$ retrieves a subset of all combinations of tuples representing objects with the same label as A and tuples representing objects with the same label as B. More formally, the relation $R(A, B)$ is:

  $$R(A, B) \subseteq \sigma_{label=a}(O) \times \sigma_{label=b}(O)$$

  where a is the label of object A and b is the label of object B.

- The `WHERE` clause provides the selection predicates, much like a selection-condition in relational algebra, which properties must satisfy in order to match the query. In our example, the predicate is that the property *name* of the object labeled *Organization* has the value *MIT*.

  Formally, a $Predicate$ is a *selection predicate* which filters out objects based on the values of the properties. A predicate s is a function $s : dom(t_1) \times dom(t_2) \rightarrow Boolean$, where $t_1, t_2 \in T$ are simple types and $Boolean = true, false$.

  The predicates support eight comparison operators: $=$ (equals), $! =$ (not equals), $>$ (greater than), $>=$ (greater or equal than), $<$ (less than), $<=$ (less or equal than), $><$ (contains) and $<>$ (not contains). The first six operators can be applied to ordinal properties, while nominal properties can only use $=$, $! =$, $><$ and $<>$.

- The `SELECT` clause tells which relations that describe the network to be returned. In our query, the network returned is formed by the objects *Paper (p1)* and *Author (r1)* and the relation *Writes*.

The basic strategy for executing SociQL statements can be stated as follows:

Step 1. We will assume a new relational database, named $\mathrm{SEARCH\_PATTERN}$, with the same database schema as $\mathrm{SOCIAL\_NETWORK}$. The purpose of this schema, is to create a copy of the social network, with only the relevant tuples for the query. We will use the subscript N, for tables in the $\mathrm{SOCIAL\_NETWORK}$ database, and P, for tables in the $\mathrm{SEARCH\_PATTERN}$ database.

Step 2. The query considers all tuples of the objects in the `FROM` clause. However, we only need to consider the objects in which the predicates in the `WHERE` clause are *true*. For example, in our query, we have to consider the objects with label *author*, *paper* and only the *organizations* with the *name* property with value *MIT*. Then, for every object O_i, where $1 \leq i \leq k$, present in the `FROM` clause, we populate the table O_P with all tuples of table O_N in which

O	
id	label
MIT	Organization
John	Author
Charlie	Author
Alice	Author
Mike	Author
Paper1	Paper
Paper2	Paper

(a)

R		
id1	id2	label
Mike	MIT	Affiliated
Alice	MIT	Affiliated
John	Stanford	Affiliated
Charlie	Stanford	Affiliated
Mike	Paper2	Writes
Alice	Paper1	Writes
Alice	Paper2	Writes
John	Paper1	Writes

(b)

Figure 5.2: $\mathrm{SEARCH\_PATTERN}$ database after executing Step 2 and Step 3 in the execution strategy for *Q13*.

the label is the same as in O_i and the conditions applied to the properties of the object are *true*. Figure 5.2(a) shows table O_P for our example. In this table, we copy all *authors* and *papers*, and all *organizations* with name *MIT*.

Step 3. Similarly, the query ask us to consider all tuples of the relations in the `FROM` clause, and in the same way as objects, we only need to consider the relations in which the predicates in the `WHERE` clause are *true*. For example, in our query, we have to consider the relations with label *affiliated* and *writes*. Then, for every relation R_i, where $1 \leq i \leq t$, present in the `FROM` clause, we populate the table R_P with all the tuples of table R_N in which the label is the same as in R_i, and the conditions applied to the properties of the relation are *true*. Figure 5.2(b) shows table R_P for our example, after we copy all *writes* and *affiliated* relations.

Step 4. Now we start to look for substitutions that match the search pattern described in the `FROM` clause. When a substitution of the search pattern is found, we keep the participating objects and relations, otherwise, if a tuple representing an object or a relation is not in any substitution of the search pattern, then it is removed. In our example, the tuple in table R with value $(John, Paper1, Writes)$ is not part of a valid substitution of the search pattern, since there is not a pattern with the object *John*, because he is not affiliated to *MIT*. In order to find the valid objects and relations with respect to the search pattern, we create a temporary table containing the result of the join of the elements, following the pattern presented in the `FROM` clause. In our example, we create a temporary table with the join of objects and relations,

O	
id	label
MIT	Organization
Alice	Author
Mike	Author
Paper1	Paper
Paper2	Paper

(a)

R		
id1	id2	label
Mike	MIT	Affiliated
Alice	MIT	Affiliated
Mike	Paper2	Writes
Alice	Paper1	Writes
Alice	Paper2	Writes

(b)

Figure 5.3: $\mathrm{SEARCH\_PATTERN}$ database after executing Step 4 in the execution strategy for *Q13*.

according to the search pattern. The join in our example is equivalent to J in the following relational algebra expression:

$$\begin{aligned}
O1 &= \sigma_{label=`Paper'}(O_P) \\
O2 &= \sigma_{label=`Author'}(O_P) \\
O3 &= \sigma_{label=`Organization'}(O_P) \\
R1 &= \sigma_{label=`Writes'}(R_P) \\
R2 &= \sigma_{label=`Affiliated'}(R_P) \\
J &= O1 \bowtie_{id=id1} R1 \bowtie_{id=id2} O2 \bowtie_{id=id1} R2 \bowtie_{id=id2} O3
\end{aligned}$$

The resulting table will contain all the valid patterns for the query. From this table we can check if the objects and relations in the tables O_P and R_P respectively, are part of the search pattern. If the tuple is part of, at least, one substitution of the search pattern, we keep it, otherwise, it is deleted. Figure 5.3(a) and Figure 5.3(b) show the table for relations O_P and R_P, respectively, after removing the tuples not included in at least one substitution of the search pattern.

Step 5. As the query may not select all the network specified in the `FROM` clause, but just a portion, then we delete from R_P all the tuples in which the label is not included in the selection. In the same way, we keep only the tuples for the objects participating in the selection in O_P. In our query, we specify in the `SELECT` clause that we only want to get the relation *writes*, then, all the relations in R_P with a label different from *writes* are removed. Similarly, we only want to get the objects participating in the relation *writes*, which are *paper* and *author*, then all the objects in table O_P with a label different than

O	
id	label
Alice	Author
Mike	Author
Paper1	Paper
Paper2	Paper

(a)

R		
id1	id2	label
Mike	Paper2	Writes
Alice	Paper1	Writes
Alice	Paper2	Writes

(b)

PO		
id	label	value
John	name	John
Charlie	name	Charlie
Alice	name	Alice
Mike	name	Mike
Paper1	title	Structured search...
Paper1	year	2010
Paper1	pages	313-324
Paper2	title	Implementation...
Paper2	year	1995
Paper2	pages	9-14

(c)

Figure 5.4: $\mathrm{SEARCH\_PATTERN}$ database after executing every step in the execution strategy for *Q13*.

paper and *author* are removed. Figure 5.4(a) and Figure 5.4(b) shows the final tables O_P and R_P for our example, after the tuples not included in the selection are removed.

Step 6. Once we know all the elements participating in the query, we start to add the properties. For object properties, we include every tuple from PO_N where the *id* is present in the table O_P we just populated. Figure 5.4(c) shows the final table PO_P for our example, after the execution of this step.

Step 7. Analogously to the objects, for properties of relations we include every tuple from PR_N where the two *ids* are present in the table R we just populated. In our example, we only select the relation *writes*, and since it does not contain properties, then the table PR_P will be empty.

5.2 Path Expressions for Structure-Specifying Queries

When querying a *n*-mode social network[2], especially when the exact pattern is unknown, it is convenient to use a form of navigational querying based on path expressions. The idea is to specify paths in any of the layers of the architecture, that define a sequence of relations in the network. This section explores the path expressions, which enable users to obtain a set of objects reachable from and to a constrained object.

[2]The term “modes” refers to a distinct set of actors. A network data set containing more than two sets of actors is referred as an *n*-mode network.

The expression `NEIGHBORHOOD(X,Y,`$\langle max\_length \rangle$`[,`$\langle relation\_list \rangle$`])` implies:

- a finite sequence $o_0, r_1, ..., r_n, o_n$, where $r_1, ..., r_n$ represent the relations and $o_1, ..., o_n$ denote the objects
- for each i, there is a relation r_i between o_{i-1} and o_i
- the maximum number of relations in the path cannot exceed the number specified by `max_length`.
- if the expression defines a relation list, the type of every relation $r_1, ..., r_n$ must be one of the types specified in `relation_list`
- starts from an object $X = o_1$, ends in an object $Y = o_n$ and there may be several paths that match the expression $X, r_1, ..., r_n, Y$

Path expressions are helpful to find connections and possible impact zones in social networks. Nevertheless, they represent a syntactic convenience, since we can show how a query can be translated to a series of basic SociQL queries.

We illustrate the evaluation of queries with path expressions through an example. Consider the query in which we want to get the organizations linked to *Alice* directly or indirectly.

```
Q14: SELECT PATH(r1,o1)
     FROM author r1, organization o1,
          NEIGHBORHOOD(r1,o1,4)
     WHERE r1.name='Alice'
```

In the query above, we evaluate the objects named $r1$ and $o1$ and the path expression $r1, m, A, n, B, p, C, q, o1$. This path can be interpreted as: start from author $r1$, follow a relation m that leads to object A, then a relation n to B, then follow a relation p that leads to object C, and finally a relation q to organization $o1$. Furthermore, we limit the length of the path to a maximum of four, allowing shorter paths, like $r1, m, A, n, o1$, to form part of the answer as well. Since there may be several path expressions that match the path, we can interpret the query as a set of simple queries, one for each path combination, that are accumulated for the final result.

The precise translation of the query depends on the model that was defined for the social network at the data layers. In this particular example, there are three possible paths from an author to an organization (with at most four relations).

- A direct relation:
  $r1, affiliated, o1$
- Through a coauthor:
  $r1, writes, a2, writes, a1, affiliates, o1$
- Through an author who cited a paper written by Alice:
  $r1, writes, a2, cites, a3, writes, a1, affiliates, o1$

Then, the SociQL query shown above, is represented as the union of basic SociQL queries, with path expressions translated and shown in the following series of queries:

Q15:
```
SELECT affiliated(r1,o1)
FROM author r1, organization o1,
     affiliated(r1,o1)
WHERE r1.name='Alice'
________________________________

SELECT writes(r1,a2), writes(a1,a2),
     affiliated(a1,o1)
FROM author r1, organization o1,
     author a1, paper a2,
     writes(r1,a2), writes(a1,a2),
     affiliated(a1,o1)
WHERE r1.name='Alice'
________________________________
```

```
SELECT writes(r1,a2), cites(a2,a1),
     writes(a1,a3),
     affiliated(a1,o1)
FROM author r1, organization o1,
     author a1, paper a2, paper a3
     writes(r1,a2), cites(a2,a1),
     writes(a1,a3),
     affiliated(a1,o1)
WHERE r1.name='Alice'
```

5.3 Filtering Clause for Importance-Specifying Queries

In realistic settings, a SociQL query produces a large number of elements, unless one resorts to overly restrictive and cumbersome queries. This problem is known as the *many answers problem* [22] in the literature, and its typical solution involves ranking of the results by their importance. Unlike typical database settings, where the notion of importance is often ill-defined, in SociQL, importance is defined according to standard notions of node importance from sociology. In SociQL, the semantics of the filtering clause enables the definition of a centrality measure that will enrich the answer with a score, based on the importance of the actors, in order to filter out the objects that are not in the range specified in the clause.

In this section we describe in detail one of the novel aspects of SociQL, namely the filtering clause, and we explain how query results are filtered in terms of the requested centrality measure. In order to get the output filtered by a centrality measure, we add to the select-from-where statement a clause:

`FILTER BY` $CentralityMeasure$ `OF` O_s `ON` $relation\_list\ operator\ number$

The filtering clause calculates the centrality scores for a specific type of object, given a network described by a relation list, and then filters out the objects which filtering condition evaluates to *false*.

Through the presentation of the execution strategy for queries with `FILTER BY` clause, we will use an example query to illustrate the execution steps in the strategy. The example query will attempt to get the network of *authors* and *papers*, in which authors are affiliated to *MIT*, but will include only the authors with more than one paper. The SociQL query can be expressed as:

Q16:
```
SELECT writes(r1,p1)
FROM author r1, organization o1, paper p1
     affiliated(r1,o1), writes(r1,p1)
WHERE o1.name='MIT'
FILTER BY (OUTDEGREE OF r1 ON writes) > 1
```

When a `FILTER BY` clause is present in a query, a network has to be created over which an algorithm for actor centrality will be executed, in order to get a list of object ids and associated centrality scores. The following strategy presents the sequence of steps to create the network that will be used to calculate actor centrality.

Step 1. We will assume a new relational schema, named $\mathrm{CENTRALITY\_NETWORK}$, with the same schema as $\mathrm{SOCIAL\_NETWORK}$, but only for tables O and R, because in order to calculate the centrality scores we do not need properties. The purpose of this schema, is to create a copy of the social network, with only the relevant tuples for the network to calculate actor centrality. We will use the subscript N, for tables in the $\mathrm{SOCIAL\_NETWORK}$ schema, and C, for tables in the $\mathrm{CENTRALITY\_NETWORK}$ schema.

Step 2. For every object O_i, where $1 \leq i \leq k$, present in the `FROM` clause, we populate the table O_C with all tuples of table O_N in which the label represents an object that participates in one or more of the relations in the list of the filtering clause. However, in this case we are not going to evaluate the predicates in the SociQL query, as we did with the basic queries, because we need the entire network to calculate the centrality measure, not just the result network. Figure 5.5(a) shows table O_C for our example, after the execution of this step.

Step 3. Similarly, for every relation R_i, where $1 \leq i \leq t$, present in the `FROM` clause, we populate the table R_C with all the tuples of table R_N in which the

O	
id	label
John	Author
Charlie	Author
Alice	Author
Mike	Author
Paper1	Paper
Paper2	Paper

(a)

R		
id1	id2	label
Mike	Paper2	Writes
Alice	Paper1	Writes
John	Paper1	Writes
Alice	Paper2	Writes

(b)

Figure 5.5: Tables for CENTRALITY_NETWORK after applying the strategy to create networks for centrality calculation in *Q16*.

O	
id	label
Alice	Author
Mike	Author
Paper1	Paper
Paper2	Paper

(a)

R		
id1	id2	label
Mike	Paper2	Writes
Alice	Paper1	Writes
Alice	Paper2	Writes

(b)

Figure 5.6: Tables O and R after the execution of the example query *Q16*. The rows highlighted are the tuples that are removed by the filtering clause.

label is in the relation list of the filtering clause, and the label of the participating objects correspond to the ones of the relation in the query. Figure 5.5(b) shows table R_P for our example, after the execution of this step.

Step 4. Once we have a network defined by objects and relations, the network is used by a function that processes it, according to the centrality algorithm specified in the `FILTER BY` clause. The function returns a list of object ids and associated centrality scores.

When a query involving filtering by importance is executed, two processes have to be performed. First, a network for centrality calculation is created, as described before, in order to get the centrality scores. And then, the query is executed as a basic query, ignoring the filtering clause, but in Step 1 of the basic query execution, we remove the tuples in O_P which score is not in the range specified by the filtering clause. Figure 5.6 shows the resulting tables O_P and R_P for the example, using the basic query execution, but highlighting the tuples that are removed with the filtering clause. In our example, we remove the tuple $(Mike, Author)$ from O_P because the degree score for *Mike* is 1, and consequently, the relations in R_P involving *Mike* have to be removed.

O	
id	label
Mike	Author
Paper2	Paper

(a)

R		
id1	id2	label
Mike	Paper2	Writes

(b)

Figure 5.7: Tables O and R after the execution of example query *Q17*.

5.4 Limiting the Output

Sometimes we may not want to retrieve all the records that satsify the critera specified in the basic query. The `LIMIT` clause is used to limit query results to those that fall within a specified range. In other words, the clause can be used to show the first *n* result patterns. To get limited output, we add at the end of the query a clause:

$$LIMIT\ max\_results$$

The following example is a rewrite of our original example, presented in the execution of basic queries, but in this case we want to limit the output to the first substitution of the query pattern.

Q17:
```
SELECT writes(r1,p1)
FROM author r1, organization o1, paper p1
     affiliated(r1,o1), writes(r1,p1)
WHERE o1.name=‘MIT’
LIMIT 1
```

The execution is the same as a basic query, but in Step 4 of the execution, we only take into account the first $max\_results$ tuples generated with the join, and the rest are discarded. Figure 5.7 shows a possible result of the example presented in *Q17*.

Chapter 6

Query Engine Implementation

In this chapter, we present an implementation of the query engine, exposing the considerations in the design and implementation of the engine and the format in which the results are presented to the user. The remainder of the chapter presents a way of visualizing the results using the appropriate SociQL query.

6.1 Architecture

The architecture of the SociQL engine is presented in Figure 6.1. Once a query is received from an application (such as a query editor), the *lexical analyzer* converts the query into a sequence of tokens, that then is analyzed by the *parser* to determine the validity of the grammatical structure of the query and creates an internal representation of it. Then, an *execution plan* is found for the query, specifying the order in which the statements are executed, and, specially, the order in which data from external sources will be requested. At the *query processing* step, all external data (if any) used by the query is fetched and stored locally first, then the actual SociQL query is translated into an SQL expression that is executed on the local relational database containing all data needed by the query. Finally, an additional step may be necessary, when the `FILTER BY` or `ORDER BY` clause is present; namely to *rank* the objects in the answer to the query according to their importance. The remainder of this chapter details some of the challenges in each of these steps.

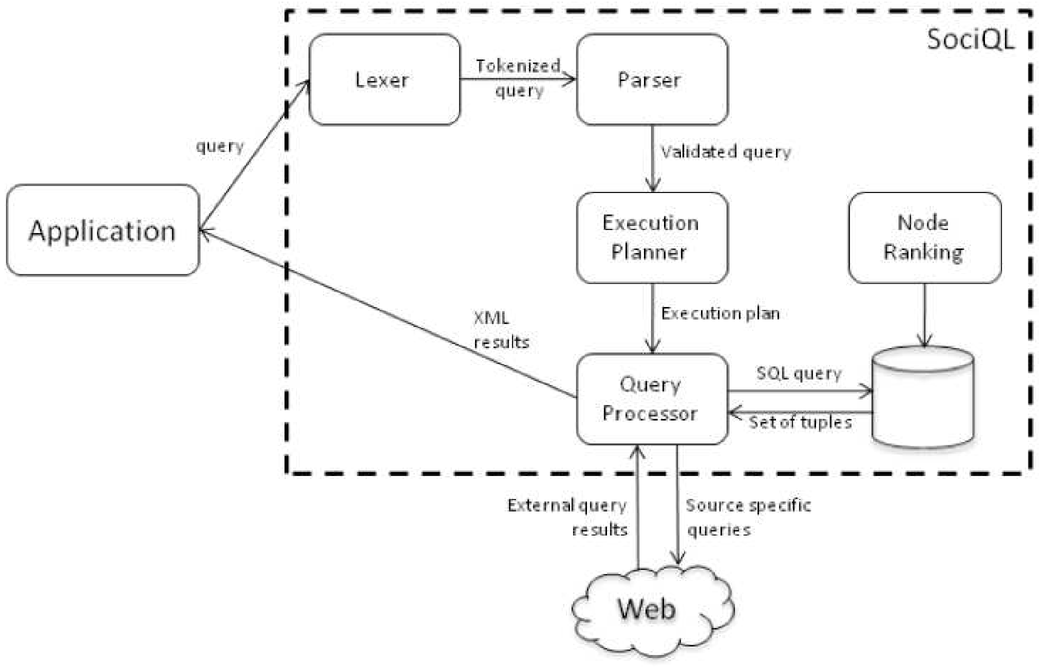


Figure 6.1: SociQL Architecture.

6.2 The System Catalog

The catalog describes all the social network elements in the system and the mechanism by which they can be accessed. More specifically, these are essentially queries that return

- for objects: unique id within the network;
- for relations: unique pair of object ids;
- for object properties: an object id and a value; and
- for relationship properties: triples with two object ids and a value.

Table 6.1 shows part of the catalog for the social networks on top of ReaSoN.

For networks whose data is accessible through APIs, such as Facebook or DBpedia, the catalog contains the API calls that produce data in the same format. When a query is executed, if necessary, our tool will allow the user to authenticate into the remote social network site (e.g., Facebook), thus ensuring access control and

Table 6.1: Model Specification for Research Network

Type	Attribute	Value
Context	Name Endpoint Type	Reason Local SQL
Object	Name Site Query ObjectId	Author Reason `SELECT author_id FROM researcher` researcher_id
Property	Name Object/Relation Query Type	Id Author `SELECT researcher_id FROM researcher` Nominal
Property	Name Object/Relation Query Type	Name Author `SELECT researcher_id, name FROM researcher` Nominal
Object	Name Site Query ObjectId	Paper Reason `SELECT paper_id FROM paper` paper_id
Property	Name Object/Relation Query Type	Id Paper `SELECT paper_id FROM paper` Nominal
Property	Name Object/Relation Query Type	Title Paper `SELECT paper_id, title FROM paper` Nominal
Relation	Name FromProperty ToProperty Query	Writes Author.Id Paper.Id `SELECT author_id AS id1, paper_id AS id2 FROM writes`

privacy settings are respected. After, we are able to materialize all network data locally, and convert SociQL expressions over equivalent SQL ones that identify all objects that belong in the answer to the query.

6.3 Actor Centrality Calculation

A central issue when visualizing a subset of a social network (such as the result of a SociQL query), is identifying the *relative importance* of the objects in that set, in contrast with a search in the database sense, where one is interested in *every* object that satisfies the criteria.

Finding the relative importance of objects in social networks is a vast field comprising decades of work. Because there are so many metrics to be used, each with its own nuances in interpretation, we designed SociQL to be as orthogonal as possible to the specific notion of reputation used. In our current implementation, the *Node Ranker* module is implemented as a web service that requests for the graph and the centrality measure, and returns the list of nodes with its associated score. Currently, the centrality measures supported are: degree, indegree, outdegree, closeness, betweenness and pagerank.

Ranking of objects is done explicitly in SociQL: we provide a language contruct, `FILTER BY`, that specify how the ranking should be done. In order to calculate the visibility, we define a network with the objects declared in the `FROM`, but including only the relations specified in the clause. Then this network is sent to the *Node Ranker* where is processed using libraries to model and analyze graphs. After the graph is processed, the web service returns a list of nodes with its respective score. Once we have the nodes and scores, we select only the nodes that are included in the result and form the result network.

6.4 Query Execution

In this subsection we discuss the actual execution of queries in SociQL (recall Figure 6.1). In our approach, as we stated in previous sections, the data is stored in a relational database, so all the queries defined in SociQL are translated to SQL.

Once the query has been validated and the data materialized in the relational database, the bulk of the work consists in translating the expression in SociQL into an equivalent executable SQL statement, based on the specifications of the networks in the system catalog. Our goal on doing so is to leverage the several decades of performance improvements on relational query processing engines. The main idea behind the translation is to treat each element of the data model defined in the catalog as a *view*; the final query is defined in terms of all such views.

This translation is illustrated through an example. Consider the following query, which returns the coauthors of John together with their papers containing the keyword "Search" in the title. In order to make the example shorter, we will assume *author* has only one property, *name*, and *papers* have two properties, *title* and *year*.

Q18:
```
SELECT writes(r1,p1)
FROM author r1, author r2, paper p1,
     writes(r1,p1), writes(r2,p1)
WHERE r2.name='John'
     AND p1.title><'Search'
```

Notice that there are two properties of *Paper* objects that are needed in the projection of the query: *title* and *year*. This means that the final SQL query will have two table expressions in its `FROM` clause that are individual queries over the base table that stores all the data about *Paper* objects: *p1_title* and *p1_year*. In order to make sure that every *Paper* object is faithfully reconstructed from the database, we define equality joins over object ids in every such view. In our example, this is captured by adding *p1_title.id = p1_year.id* to the `WHERE` clause of the resulting SQL query. In the same way, one property from every author objects are necessary in the table expressions in the `FROM`: *r1_name* and *r2_name*. In this case, there is no need to join the properties, since they are from two different objects. Subsequently, the execution engine examines the predicates and tries to incorporate the conditions into the table expressions in the `FROM` to improve execution time, otherwise, the conditions will be added in the `WHERE` clause of the SQL query. Additionally, the table expressions corresponding to the relations are added to the `FROM`, and as their name may not be unique in the context, then a sequential number is assigned to the alias of the nested query, in this case *writes_1* and *writes_2*. In order to avoid Cartesian products in the relations, the natural joins are defined over the object ids from the relation and the participating objects, like in *r2_name.id=writes_2.author_id AND p1_title.id=writes_2.paper_id*.

Finally, the selection criteria specified in the SociQL expression are added to the SQL query, and expressed over the appropriate view (defining the property or relation). In this order of ideas, when the SociQL query is translated to SQL, we have:

Q19:
```
SELECT r1_name.name, p1_title.title,
       p1_year.year
FROM
       (SELECT id, name FROM author)
          AS r1_name,
       (SELECT id, name FROM author
        WHERE name='John')
          AS r2_name,
       (SELECT id, title FROM paper
        WHERE title='%Search%')
          AS p1_title,
       (SELECT id, year FROM paper
        WHERE title='%Search%')
          AS p1_year,
       (SELECT author_id, paper_id
        FROM writes)
          AS writes_1,
       (SELECT author_id, paper_id
        FROM writes)
          AS writes_2,
WHERE p1_title.id=p1_year.id AND
       r1_name.id=writes_1.author_id AND
       p1_title.id=writes_1.paper_id AND
       r2_name.id=writes_2.author_id AND
       p1_title.id=writes_2.paper_id
```

Figure 6.2 shows schematically how the query is executed using relational algebra. The process follows the same steps described before for the example query.

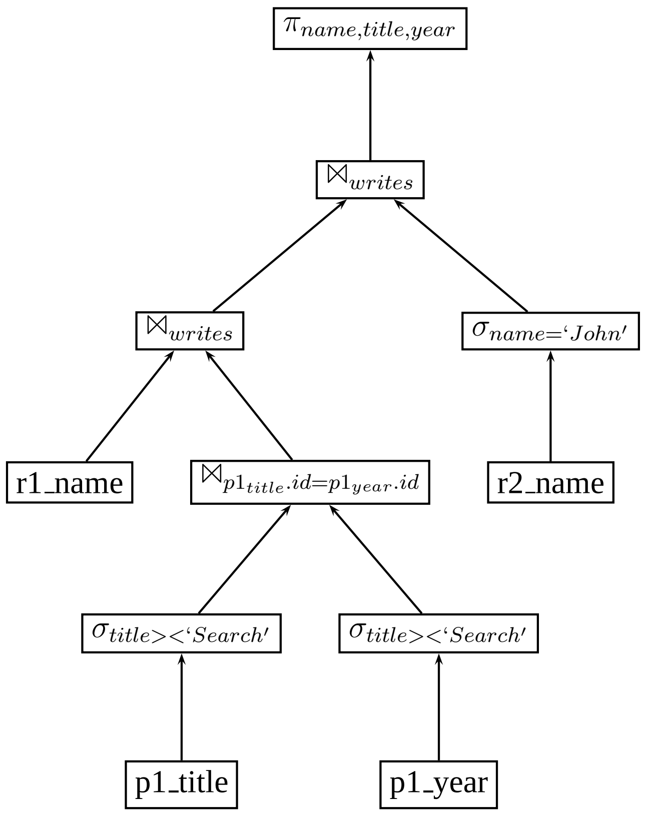


Figure 6.2: Schematic representation of query execution.

6.5 Query Planner

Our query planning step is concerned with identifying the ordering in which to request data from external social networking sites (which are used in interlinking relations). The goal is to find a strategy for answering the queries that minimizes the amount of external data that is retrieved. Some of the issues that need to be considered are the access control restrictions as well as limitations in the number of answers that API calls are allowed to obtain.

The first steps attempt to define the query at the lowest level, in other words, special constructs like semantic network elements or *path* relationships will be translated to simple elements. Whenever a concept or network layer element appears, the query planner finds recursively the proper mapping in the model. As one of those elements can be mapped to multiple elements at the data layer, like the concept element *publication* is mapped to several data elements, like *papers*, *books* and *posters*, then the planner generates a series of queries with valid combinations of the data layer elements. Recall the query presented in Section 4.3 in which we ask for the *actors* and their associated *artifacts*. In this case, three queries at the data layer are generated: *author-paper*, *author-poster*, *author-book*; as the product of applying the mapping on *actor* and *artifact* respectively and checking the consistency in the model.

Once we have all the elements at the lowest level, it is time to plan for the *path* relationships. For queries over path relationships, the planner performs a breadth first search on every path relation, up to a maximum number of levels defined in the query. Then, every path will create an independent query.

At this point, we have a set of queries, but as our objective is to present a unified answer and take advantage of the optimizations in the relational database management system (RDBMS), then all the queries are combined using `UNION`, applying casting to some fields when necessary. The final query can be sent to the RDBMS as a monolythic statement, but before we need to perform a series of optimizations, in order to minimize the number of API calls and result tuples for external queries.

Our solution follows the rationale of Li et al. [35]. Basically, the planner attempts to find the most cost effective plan, by examining the costs of the various subqueries. Local queries, for example, queries directly translatable to SQL and issued to a database on the same intranet are assumed to be the least expensive.

Since local queries are most preferable, the planner makes a subquery with only the local actors, in order to get an initial subset of possible results and, based on it, to narrow the queries to the external resources. In case all the objects are external, the planner focuses first on the most *constrained* objects (those objects which have the highest number of selection criteria defined over them), in an attempt to increase as much as possible the *selectivity* of the queries to be submitted to external sites. In this steps we minimize the number of tuples materialized from external sources, to finally execute the SQL query in retrieve the final result.

6.6 REST APIs

As mentioned before, our main data source is the ReaSoN (REseArcherSOcial Networks) project, which studies the collaboration of researchers, focusing on computing scientists. The ReaSoN dataset is based on two large databases of computer science literature: DBLP and the ACM Digital Library. The current version of ReaSoN has:

- publications: 485,267
- authors: 379,188
- citations: 1,301,365
- venues: 3,793
- organizations: 1,865

In addition, there are other sources of information that were considered in SociQL. *Facebook* is an online social networking service with more than five hundred million active users and more than nine hundred million objects of interactions (e.g.

pages, groups, events), which provides an API to access user's information and affiliations. *DBpedia* provides a SPARQL endpoint to pose queries that access a set of services providing information associated with Wikipedia, including links to other related datasets. DBpedia's knowledge base consists of more than two million entities in thirty different languages, with more than three million links to external content.

Although only Facebook and DBpedia were considered, the data model allows an easy incorporation of diverse data sources defined as a relational model or REST-style services.

In order to interlink the different communities, the different online social networks describing an entity or resource must refer to it with a consistent identifier. However, the developers of such social networks are using identifiers only valid in a particular context, which results in a proliferation of identifiers that prevents the merging of social networks, as described in the Pirandello's identity problem.

In practical terms, the alternative of making approximate joins, computing the similarity of certain attributes is not viable, due to the multiple cases of homonyms and the access restrictions on some attributes imposed by some social networks. But other approaches based on collaboratively tagged data within a social computing framework offers the additional advantage that it effectively reduces name ambiguity so that one person can be referenced through multiple name variations in different situations [45].

In our approach, we use social tagging to connect the different identifiers across the data sources. We extend SociQL's structure with a central repository for entity identifiers. This repository can be imagined like a thesaurus that links the ReaSoN identifier with their counterparts in the other data sources.

6.7 Visualizations and Post-Processing

Some researchers suggest that the use of images is an important part in the progress of various fields. Even the historian Alfred Crosby proposed that is one of only two factors that are responsible for the explosive development of all of modern science [27].

From the beginning, images of social network have provided investigators with new insights about network structures and have helped them to communicate those insights to others [27].

In this section we present details of the implementation and representation of the different formats in which the results are presented to the user. The SociQL engine is designed to work as a web service that feeds other systems that can provide more insight on the results. In the following, we present the several visualizations we developed that consume SociQL results, and a general purpose format in XML.

6.7.1 Tabular Queries and Ordering Clause

As stated in the previous chapter, typical `SELECT` queries in SociQL return data in network form, since it is natural to get a network when the underlying data model is also a network. However, typical query languages, including graph query languages, return the results in tabular form.

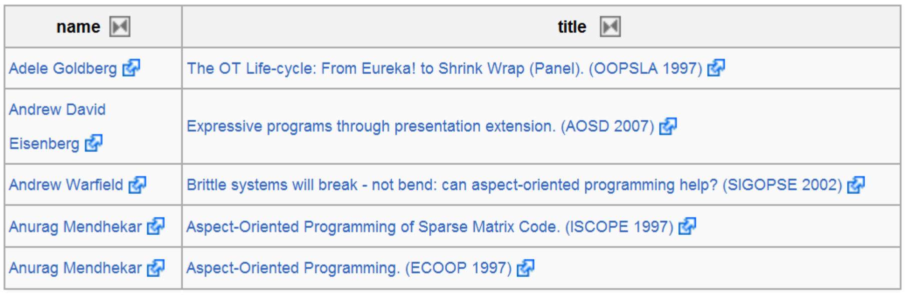
In our implementation, we have included tabular queries, in case the user is interested in only a subset of the properties in the network, which in turn leads to an improvement in the response time of the query engine.

Additionally, we define a post-processing function for tabular queries in SociQL, namely the ordering clause. Through the semantics of the language, we have showed the importance of centrality measures in a social network, and with the `ORDER BY` clause, we seek to enrich the answer by sorting the elements, with respect to a centrality score. The centrality measures that can be used in the ordering clause are: degree, indegree, outdegree, closeness, betweenness and pagerank.

A typical tabular query, with ordering clause, looks like the following:

Q20:
```
SELECT r2.name, p1.title
FROM paper p1, author r1, author r2
     writes(r1,p1) AND writes(r2,p1)
WHERE r1.name=‘Alice’
ORDER BY BETWEENNESS ON r2
```

The previous query returns a table with the name of Alice's coauthors and the title of the paper they coauthored, sorted by the betweenness of the coauthors. In the first line, the query lists the object properties to be projected, indicating the alias of the object and the name of the property, separated by a dot. An example of a result in tabular form can be seen in Figure 6.3.



name	title
Adele Goldberg	The OT Life-cycle: From Eureka! to Shrink Wrap (Panel). (OOPSLA 1997)
Andrew David Eisenberg	Expressive programs through presentation extension. (AOSD 2007)
Andrew Warfield	Brittle systems will break - not bend: can aspect-oriented programming help? (SIGOPSE 2002)
Anurag Mendhekar	Aspect-Oriented Programming of Sparse Matrix Code. (ISCOPE 1997)
Anurag Mendhekar	Aspect-Oriented Programming. (ECOOP 1997)

Figure 6.3: Example of a SociQL result in tabular form.

6.7.2 Map Queries

Many problem domains have a strong geographic aspect, like the propinquity problem presented in Section 3.1.6. The physical locations of things become very significant to the overall problem. Such domains are best served by visualization tools based on a geo-spatial reference, like a map.

In our map visualization, we show the location of the result objects using positional markers. At the same time, the marker may contain information about the object. Recall *Q5* from Section 3.1.6, where we ask for the geographical distribution of coauthors of Alice. In the first line:

```
MAP r2:name, url
```

it starts with the word `MAP`, indicating that the result of the query will be plotted in a map, then the next token denotes the object represented by the markers, and finally, after the colon appears a list of properties of the object that will appear as the information of the marker.

In the background, map queries are translated to augmented `SELECT` queries of the tabular form, including two properties for the geographic location, namely latitude and longitude. The result of a Map query can be seen in Figure 3.5

6.7.3 Explore Queries

Explore queries provide a graphical representation of the results that makes more evident the relationships between the objects and the importance of the object, inferred based on the number of connections. The graphical representation can also be used as a browser, since every element in the graph can be expanded or contracted by double clicking the element. Figure 6.4 shows the graphical representation of the results returned by query like the one showed in the Tabular queries section.

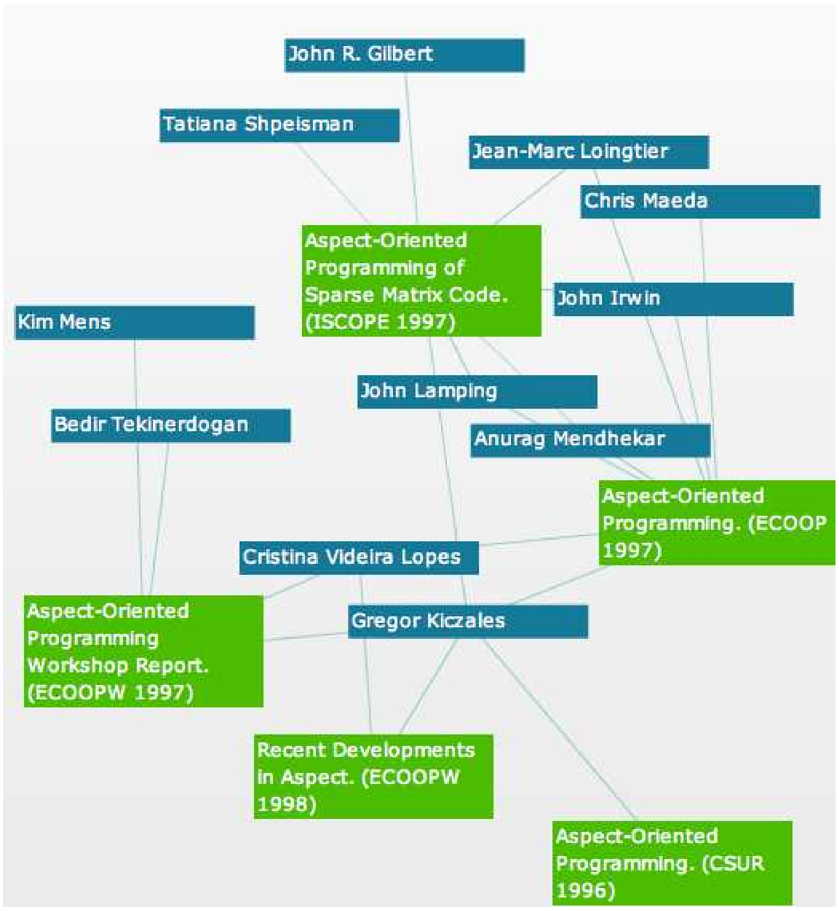
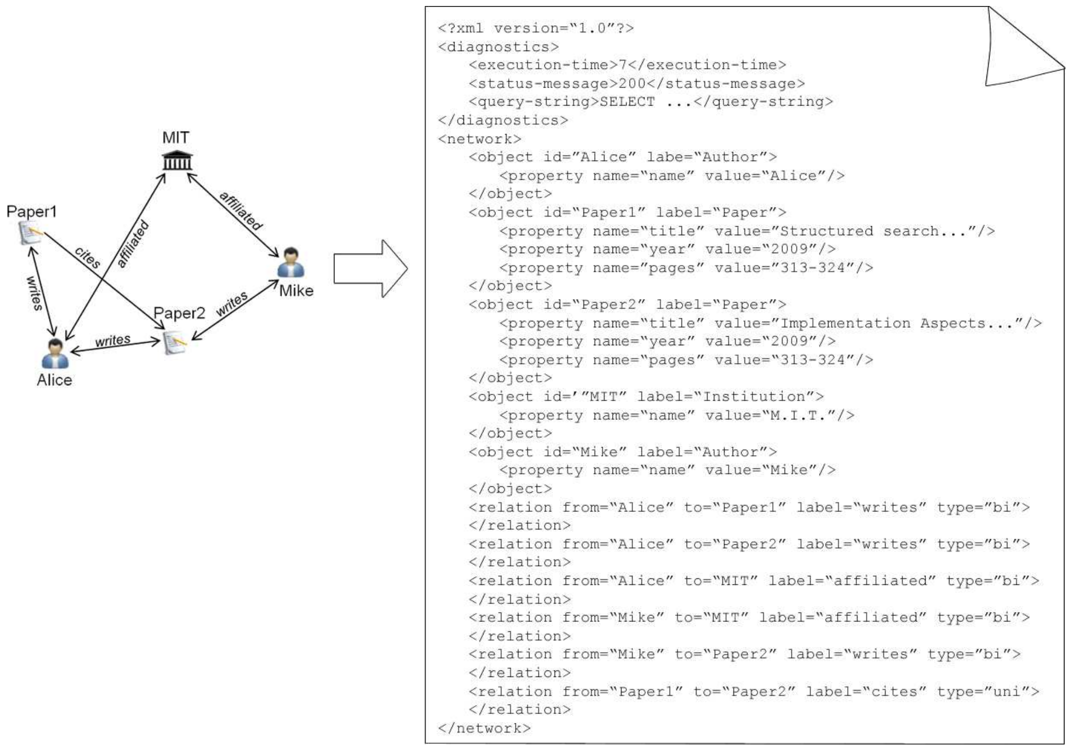

John R. Gilbert
Tatiana Shpeisman
Jean-Marc Loingtier
Chris Maeda
Aspect-Oriented Programming of Sparse Matrix Code. (ISCOPE 1997)
John Irwin
Kim Mens
John Lamping
Bedir Tekinerdogan
Anurag Mendhekar
Aspect-Oriented Programming. (ECOOP 1997)
Cristina Videira Lopes
Aspect-Oriented Programming Workshop Report. (ECOOPW 1997)
Gregor Kiczales
Recent Developments in Aspect. (ECOOPW 1998)
Aspect-Oriented Programming. (CSUR 1996)

Figure 6.4: Result of an Explore query.

In the graph based visualization, the purpose is to present social networks using a familiar node-link representation, where nodes represent social objects, and links symbolize relations. In this view, the squared nodes take different colors depending on the kind of object it represents. The edges also take different colors depending on the type of relation, and moreover, the label of the relation is shown in the edge.

6.7.4 Converting results to XML

As stated in previous sections, the results from SociQL queries are mainly destined to be consumed by a third application, which is our main motivation to convert results to an easily machine-readable language, like XML.



```xml
<?xml version="1.0"?>
<diagnostics>
   <execution-time>7</execution-time>
   <status-message>200</status-message>
   <query-string>SELECT ...</query-string>
</diagnostics>
<network>
   <object id="Alice" labe="Author">
      <property name="name" value="Alice"/>
   </object>
   <object id="Paper1" label="Paper">
      <property name="title" value="Structured search..."/>
      <property name="year" value="2009"/>
      <property name="pages" value="313-324"/>
   </object>
   <object id="Paper2" label="Paper">
      <property name="title" value="Implementation Aspects..."/>
      <property name="year" value="2009"/>
      <property name="pages" value="313-324"/>
   </object>
   <object id='"MIT" label="Institution">
      <property name="name" value="M.I.T."/>
   </object>
   <object id="Mike" label="Author">
      <property name="name" value="Mike"/>
   </object>
   <relation from="Alice" to="Paper1" label="writes" type="bi">
   </relation>
   <relation from="Alice" to="Paper2" label="writes" type="bi">
   </relation>
   <relation from="Alice" to="MIT" label="affiliated" type="bi">
   </relation>
   <relation from="Mike" to="MIT" label="affiliated" type="bi">
   </relation>
   <relation from="Mike" to="Paper2" label="writes" type="bi">
   </relation>
   <relation from="Paper1" to="Paper2" label="cites" type="uni">
   </relation>
</network>
```

Figure 6.5: Example translation of a network to XML.

Consider the query in the Section 6.7.1, which asks for the name of Alice's coauthors and the title of the paper they coauthored. Constructing such results from a relational system follows a straightforward translation in which each object is represented as a direct child of the root node, and at the same time, containing all the properties of the object or the ones requested in the projection of the query (if is a tabular query). The relations are represented as a direct child of the root node, and its properties are represented similarly to object properties. Figure 6.5 shows a conversion from a network to an XML document.

The first child of the resulting XML document presents data about the diagnostics of the query. In Figure 6.5, the diagnostics include the execution time in milliseconds, the status message, and the query that originated the results.

Chapter 7

Conclusions

This section summarizes the particularities of this thesis, the contributions of the query language for social networks, and directions for future work.

7.1 Summary

As we pointed out in Chapter 1, the widespread use of social networking sites has opened new challenges in data management, such as integrated social network analysis and availability of increasing volumes of data. Although several kinds of query languages have been proposed to address these or similar problems, none of them present the versatility and specialized features needed for social networks. In this thesis we have presented SociQL, a query language for social networks grounded on sound sociological theories.

7.2 Contributions

This thesis makes two important, novel contributions to the state of the art in social-network analysis: the first is the SociQL language itself and the second is the prototype implementation of the SociQL query engine.

- Adopting a three-layered model for social-networks data, based on object-centered sociality, presented in Chapter 4, and a familiar SQL-like syntax, we define an expressive query language for social networks, as described in Chapter 5. SociQL incorporates primitives that enable users, who are interested in social-science questions around social networks, to formulate their queries in terms meaningful to them. These primitives explicitly capture notions of

  - special structures of social networks, such as *path*, which produces sequential connections between social objects in the network, and
  - importance of objects in the social network, such as the `FILTER BY` clause, which retrieves a subset of the network, based on centrality metrics on the objects of sociality.

- We have implemented a prototype SociQL query-processing engine, presented in Chapter 6, in order to serve as an integration layer on top of a set of heterogeneous social-networks systems, including ReaSon, our own researcher social network system, Facebook, and DBpedia. As ReaSoN was developed by our own group, its data is accessible through SQL queries to its relational database. On the other hand, the data of Facebook and DBpedia are accessible through the RESTful web services that these systems support. Our SociQL query engine offers

  - a query planner that supports the compilation of queries expressed in any of the three layers of the SociQL model into a composition of ReaSoN SQL queries and Facebook and DBpedia API calls;
  - a mechanism for optimizing the actual execution of the above calls so that the cost of data transfer of the network and data aggregation at the query engine is minimized; and
  - a set of visualizations for the query results, in order to better communicate the information they encapsulate to the user. The first visualization presents data in a generic tabular form and is useful for sorting operations and for programmatically exporting data for further processing.

The second visualization shows physical locations of objects in the results, plotted in a map, enabling the user to gain insights about how spatial proximity might affect social-network relations. And finally, the graph visualization uses a node-link representation to show the network returned by a query, in a manner generally familiar to those who study social networks today.

7.3 Future Work

Currently, SociQL allows to express many useful queries, but there are several enhancements that could improve the applicability of the language. In the rest of this section, we present several proposals to improve the functionality and performance of the query language.

Path expressions used in SociQL queries are useful for general problems, but it is difficult to define complex regular expressions over the structure of the network. A future version of the language should allow the definition of operators in the paths that simplify the statements involving discovery of structures in the network.

Some technical problems need further investigation, like the efficient computation of multiple centrality scores. Centrality measures, such as betweenness, are powerful for identifying central nodes in network analysis, but its computation in large and dynamic social networks is, usually, costly. However, many studies have proposed methods to efficiently compute centrality measures, using approximations or parallel computations. These methods can be, in principle, easily incorporated to SociQL, since the actor centrality algorithms are orthogonal to the implementation of the language.

External sites exhibit limitations in the number of invocations by minute, the size of the result documents, and in some cases, restrictions due to the calculated complexity. In view of the problems that a query involving external datasources can lead to, in future versions, we have to improve processing time and recall, by using a model of query planning having as goal the minimization of web service invocations and the maximization of relevant tuples returned by service invocation.

Current research in sociology exhibits a strong temporal aspect. In the future, SociQL should consider the evolution of social networks with temporal information about object and relation arrivals. Along with time considerations, the language will have to contemplate more structure primitives, in order to evaluate properties, such as cohesion or clustering coefficient, with respect to other nodes in the network.

Another direction would be the parallelization of the query execution. The subqueries composing a SociQL statement could be parallelized by submitting queries to external sources in parallel, and by separating the execution of the centrality network from the query itself.

Bibliography

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Rebecca G. Adams. *Placing Friendship in Context*. Cambridge University Press, 1999.

[3] Aris Anagnostopoulos, Ravi Kumar, and Mohammad Mahdian. Influence and correlation in social networks. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 7–15, New York, NY, USA, 2008.

[4] J. M. Anthonisse. The Rush in a Graph. *Amsterdam: Mathematische Centrum*, 1971.

[5] Gustavo O. Arocena and Alberto O. Mendelzon. Weboql: Restructuring documents, databases, and webs. In *Proceedings of the Fourteenth International Conference on Data Engineering*, ICDE '98, pages 24–33, Washington, DC, USA, 1998.

[6] Elliot Aronson, Timothy D. Wilson, and Robin M. Akert. *Social Psychology*. Prentice Hall, fifth edition, February 2004.

[7] Sren Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *6th International Semantic Web Conference*, pages 11–15, Busan, Korea, 2007.

[8] Albert-László Barabási. *Linked: The New Science of Networks*. Basic Books, 1st edition, May 2002.

[9] Vladimir Batagelj, Vladimir Batagelj, Andrej Mrvar, and Andrej Mrvar. Pajek - analysis and visualization of large networks. In *Graph Drawing Software*, pages 77–103, 2003.

[10] A. Bavelas. A mathematical model of Group Structure. *Human Organizations*, 7:16–30, 1948.

[11] Murray A. Beauchamp. An improved index of centrality. *Behavioral Science*, 10(2):161–163, 1965.

[12] Morroe. Berger and R. M. MacIver. *Freedom and control in modern society*. Octagon Books, New York, 1964.

[13] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, pages 34–43, May 2001.

[14] Uldis Bojars and Alexandre Passant. Sioc project: Semantically interlinked online communities. In *Proceedings of the Second Multi-Agent Logics, Languages, and Organisations Federated Workshops*, MALLOW '09, pages 7–10, Turin, Italy, 2009.

[15] Phillip Bonacich. Factoring and weighting approaches to clique identification. *Journal of Mathematical Sociology*, 2:113–120, 1972.

[16] Stephen P Borgatti, M G Everett, and L C Freeman. Ucinet for windows: Software for social network analysis. *Harvard Analytic Technologies*, 2006, 2002.

[17] Stefano Bortoli, Heiko Stoermer, Paolo Bouquet, and Holger Wache. Foaf-o-matic - solving the identity problem in the foaf network. In *Proceedings of the Fourth Italian Semantic Web Workshop*, SWAP 2007, pages 130–139, Bari, Italy, 2007.

[18] Dan Brickley and Libby Miller. Foaf vocabulary specification 0.98. Retrieved 15 September, 2010, from `http://xmlns.com/foaf/spec/20100809.html`, 2010.

[19] Anne Asserson Brigitte Jrg, Keith Jeffery. Cerif 2008 1.2 (core) semantics. Retrieved 20 September, 2010, from `http://www.eurocris.org/Uploads/Web%20pages/CERIF2008/CERIF2008_1.2_CORE-Semantics_FirstDraft.pdf`, 2010.

[20] Peter J. Carrington. *Models and methods in social network analysis*. Structural analysis in the social sciences. Cambridge University Press, 2005.

[21] K. K. Cetina. Sociality with objects: Social relations in postsocial knowledge societies. *Theory, Culture & Society*, 14(4):1–30, November 1997.

[22] Surajit Chaudhuri, Gautam Das, Vagelis Hristidis, and Gerhard Weikum. Probabilistic ranking of database query results. In *Proceedings of the Thirtieth international conference on Very large data bases*, VLDB '04, pages 888–899, Toronto, Canada, 2004.

[23] Nicholas A. Christakis and James H. Fowler. *Connected: The Surprising Power of Our Social Networks and How They Shape Our Lives*. Little, Brown and Company, September 2009.

[24] CYRAM Co. Ltd. Netminer - social network analysis software. Retrieved October 8, 2010, from `http://www.netminer.com/NetMiner/home_01.jsp`, 2008.

[25] Bruce D'Arcus and Frdrick Giasson. Bibliographic ontology specification. Retrieved October 8, 2010, from: `http://bibliontology.com/specification`, 2009.

[26] Facebook. Fql facebook developers wiki. Retrieved March 5, 2010, from: `http://wiki.developers.facebook.com/index.php/FQL`, 2007.

[27] Linton C. Freeman. Visualizing social networks. *Journal of Social Structure*, 1(1), 2000.

[28] Veselin Ganev, Zhaochen Guo, Diego Serrano, Brendan Tansey, Denilson Barbosa, and Eleni Stroulia. An environment for building, exploring and querying academic social networks. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, MEDES '09, pages 282–289, Lyon, France, 2009.

[29] O. Hassanzadeh and M. P. Consens. Linked movie data base. In *Proceedings of the WWW2009 workshop on Linked Data on the Web*, LDOW '09, Madrid, Spain, 2009.

[30] Jason J. Jung and Jérôme Euzenat. Towards semantic social networks. In *Proceedings of the 4th European conference on The Semantic Web*, ESWC '07, pages 267–280, Berlin, Heidelberg, 2007.

[31] Charles Kadushin. *Making Connections: An Introduction to Social Network Concepts, Theories and Findings*, chapter Some Basic Network Concepts and Propositions. Oxford University Press, 2011. In-press.

[32] Young A. Kim and Jaideep Srivastava. Impact of social influence in e-commerce decision making. In *Proceedings of the ninth international conference on electronic commerce*, ICEC '07, pages 293–302, New York, NY, USA, 2007.

[33] Graham Klyne and Jeremy J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. Retrieved 21 April, 2010, from the World Wide Web Consortium site: `http://www.w3.org/TR/rdf-concepts/`, February 2004.

[34] Jure Leskovec and Eric Horvitz. Planetary-scale views on a large instant-messaging network. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 915–924, Beijing, China, 2008.

[35] Chen Li, Ramana Yerneni, Vasilis Vassalos, Hector Garcia-molina, Yannis Papakonstantinou, Jeffrey Ullman, and Murty Valiveti. Capability based mediation in tsimmis. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, SIGMOD '98, pages 564–566, Seattle, Washington, 1998.

[36] Wen-Syan Li, Junho Shim, K. Selcuk Candan, and Yoshinori Hara. Webdb: A web query system and its modeling, language, and implementation. In *Proceedings of the Advances in Digital Libraries Conference*, ADL '98, page 216, Washington, DC, USA, 1998.

[37] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27(1):415–444, 2001.

[38] Alberto O. Mendelzon, George A. Mihaila, and Tova Milo. Querying the world wide web. In *Proceedings of the fourth international conference on on Parallel and distributed information systems*, DIS '96, pages 80–91, Washington, DC, USA, 1996.

[39] Stanley Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.

[40] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[41] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. Retrieved 13 March, 2010, from W3C site: `http://www.w3.org/TR/rdf-sparql-query/`, 2008.

[42] R. Ronen and O. Shmueli. Soql: A language for querying and creating data in social networks. In *Proceedings of the IEEE 25th International Conference on Data Engineering*, ICDE '09, pages 1595–1602, 2009.

[43] G. Sabidussi. The centrality index of a graph. *Psychomatrika*, 31:581–603, 1966.

[44] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: a core of semantic knowledge unifying WordNet and Wikipedia. In *Proceedings of the 16th International World Wide Web Conference*, WWW '07, pages 697–706, Banff, Canada, 2007.

[45] Yonghong Tian, Jaideep Srivastava, Tiejun Huang, and Noshir Contractor. Social multimedia computing. *Computer*, 43:27–36, 2010.

[46] Sandra A. Vannoy and Prashant Palvia. The social influence model of technology adoption. *Commun. ACM*, 53:149–153, June 2009.

[47] S. Wasserman and K. Faust. *Social network analysis: Methods and applications*. Cambridge University Press, 1994.

[48] Yahoo. Query language guide. Retrieved 4 April, 2010, from `http://developer.yahoo.com/yql/guide`, 2008.

Appendix A

Syntax of the Language

The SociQL syntax is inspired from the syntax of the standard SQL `SELECT` statement. Our language is designed exclusively to compute information, therefore it does not include any other data manipulation constructs from SQL, such as `INSERT`, `UPDATE` or `DELETE`. In our implementation of the SociQL query-processing engine, we also include three new constructs in addition to the standard `SELECT` statement, which can be used to post-process the network produced by the `SELECT` statement. These are `ORDER BY`, `MAP` and `EXPLORE`. This post-processing phase augment the `SELECT` results so that that may be visualized, and interactively explored by users, in a custom way. Their meaning will be discussed in the next chapter, where we discuss details of the implementation and extensions to the language. The BNF specification of the language is shown in the Listing A.1 and A.2.

Listing A.1: BNF Specification

```
selectQuery ::= SELECT relationList FROM network WHERE
   predicateList [FILTER BY filteringPredicate] [LIMIT
   number]

network ::= ( selectQuery ) | networkElemList

property ::= objectAlias . propertyName
relationList ::= relationProjected | relationList ,
   relationProjected
relationProjected ::= relationName ( objectAlias ,
   objectAlias ) | PATH ( objectAlias , objectAlias)
```

Listing A.2: BNF Specification (continued)

```
networkElemList ::= networkElement | networkElemList ,
   networkElement
networkElement ::= object | relation
object ::= [layer] objectName objectAlias
relation ::= [layer] simpleRelation | [layer] propRelation |
    pathRelation
simpleRelation ::= relationName ( objectAlias , objectAlias
   )
propRelation ::= relationName . relationProperty (
   objectAlias , objectAlias , relationVariable )
pathRelation ::= NEIGHBORHOOD ( objectAlias , objectAlias,
   number [, relationType] )
layer ::= layerIdentifier .
layerIdentifier ::= CON | NET | DATA

predicateList ::= predicate | predicateList AND predicate
predicate ::= relationVarPredicate | conditionPredicate
conditionPredicate ::= property operator value | property
   operator property
relationVarPredicate ::= relationVariable operator value

filteringPredicate ::= ( centralityMeasure OF objectName ON
   relationNameList ) operatorOrdinal number
relationNameList ::= relationName | relationNameList ,
   relationName
centralityMeasure ::= INDEGREE | OUTDEGREE | DEGREE |
   PAGERANK | CLOSENESS | BETWEENNESS
operator ::= operatorOrdinal | operatorNominal
operatorOrdinal ::= = | != | > | >= | < | <=
operatorNominal ::= >< | <>

objectAlias ::= id
objectName ::= id
propertyName ::= id
relationName ::= id
relationType ::= id
relationVariable ::= id

number ::= digit | number digit
digit ::= [0-9]
letter ::= [a-Z]

id ::= letterrestofid | _ restofid
restofid ::= | validchar | restofidvalidchar
validchar ::= letter | digit | _
```