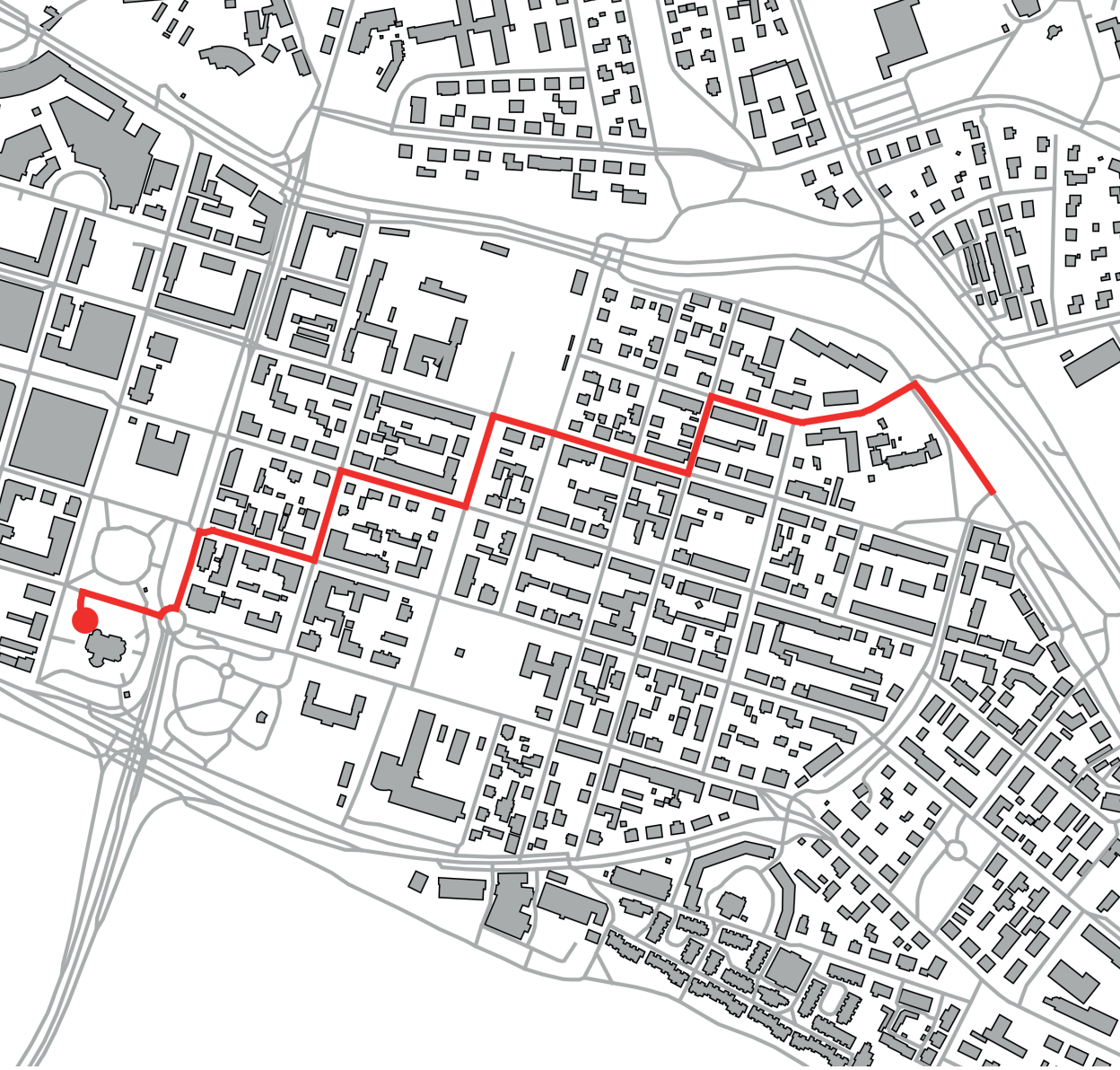

Natural Language Interfaces over Spatial Data: Investigations in Scalability, Extensibility and Reliability

Johan Mollevik

Department of Computing Science
Umeå University, Sweden 2013

Natural Language Interfaces over Spatial Data:

Investigations in Scalability, Extensibility and Reliability

Johan Mollevik

Licentiate Thesis, November 2013
Department of Computing Science
Umeå University
Sweden

Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden

mollevik@cs.umu.se

ISBN 978-91-7459-777-6
ISSN 0348-0542
UMINF 13.22

Printed by Print & Media, Umeå University, 2013.

Abstract

This thesis focuses primarily on constructing voice-only pedestrian guidance systems using spatial database techniques. In the process of doing this we first explored how to use authoring tools to build natural language interfaces over large databases. Specifically we built a natural language interface over the MusicBrainz database of 1.5GB and confronted the resulting scalability issues. We then explored vague querying, specifically spatial queries using 'near'. Assuming 'language as a set of conventions', we proposed an approach for handling vagueness by defining contexts that are compiled to crisp SQL view definitions. In our recent work, as partners in the Spacebook project (`http://www.spacebook-project.eu`), we have focused on how to build reliable, scalable and extensible text-to-speech (TTS) based navigation systems for pedestrians. Technical aspects we have worked on include building the system Janus (`http://janus-system.eu`), with sensor reports and bidirectional voice channels. Experimental work has been mostly focused on measuring accuracies and latencies with available hardware. We have also, very recently, started human usability experiments. Our theoretical work has been in defining models for how a system can interact with pedestrians over high latency data links with poor GPS quality using prediction of pedestrian positions and scheduling utterances to mask latencies. To allow for scalable deployment, we have only used standard smart phones and inexpensive servers.

Preface

This thesis consists of an introduction and five papers. In the introduction background and earlier works in natural language interfaces, spatial databases and pedestrian navigation systems are discussed as well as some of the practical contributions of this thesis and a direction for future work. The papers making up this thesis are the following:

Paper I — Johan Granberg[1] and Michael Minock, "A Natural Language Interface over the MusicBrainz Database", *proceedings of the first Workshop on Question Answering over Linked Data (QALD-1),* pages 38-43. May 2011.

Paper II — Michael Minock and Johan Mollevik. Context-dependent 'near' and 'far' in spatial databases via supervaluation. *Journal of Data and Knowledge Engineering (DKE), Elsevier 86:295-305,* 2013

Paper III — Michael Minock, Johan Mollevik and Mattias Åsander: "Toward an Active Database Platform for Guiding Urban Pedestrians" *Technical Report Umeå Universiy UMINF-12.18* October 2012

Paper IV — Michael Minock, Johan Mollevik, Mattias Åsander and Marcus Karlsson: "A Test-Bed for Text-to-Speech-Based Pedestrian Navigation Systems." *proceedings of the International Conference on Applications of Natural Language to Information Systems (NLDB),* pages 396-399. June 2013

Paper V — Michael Minock and Johan Mollevik. "Prediction and scheduling in navigation systems." *In Proceedings of the Geographic Human-Computer Interaction (GeoHCI) workshop at CHI,* April 2013

[1] Name changed from Granberg to Mollevik in 2012 due to marriage.

Acknowledgments

During my work with this thesis there are a lot of people who have helped and supported me. I would like to thank my supervisor Michael Minock for having faith in me and for hearing me out and listening to my ideas. I would also like to thank my co-supervisor Henrik Björklund, my colleagues Mattias Åsander and Marcus Karlsson working with me on Spacebook in Umeå, the people in the NFL research group and the department's support group. Thanks also go to the Department of Computing Science for being a good workplace with friendly colleagues, the European Community's 7th Framework Programme and the Kempe Foundation for funding this research. Finally I would like to thank my friends and family for their continuous support and especially my wife for being wonderful.

Contents

Chapter 1

Introduction

This thesis explores how to build *reliable*, *scalable* and *extensible* natural language interfaces to databases, focusing mostly on spatial databases. Natural language interfaces (NLIs) have many applications in our everyday lives, for example when we want to control computers with our eyes and hands free or in cases where we want to avoid learning yet another cryptic user interface for some rarely used system. To illustrate, consider the applications demonstrated in the papers. In the MusicBrainz domain we allow for searches of music meta-data. Here a good natural language interface could allow for construction of playlists using either voice or text input. Or for more leisurely browsing of music meta data (See fig 1.1 for an example interface). For example, answering questions like who performs on the currently playing music track, where the track itself is queried from the user's music player.

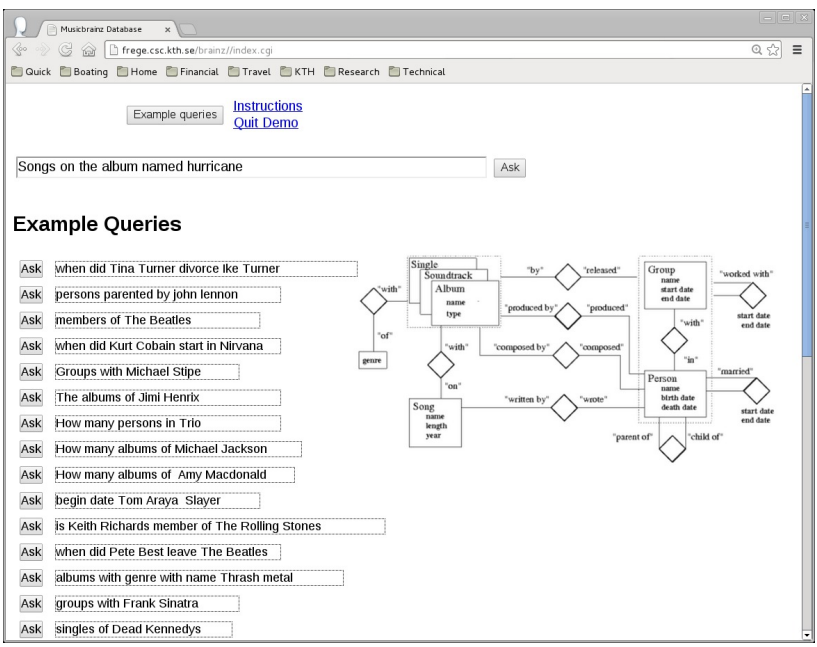

Figure 1.1: A screenshot of our MusicBrainz interface

In the spatial domain, our primary focus, two use-cases have been more closely studied. The first one is the issues of vagueness in geographical information (GIS) systems. Specifically capturing the meaning of 'near' is hard; there is no hard limit when things are not near any more. The crisp counterpart 'nearest' is more straightforward and is the focus of much recent research in spatial databases. The

second use-case, a tourist application, is an NLI based guide using audio as the sole means of user communication. This allows the users to keep looking at attractions around them while simultaneously querying about their surroundings. They can either ask for information about what they see or to ask for route guidance (See figure 1.2). In this domain an NLI can also be envisioned as an interface for a GIS system.

Figure 1.2: A pedestrian using our natural language guidance system

All of the above use cases have similar requirements: 1.) the systems in question must be very *reliable*. A system that does not work half the time will send the users back to manually clicking in their GUI application or to take up their tourist map; 2.) such systems need to be *scalable*. They should work on large datasets such as a map of an entire city or over music collections of several terabytes. This also means that such systems should be able to run on hardware that is available to the user, such as old home PCs or cheap smart phones. At the same time the system must not require massive servers that do all work remotely; 3.) These systems must be easily *extensible*. If the systems are not extensible it will be very hard to correct their limitations. And limitations will arise when users use the systems in unexpected ways. This rules out designs that can not handle, for example, new types of objects that the user wants to find or new categories of meta information. Of course the new functionality has to be implemented but *old* functionality should not have to be rewritten.

While there are more things that are important for such systems to succeed these are the three that are the focus of my work.

Chapter 2

Natural Language Interfaces

Natural Language Interfaces (NLIs) are interfaces where the computer understands human language rather than humans learning the computer's interface. When building a natural language interface there are two directions to consider, the *analysis* direction where the system analyses what the user wrote or said and the *generation* direction where the system conveys relevant information in natural language. Work on NLIs has been conducted since the late sixties [2] with LUNAR [45] and RENDEZVOUS [10] being well known examples. More recent examples are TEAM [17], ORAKEL [9], C-PHRASE [32, 29] and, very recently, the multi-domain capable WOLFRAM ALPHA [26]. (`http://www.wolframalpha.com/`)

One thing that remains important when constructing NLIs is restricting the interface to a limited domain [11, 30]. Copestake states in [11] that to use the restricted domain property the domain must be communicated to the user in a clear way. Failure by the user to understand the limitations of the domain will result in both questions the NLI can not handle and users that avoid asking questions that the NLI could manage. It is also stated that communicating the limitations is the correct solution, the other alternative of adding information from neighbouring areas leads recursively to the need to cover all of English. The size of the domain can be restricted in different ways. One can restrict the types of things the system knows about, mapping to verbs and nouns etc. The work in [30] discusses the requirements of viable applications of restricted domain natural language interfaces on today's web. Additionally one can restrict the types of sentences the system can handle. In [8] Boye and Wirén argue that by limiting expressiveness to a semantic less powerful than first order logic robustness can be increased while retaining enough expressive power to be usable.

For the generation direction of an NLI there is a need to decide how the output is going to be generated. The simplest solution, which is often sufficient, is to simply use pre-written text templates. This works if the types of things the user can ask for are few, for example if the only thing the user can ask for is upcoming flights between different airports. If the output is more complex, other strategies will be needed. Reiter and Dale [36], propose an architecture consisting of three phases, document planing, micro planing and surface realization. The idea is to separate the decisions of which communication goals to accomplish, what information should be used to accomplish them and how to fit that into a grammatical sentence in a natural language. This decomposes one hard problem into three easier problems. Of course there is more to it than that, the decomposition is not random but is derived from careful design.

2.1 Natural language interfaces to databases

Databases are a convenient data store for a natural language interface as well as providing well defined query languages to compile user request into. As such natural language interfaces to databases are an important sub-domain of natural language interfaces in general. One motivation for the creation of natural language interfaces to databases is to allow humans to explore structured data without learning a query language. Additionally almost any computer system can be made to use database tables to represent input and output. For example many basic applications can be modeled as an append only database with tables for input, output and system logs. Also note that database in this context need not refer only to SQL databases, other forms of structured data such as XML can employ the same or similar techniques.

When creating a natural language interface to a database the parsing of the human's request has far more potential for ambiguity than the computer's response, which is more directly under the system builder's control. In the generation direction the system can use standardised language to ensure that the natural language response answers the question as the system understood it. The system can also use query paraphrase techniques (See [10, 31]).

The analysis direction is more difficult. The system has to understand the user's request as best as it can and answer the question it believes it got or engage in clarification dialog. The measures of *precision*, *recall* and *willingness* [32] , defined below, can be used to measure the quality analysis.

$$willingness = \frac{\#correct + \#incorrect}{\#total}$$

$$precision = \frac{\#correct}{\#correct + \#incorrect}$$

$$recall = willingness * precision = \frac{\#correct}{\#total}$$

Willingness is the percentage of requests the system answers, in this case an answer 'I did not understand that' does not count, only answers to the actual question. *Precision* on the other hand concerns itself only with the answered queries and is the percentage of those that where correctly answered. *Recall* is the product of the two and is the percentage of the questions that were answered correctly. To maintain user trust, the system precision should ideally be one. In any other scenario the user gets the wrong answer without any system indication that anything is wrong. A low willingness on the other hand can cause frustration as the system rejects a large number of requests out of hand.

In trying to design NLI frontends to databases that have high precision,willingness and recall, the community has come up with a number of approaches to the input analysis problem.

Some early experiments used what we can call the *pattern matching approach*. The idea is that there are a number of patterns the system understands. An

example would be a city name followed by a country name, which is interpreted as a question whether the city is in the country. A common technique is to ignore some words such as 'the' and 'in' if they do not exist in the pattern. Doing this will increase recall but at the risk of decreasing precision. By creating a large number of such rules the builder of such an interface can make the system give reasonable answers within its domain [2], however such a large number of rules are costly to build, especially for larger databases. And of course the approach has difficulty with more syntactically complicated sentences as well.

A more sophisticated approach is used in *Syntax based systems* which work by parsing the input sentence into a parse tree. Each node in the tree is then mapped into a query language expression. An example of this approach is LUNAR [45] from 1972. A problem with the approach is that domain specific mappings are usually needed as it is hard to map to general languages like SQL [2].

A similar approach to the *Syntax based systems* is the *semantic grammar approach*. Like the *Syntax based systems* they use a parse tree. Unlike the *Syntax based systems* the parse tree is not structured based on the word classes of the natural language but on structures of a formal language query language used to query the database. A problem with these systems is the need to construct new rules when switching domains, as the rules for parsing becomes tied to the domain by the coupling of natural language with a formal language grammar specialized to the domain. An early system of this type was LADDER [23]. More recently we have work by Mooney and Wong on the λ-WASP algorithm [44] and the C-PHRASE system [32, 29].

In an attempt to solve the problems of being tightly tied to the domain, the *transportable approach* was conceived. The idea here is to do the analysis in two steps. First the query is translated into a logical form independent of databases. This allows the use of wide coverage parsers built by professional linguists. This logical form is then, in a second step, transformed into a query for a specific database. Two examples of these types of systems are CLE [1] and TEAM [17].

Apart from the classifications of pattern matching approach, syntax based system, semantic grammar approach and transportable approach there is an orthogonal classification. Here we make the distinction between *rule-based systems* and *statistical systems* [14]. Rule based systems (like C-PHRASE [32, 29]) are systems where a developer has to manually write the rules for the system. The rules might be at any level of abstraction and there might be limited auto generation but ultimately it is the system designer/developer/operator who decides which rules are included [33]. By contrast, a statistical system (like λ-WASP [44]), which might be based on rules, is constructed automatically. In statistical systems a set of examples is presented to the system which uses them for training. In the same sense as a rule based system might have some automation, a statistical system might have some hand crafted rules. An example of this is to manually encode the rules for function words in the target language while letting the system learn the domain vocabulary. See [16] for a study in which we attempted to replicate results in [44].

In the work in this thesis we have focused on rule based, semantic grammar type systems. They are easier to control than the statistical variant making them more reliable. They are also computationally faster as the rule bases tend to be smaller. Finally they are more predictable, thus extending them it is less likely to break what is already working. The negative side of this trade-off is that rule based systems take longer to construct than statistical systems. We argue that this exchange of construction time versus extensibility, scalability and reliability is worth the cost. We expect users to be more annoyed by system bugs than by smaller domains (due to construction costs). This has not, as of yet, been validated empirically. The choice of a semantic grammar approach as opposed to a transportable approach is motivated by having a better knowledge of semantic grammar type technology and not seeing compelling evidence that the transportable approach is better with current wide coverage parsers.

The main contributions this thesis makes to the area is the following. Demonstrating a way to quickly build an NLI over a large database using authoring (see Paper I). We are conducting evaluations on how to build a minimal NLI to guide pedestrians efficiently (See Paper IV). Apart from this we have been involved in the development of an meaning representation language (MRL) to handle dialog in a tourist guidance context (see section 5.3) in collaboration with the authors of [43].

Chapter 3

Spatial Databases

Figure 3.1: The Bedolina map, a stone carving from around 1500BC picture Luca Giarelli / CC-BY-SA 3.0

Historically the format for storing geo-spatial data has been maps. While we can not say for sure when the first map was created, we know of maps from about 1500BC from finds in Val Camonica in northern Italy [6] (See figure 3.1). The maps carved into stone at this site show houses, fields and irrigation channels represented as outlines clearly recognizable as a map of a village. Since then, numerous advances have been made in cartography. During the renaissance various projections were used to represent the earth with more mathematical accuracy [25] (See figure 3.2 for comparison with a map in an older style). In more modern times other types of spatially related data have been developed. For example, in the sixties, the US Bureau of the Census operated in a mail out/mail in manner [12]; they had a lot of data that was spatially related, keyed by postal address.

An early recognition of the possibilities of using computers to store, process and display geo-spatial data comes from Tobler in 1959 [42]. He states that

> It seems that some basic tasks, common to all cartography, may in the future be largely automated, and that the volume of maps produced in a given time will be increased while the cost is reduced.

Early systems [12] working with spatial data were mostly driven by the need to solve large tasks using whatever technology was at hand despite its limitations.

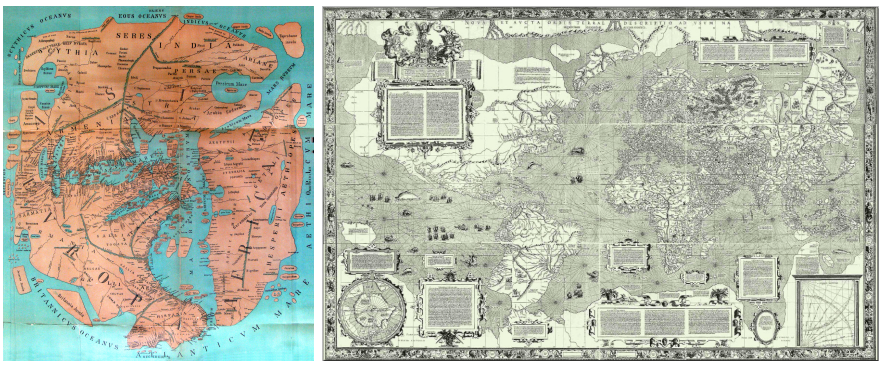

Figure 3.2: (left) A recreation of a map by roman cartographer Pomponius Mela. (right) The Mercator world map of 1569.

The trend was to implement a system to solve a task that was otherwise infeasible and then leave it as a legacy system after the task was done. In short spatial data processing was a means to an end, not the focus of research in its own right.

It is unclear exactly when the term 'spatial databases' was coined, but in a paper from 1994 Güting [18] defines the following properties needed for a system to be a *spatial database system.*

- A spatial database system is a database system.
- A spatial database offers spatial data types (SDTs) in its data model and query language.
- A spatial database supports spatial data types in its implementation, providing at least spatial indexing and efficient algorithms for spatial join.

Given that we have these features, a host of spatial problems becomes easily expressible.

An example would be the following query "what's the longest any resident has to walk from home to the closest bus stop". As long as we have a digital map of the city including information about bus-stops and which buildings are homes and what paths are walkable, we can compute the answer with a spatial database. Or similarly "which street has the most cramped households" can be answered if you have a city map where the area of homes is apparent and some non-spatial data of how many residents each house have.

A key issue however is that queries should be declared declaratively. While Güting does not declare that the database must be relational, we, and many others, see no viable alternatives. Let us review what it means for a database to be relational.

For a database to be relational it has to have at least the following properties:

- Must store its data as a set of relations (also called tables).

- Each relation is a set of tuples (also called rows) of the same structure and with the same names. That means that all tuples in the same relation have the same number of elements and that elements named the same are of the same type in all tuples in the same relation. (Tuple elements sharing a name in a relation form a column.)
- The database system must be able to join together relations to form new relations at runtime and filter them by specifying constraints on the tuples in the computed relation.
- It should be possible to specify constraints on relations such that the database system can guarantee that the tuples stored in them do not violate these constraints. Constraint types should include at least:
  - Primary key constraints: the specified columns in each tuple are enough to uniquely distinguish the tuple among all tuples in the relation.
  - Foreign key constraints: the specified columns have the same value as the primary key of some existing tuple in a specified relation.

While not strictly needed by the definition most major database systems also include the following for performance and ease of use:

- Indices: a device to speed up queries, works by taking a subset of the columns in a relation and computing a typically tree-base data structure in some ordering that makes joins and/or condition testing faster. It is worth noting that the database system figures out by itself if using an existing index will speed up a query. Commonly the developer is responsible for creating the indices.
- Server-side functions: having code run within the database system allows for more complex queries to be written and thus simplify development.

A query to the database engine is *declarative*. The developer does not have to bother with how the database will access the underlying storage nor with enforcing that the constraints are not being violated on data modifications. This is a huge strength of relational databases, the fact that the developer can formulate his query in a relatively compact manner and know that the result is correct. In the absence of a declarative approach the developer has to specify the disk or memory storage formats himself, implement search algorithms himself, check consistency himself.

Looking more closely at queries we can discern the following common cases:

- Singleton query: the query returns one or zero tuples as its result. In some database systems you can use the result of a singleton query as a value in a condition, allowing for more complex queries.
- Bounded queries: these queries return between zero and some fixed number of results, a special condition is formulated that restricts the number of

returned tuples to at most that amount. These are interesting as they can often be optimized to run faster. For example if the database system has an index over salaries ordered from highest to lowest, and is asked to retrieve the ten highest salaries, it only has to pick them directly from the top of the index instead of searching the entire index.

- Range queries: some conditions are specified to return the tuples whose value of some columns is between a lower and upper bound. These too can take advantage of indices.

When extending a relational database system to be spatial we have to, according to Gütings criteria, add spatial types and indices as well as efficient spatial joins. This can be accomplished by adding the following.

- Data types for points, linestrings and polygons.
  - A point can be represented as a pair of floats. (or triple if the database is working with 3D data)
  - A linestring is a sequence of connected lines, the end of the first connecting to the beginning of the next one and so on.
  - A polygon is an area. For 2D this can be represented as linestring that has the same startpoint as endpoint and does not cross its own path.
- Indices that work with points, linestrings and polygons. The common types are R-Trees [19] or GIST [22] indices.
- Efficient algorithms for joins filtered by distance between spatial objects, spatial object containment and spatial object overlap.

Something that is also desirable to add to the spatial database system, is support for the following query types:

- *k-nearest-neighbour* (knn) queries: this is a type of bounded query which returns the K tuples that are closest to some reference geometry, while also matching other filter conditions.
- *Route queries*: this query type will return a linestring from an original point to a point matching some goal condition, while observing conditions on which path it may take. For example, one such condition might be, 'must not intersect any polygon in the building table'. Or 'must be contained by the linestrings in the road table'. In this later case A* [21] is a suitable algorithm to implement this query.

Let us look at the example "what's the longest any resident has to walk from home to the closest bus stop". This query can be encoded as: take the relations containing residences and for each of them, make a (knn) query against the relation containing bus stops, filter the resulting table to find the maximum distance. This will be the maximum distance of any resident to their closest bus stop. Note

that strictly speaking distance should be related as *network distance*. All to often *euclidian distance* is used in practice.

To work with spatial data it is of course not enough to have a spatial database, of equally large importance is sources of 'good' data. The determination of 'good' is complex. For example the following can affect how 'good' data is in a given application:

- Does it support 2D or 3D?
- What is the granularity? (only buildings or everything down to every lamp post, does it include trees and other natural features)
- How many objects of represented types are missing?
- What is the accuracy of object placement?
- How small details of an object are discernible?
- How old is the data?
- Is temporal data present? i.e. can we see what it looked like in the past?
- Is the data being updated, how often?
- Is the data consistently represented?
- Is the data freely available?

The work in Paper II-V has used OpenStreetMap [20] data (See figure 3.3 for an example rendering of this data). OpenStreetMap is a project to provide geographical data under a licence that allows anyone to use it. It is built by volunteers contributing data. In many cases by going around their neighbourhood with a GPS and annotating the data points gathered, improving the map quality of their local area. OpenStreetMap provides 2D map data and in some cases simple 3D data (building heights and which road is an overpass/underpass). It has an accuracy for object placement similar to consumer GPS devices with a human doing sanity checks meaning it is quite good for the pedestrian navigation we are doing. OpenStreetMap also covers a huge range of object types including pedestrian crossings, ATMs and bus stops which have been useful. On the negative side, from a pedestrian guidance point of view OpenStreetMap does have a lot of objects missing from its data, how much varies with location, but outside central areas in many cases only streets are present.

Another data source we consider is the government, for example United Kingdoms Ordinance Survey and Swedish Lantmäteriet have similar data available. In both cases they have maps of all buildings and streets in the area and in addition to 2D information height above the sea level are available. On the negative side many small details are not present such as ATMs and benches. The resolution for height data is about 1 point every 2×2m which is slightly too low to accurately detect sidewalks and similar features. A major problem with these kinds of data sources

Figure 3.3: Renderings of OpenStreetMaps data (left) Stockholm (right) Umeå

is that they are not freely available. Pricing may vary and getting free access for research is feasible but the data can not be freely redistributed.

The spatial database implementation we have used in this work is PostGIS [35] which is an extension that spatially enables PostgreSQL [37].

The contributions of this thesis to this area are mainly: A method for storing and querying vague data including spatial vagueness in databases in Paper II. A tool for plotting the result of SQL queries against a PostGIS database described in more detail in section 5.2. And ideas of how to use bi-temporal databases for working with pedestrian movement prediction in Paper V and section 6.1.

Chapter 4

Pedestrian Navigation Systems

Early work on computer based navigation systems focused on vehicle navigation. This includes the Electronic Route Guidance System (ERGS) [39] developed in 1970. Since then the number of car navigation systems have become ubiquitous with commercial systems from, for example, TomTom. Then, as well as now, the goal of a navigation system is to guide its user in an efficient and safe manner to some target destination. When the mode of transportation is automotive this consists of advising the driver which turns to take and in more advanced systems which lanes to be in. Typically this is done with a device displaying a map fixed to the cars dashboard augmented with voiced instructions.

For a system to know user location some kind of position sensor is required. In early work or work focusing on indoor scenes, a variety of devices are employed, passive ones such as accelerometers or local radio beacons or active ones such as radar, lidar or sonar. For outdoor use, global navigation satellite systems (GNSS) like NAVSTAR GPS or GLONASS are a cheap choice (See figure 4.1 for pictures) Even with their inaccuracies, GPS devices are a huge improvement over earlier alternatives. When hand-held GPS (globally available in 1994) receivers became commercially available in the late eighties/early nineties the accuracy was orders of magnitude better than the alternative. And this was despite the US policy to slightly degrade the GPS signals for civilian use. Since the degeneration of signals has been suspended, GPS devices are now accurate to within a few meters, but there is a catch. This applies if the receiver is good enough and with a reasonably clear view of the sky. Among high buildings the signals bounce reducing accuracy. To get higher precision GNSS can be complemented with other sensors, doing this can yield a very accurate position, but tends to require expensive sensors. GPS devices are well suited for pedestrian navigation as modern GPS devices are cheap, small and light, as they are built into mobile phones.

When looking at pedestrian navigation as opposed to vehicle navigation, the problem gets more challenging. In general vehicles turn rather slowly and have a reasonably predictable speed. In most cases vehicle guidance needs relatively low precision position measures. A car driver will not be overly inconvened if his navigation system believes his position is 15m from where he is. For a pedestrian the same error can cause major problems. Errors on that scale can for example mean that the system thinks the pedestrian is on the other side of a building. The pedestrian's ability to stop or change movement direction almost instantly also introduces challenges. Unstable GPS readings make it hard to tell if a pedestrian turned or if the measure was sensor just noise. Interpolation over time can make the position clearer but that will also mean that the system has to work with old information making it slow to react. Another aspect of the slower movement speed of pedestrians is that the landmarks they are looking at are a lot smaller than a car driver's. This is mainly an issue of map quality. Getting a map showing buildings

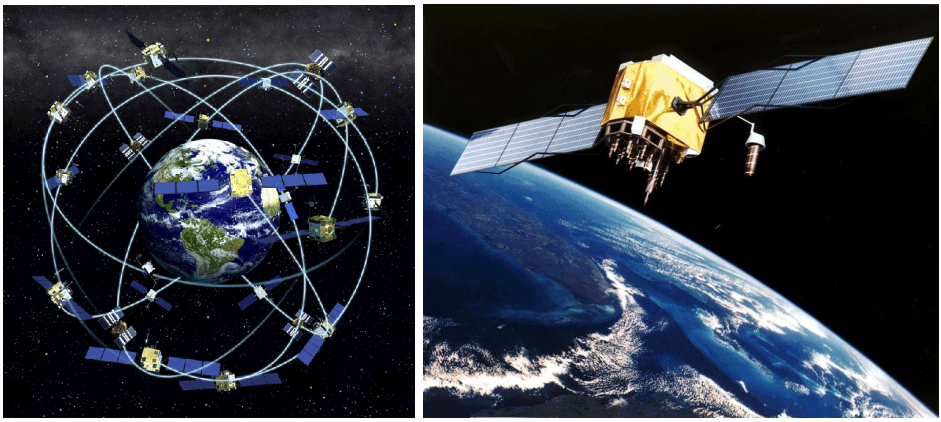

Figure 4.1: (left) 24 satellite GPS constellation, as can be seen the satellites are spread out in multiple orbital planes (right) an artistic rendering of an GPS satellite

and streets is quite easy. Maps showing trees and benches, on the other hand, are not generally available.

When looking at the history of pedestrian navigation systems, early systems focused on navigational aides for the blind (for example [27]). After 2000, work focusing on a broader audience becomes more common. In 2003, May, Ross, Bayer and Tarkiainen [28] write about navigation strategies for pedestrians as opposed to car drivers. Their main conclusion is that landmarks are more efficient in guiding pedestrians than turn instructions. The NAVIO PROJECT [15] works on building a system that works both indoors and outdoors by fusing sensor data from different sensor types to handle GPS blackouts. In Japan the NAVITIME [3] system has been successfully deployed to guide pedestrians through the railway system and at street level through the use of a map interface running on mobile phones.

Looking more specifically at tourist navigation, IKAROS PROJECT [41] criticize the branding of navigation systems designed for car navigation as "suitable for pedestrians".

> The concept of car navigation is not appropriate for pedestrian navigation. Pedestrians can pass lanes, cut across open spaces and use one-way-streets in both directions. A pedestrian navigator should take that into consideration. [41]

A slightly different approach to tourist navigation is "I DID IT MY WAY" [38] that instead of displaying a map uses a hand held device that gives haptic feedback when held in the bearing of the target location. This approach is intended to keep the eyes of the pedestrian free and support more free form exploration. Another similar project using a tactile display to guidance is POCKETNAVIGATOR [34]. POCKETNAVIGATOR combines a traditional map display with tactile feedback allowing the user to choose which mode of operations they want to use at the time.

The Spacebook project [24] shares the goal of "I Did It My Way" and PocketNavigator to allow more free form exploration of the city than holding a map displaying device in hand allows. The approach taken here is to use speech for both input and output directions. This enables the pedestrian to be eyes and hands free, while still getting both guidance instructions and interesting information about the locations as they pass them. This builds on work in natural language generation of route descriptions, for example the CORAL system [13]. It also builds on earlier audio only interfaces like EARS [5]. The Spacebook project has explored other issues as well, such as spatial grounding to compensate for unreliable GPS readings [7].

This thesis contributions to this area are the following. The Janus experimental platform in Paper III and IV (see section 5.1 for more details) which abstracts the hardware and network communication parts of an pedestrian guidance system. This abstraction greatly simplifies the creation of quick experiments testing usability and user interaction. Coupled with this contribution are the test protocols in Paper IV. Finally this thesis contributes work on the use of prediction to mask latencies and to prioritize system responses in Paper V (see section 6.1 for more detail).

Chapter 5

Software Contributions

Several different software systems, both large and small, have been constructed during this thesis. The most significant system is the Janus experimental platform for two way audio pedestrian guidances, a joint effort of our group in Umeå. Apart from that, I have developed a tool for rendering query results from PostGIS graphically, a parser/validator/unification system for the Spacebook Meaning Representation Language (MRL) and software to combat clisp stability problems during our MusicBrainz work.

5.1 The Janus experimental platform

To be able to run experiments outside of Edinburgh we have developed an experimental platform that can run anywhere OpenStreetMap data is available. This requires us to drop Ordinance Survey data, thus loosing height information. This has worked surprisingly well since the locations we have wanted to run the system in have been in central places in medium to large cities, coinciding with the places where OpenStreetMap has good coverage.

The experimental system will be released as open source under the name Janus[1] will be used in my future Ph.D. studies and beyond. It consists of an Android App that relays sound and sensor data as well as pictures from the phone's camera, the later is used during wizard of Oz (WoZ) studies[2]. On the backend we are using a PostgreSQL database with PostGIS, a SIP server called Freeswitch and a server application running on the JVM. With the exception of the SIP server, the system is easy to deploy on Debian 7 amd64 and work is being done to deprecate the SIP server to make deployment trivial. The JVM server application is based on the design and component model used in Spacebook but several modules have been reimplemented in Java or Scala to facilitate portability.

For WoZ studies and system testing, Janus contains a GUI application that can observe pedestrian movement on a map, as well as hearing the pedestrian and seeing pictures from the phone's camera (See figure 5.1). This last feature requires mounting the phone so that the camera is not blocked. During studies in Umeå a harness has been used to mount the phone on the pedestrian's chest. This GUI can be used either for WoZ studies with a wizard guiding the user or an *automatic controller* can be attached to the system and the tool can be used to observe pedestrian interactions.

The GUI application can also simulate a pedestrian for testing. In this mode no phone is connected to the system and instead the GUI application is injecting

[1] See `www.janus-system.eu` for a video demonstration of Janus.

[2] In a wizard of Oz study the experimenter tells the user they will interact with their system but in reality the system is controlled by a human.

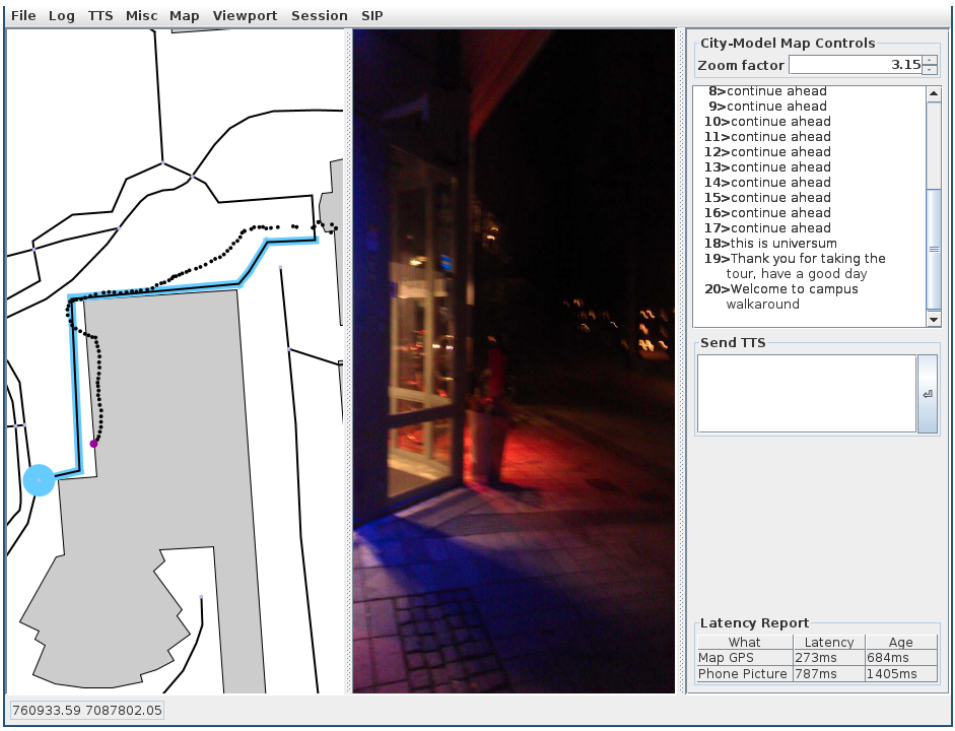

Figure 5.1: An image of the JANUS WoZ GUI. To the left is the map area, in the middle a picture from the pedestrian's phone and to the right is a log of what has been spoken to the pedestrian. Finally there is a text box to send instructions to the pedestrian with.

simulated sensor data into the system (see figure 5.2). All other components are unaware of this and act as normal. This is useful primarily for testing components during development. This practice has lead to most bugs being caught early and has helped us avoid crashes during trials with subjects.

5.2 PostGIS graphical query tool

Second to the JANUS platform the most interesting tool developed in this work is the POSTGIS graphical query tool called SB2PNG as it was originally employed against the SPACEBOOK database (build by another command line tool called OSM2SB from OPENSTREEMAP data). The way it works is taking a command line containing SQL queries and formating directives and outputting a PNG image with the geometries returned by the queries rendered onto the image.

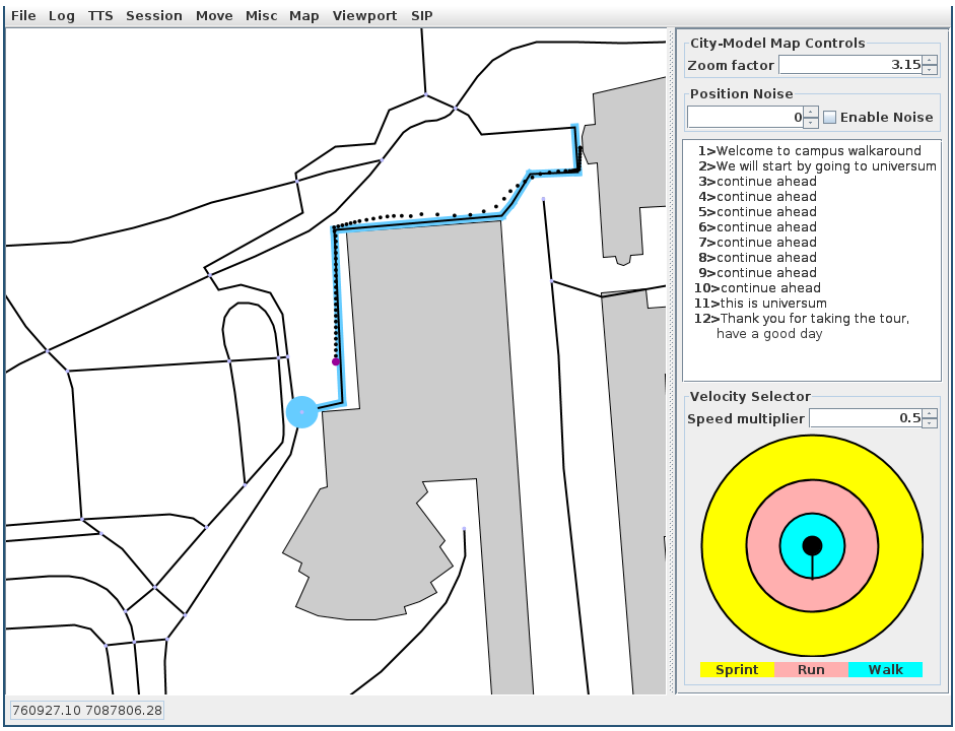

Figure 5.2: The GUI in virtual mode. Note how the picture has disappeared and a 'compass' like device has appeared in the lower right. This device can be clicked to move the virtual pedestrian.

For example:

```
./sb2png cm_3.1_stockholm_1 1600 1200 \
-c 1 0 0 -p 2 0 \
-fs 30 -t 10 -c 0 0 1 -fc 1 1 1 \
-q "select * from hasShape natural join \
isNamed natural join isA where type='restaurant'" \
-c 0.7 0.7 0.7 -t 1
-q "select geom from hasShape natural join isA where \
type='street' or type='building'" \
-z 0.8 > foo.png
```

yields the picture in figure 5.2.

Figure 5.3: Rendering of a part of Stockholm, a GPS trace shown in red, restaurants shown as blue dots. Rendered using the SB2PNG tool.

5.3 Meaning representation language (MRL) parsing and unification library

In the SPACEBOOK PROJECT we have developed an extensible meaning representation language (MRL) library over conjunctive logic extended with handles [43]. The class of MRL languages have the following structure:

- A sentence in these languages is a set of statements all joined by conjunction.
- A statement in these languages has a name an optional handle and a set of named and typed parameters.
- If a parameter is a free variable all occurrences of that variable must have types with an non-empty intersection.
- If a parameter is a handle there must exist some statement with that handle.

An example sentence would be

```
dialogAct(act:instruct,prop:H),H:pursueLeg(agent:8,leg:1008).
```

where `dialogAct` and `pursueLeg` are statements, `H` is a handle, `act`, `prop`, `agent` and `leg` are parameter names, `instruct` is a constant value and `8` and `1008` are integer values.

Our library is capable of defining a schema that defines a language from this class by listing the statements with their arguments as well as types and also create

the types. The types the system can handle are literal constants, strings, integers, floating point values as well as types that are the union of those types. Having the unions of types allows for types that can partially overlap in allowed values which is useful, for example for defining units. Consider the case of three types `TimeUnit`, `LengthUnit` and `TimeOrLengthUnit` where the last one is the union of the other two. When defining a predicate such as

```
remainingRoute(route:Id,amountLeft:TimeOrLengthUnit)
```

the union feature allows for handling the format of, in this case, the remaining route independently of whether we are representing time or distance remaining. In total this schema feature allows us to determine if a string is in our defined language.

The final piece of the puzzle is the ability to do unification between sentences. Consider the following two sentences.

```
dialogAct(act:inform,prop:H),H:distance(distance:X,unit:Y).
```

```
dialogAct(act:inform,prop:H),H:distance(distance:545.6,unit:meters).
```

here if we unify the first with the second we get.

```
{{X->545.6,
 Y->meters}}
```

This allows us to de-construct complex sentences and find the values of selected variables. More importantly it allows us to define equality of sentences independently of variable names. Also note that the unification shown is a set of sets, this is because in some cases there is more than one assignment of free variables that will unify the sentences.

5.4 Remote-hashing for Clisp

During our work with the MusicBrainz database we found some reliability and performance problems in the clisp interpreter, the lisp interpreter used by C-Phrase [32]. The problem was that when using more than a couple of MB of stack memory, clisp would crash with a segmentation fault. As a workaround we developed a C++ program implementing hash maps that could be remotely accessed by sending data over (stdin) and getting answers from (stdout). This worked very well and 'solved' the problem. In a similar situation when computations were taking a long time we developed a C program that started another clisp instance on another host accessible with ssh and sent packages of work there to utilize more compute power.

Chapter 6

Conclusions and Future Work

A conclusion from experiments in Paper I and my master's thesis [16], contrasted against the performance of the system in Paper IV is that rule-based hand crafted NLIs are easier to get usable for such a dynamic domain as navigation systems. Even if a rule-based system understands fewer alternative ways of expression we expect that it will also detect hard to parse sentences more reliably. We claim that it is better that the user must learn how to phrase themselves to be understood by the system than using a statistical system that might, by some quirk in the parser or bias in the training set, select the wrong meaning for an utterance. On top of that, statistical systems have another problem. When trying to extend the domain of a statistical system one has to give it more training data, the added training data increases the risk that the statistical system will learn something incorrect and in the end this becomes an issue in terms of both extensibility and reliability. The more you extend the system, the less reliable it becomes. A hand crafted system can combat this with either some restricted language grammar that can be formally verified or by resorting to different hand crafted parsers for different contexts.

As can be seen from the later papers in this thesis, my primary focus has become pedestrian guidance using text-to-speech (TTS). Taking Paper V as a direction, three foci of future work are *Prediction Models*, *Utterance Timing* and *Evaluation*.

6.1 Prediction models

Considering Paper V, the need for prediction models becomes apparent. While the basic idea of predicting where a user will be at some time in the near future is quite simple, initial experiments have shown that the problem can be approached from many angles and which one to choose seems non-trivial. This looks like a promising area of future research and required to construct a system like the one envisioned in Paper V. Some preliminary work has already been accomplished. See figure 6.1 for a view of our 'marble-model' predictor as currently implemented. The marble-model predictor is based on an idea on page 15-16 in [4] where the areas the pedestrian is most likely to be, like sidewalks, are 'lowest'. The predicted or measured position is allowed to roll a bit downhill to correct for measurement errors or in the case of prediction for roads not being perfectly straight and pedestrian measurements not being in a perfectly straight line. There are challenges both in terms of getting a predictor efficient enough to be used in sub-second decisions, especially when attempting to scale to more than one user. The reliability of the predictor is also a challenge. Early experiments with gravity imitating models[1]

[1] In our gravity model, each time step every point on the road network attracted the predicted position by a force decreasing quadratically with distance, apart from that the pedestrian was predicted to keep moving in their current speed and direction.

that initially looked promising and were tweakable to a point ended up failing too often to be usable. The 'marble-predictor' has performed better but that has not been fully evaluated yet.

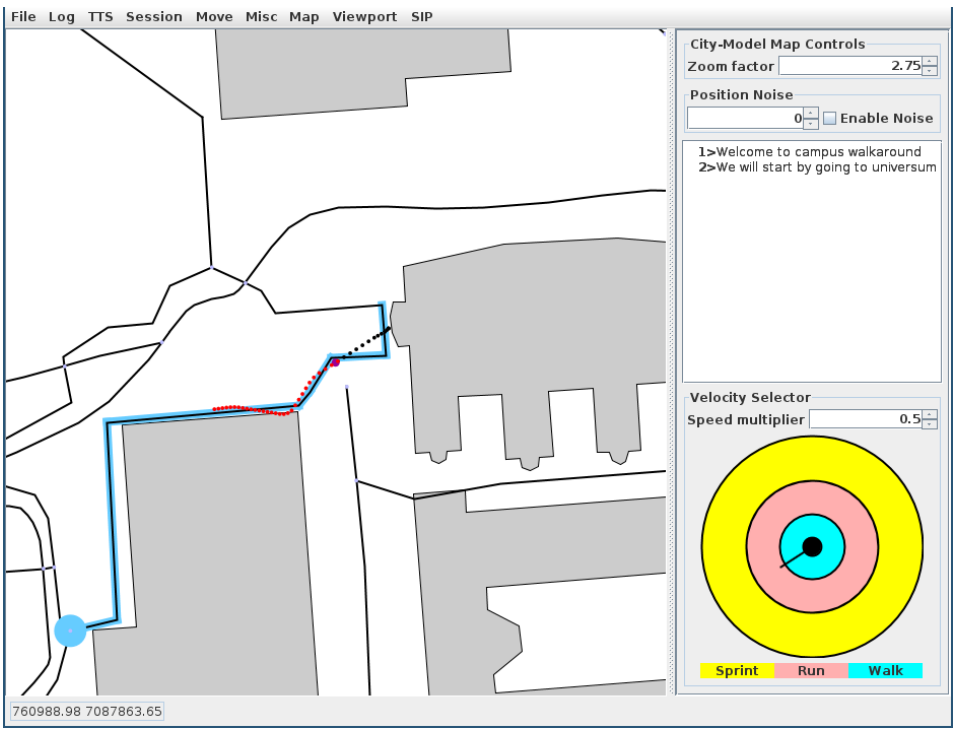

Figure 6.1: Red dots in the picture extend out from the black dots representing positions. The red dots are updated in real time allowing a detailed analysis of predictor behaviour.

An interesting fact about prediction is that it is by nature *bi-temporal* [40], that is at each point in time you can predict what the state will be at some future points in time. If this data is stored it allows asking questions like is the route now predicted similar to the one predicted 5 seconds ago? This can be useful for example in the following case. If the answer is that the current predicted route is not similar to the route predicted 5 seconds ago, that means circumstances somehow changed. Depending on what is going on in the application now might be the time to tell the user that some earlier instruction can be disregarded or that they seem to be going the wrong way.

6.2 Utterance Timing

Another important consideration is utterance timing. This includes both when to send utterances from the server so that the phone receives it at an appropriate time and how much in advance instructions should be given to pedestrians. One consideration in this area is what to run on the phone and on the server. Running code on the phone hugely reduces latencies while limiting storage and processing resources. So the decision of what to run on the phone and what to run on the server is an important issue when building *reliable* and *extensible* guidance systems. Moving computations to the slower compute unit on the phone might make the system more scalability by reducing server load. Even when the phone by itself is slower than the server, if the phone can handle some of the computations the server can handle more connected clients at once.

6.3 Evaluation

Human usability evaluations of Janus are only now starting. We have begun testing various configurations of our automatic controllers to guide subjects on a fixed tour around Umeå Universities campus. Figure 6.2 shows the flier we are using to attract subjects. We have run 5 subjects thus far through WoZ trials. Without exception the subjects were successfully guided. We are now starting evaluations to see if an automated system will be as successful.

Evaluation has been delayed in favor of perfecting a stable implementation. We will evaluate reliability by measuring task success rates. Scalability will be tested by connecting many devices at once. Extensibility is not a quality that is amenable to evaluation during user studies. Rather it can be evaluated using qualitative studies, studying how much has to be changed to add new features, etc.

Figure 6.2: The flier we are using to recruit subjects for evaluation

Chapter 7

Summary of Papers

All of the papers in this thesis are concerned with building extensible, reliable and scalable natural language interfaces over databases. Paper I is the most general and the later papers are progressively more focused. Already in Paper II the work specialises in spatial databases only to be refined further by focusing on guidance in Paper III and IV. Finally in Paper V we identify the problem of control latency and propose the use of prediction and planing to maintain reasonable response times.

Paper I:

A Natural Language Interface over the MusicBrainz Database

This paper describes how to via authoring build a natural language query application over the MusicBrainz dataset. It starts by identifying that when the user's conceptual model mismatches with the way it is stored in the database the task of constructing an NLI gets much more complicated. The solution the paper takes to that problem is to define a set of PostgreSQL views that transform the database representation into one that more closely matches what the user expects. This proved to simplify the creation of an NLI over the database. Paper I also describes various technical problems encountered when working with databases of this size (1.5GB), such as computational speed of views and how different equivalent SQL queries have different performance characteristics. The paper concludes that the approach of authoring an NLI over this domain is feasible after technical difficulties where resolved.

Paper II:

Context-dependent 'near' and 'far' in Spatial Databases via Supervaluation

This paper is the first in this thesis that handles spatial issues. The focus is on producing definitions of *near* and *far* that can be used in a database. This is a problem dependent on context and the paper approaches this by supplying data points into the database via a teacher tool. The points are then used to define bounds for NEAR and NOT-NEAR keyed on their context (similarly for FAR and NOT-FAR). If an unseen object is closer than some near object in the same context then it is near and similarly for far. This is done by compiling the contexts into SQL views.

Paper III:

Toward an Active Database Platform for Guiding Urban Pedestrians

This technical report describes the system we are using in the last three papers of this thesis. The system in question is made for guiding pedestrians either using an automatic controller or by having an operator typing instructions to the user. Our system runs on Android for the client application and JVM for the server. We also present initial observations about the system's responsiveness in regard to latency in reports from the phone to the server, which were found adequate. This report describes our understanding of the system as of October 2012.

Paper IV:

A Test-Bed for Text-to-Speech-Based Pedestrian Navigation Systems

This paper is a shorter conference paper version of Paper III. To the system in Paper III we have added two-way audio and ability to send pictures from the phone's camera to the operator. We also discuss experimental protocols for evaluating guidance efficiency.

Paper V:

Prediction and Scheduling in Navigation Systems

Paper V's main contribution is identifying an interesting future direction of improvements to pedestrian guidance systems. The paper observes shortcomings in a reactive approach to guidance, where the decision to voice an utterance happens when the system detects the user is at a decision point. We propose an alternative scheme where the system predicts future pedestrian positions. Using this prediction the system creates a schedule of things to say, accounts for priorities, time and positional constraints.

Bibliography

[1] Hiyan Alshawi, editor. *The Core Language Engine*. MIT Press, Cambridge, Mass., and London, England, 1992.

[2] Ioannis Androutsopoulos, Graeme D Ritchie, and Peter Thanisch. Natural language interfaces to databases - an introduction. *Journal of Natural Language Engineering*, 1:29–81, 1995.

[3] Masatoshi Arikawa, Shin'ichi Konomi, and Keisuke Ohnishi. Navitime: Supporting pedestrian navigation in the real world. *Pervasive Computing, IEEE*, 6(3):21–29, 2007.

[4] Phil Bartie and William Mackaness. D3.4: A pedestrian position tracker. Public deliverable, SpaceBook Project, http://spacebook-project.eu/pubs/D3.4.pdf, January 2012.

[5] Phil J. Bartie and William A. Mackaness. Development of a speech-based augmented reality system to support exploration of cityscape. *Transactions in GIS*, 10(1):63–86, 2006.

[6] Walter Blumer. The oldest known plan of an inhabited site dating from the bronze age, about the middle of the 2nd millennium bc: Rock-drawings in the val camonica. *Imago Mundi*, 18(1):9–11, 1964.

[7] Johan Boye, Morgan Fredriksson, Jana Götze, Joakim Gustafson, and Jürgen Königsmann. Walk this way: Spatial grounding for city exploration. *Proc. 4th IWSDS*, 2012.

[8] Johan Boye and Mats Wirén. Robust parsing and spoken negotiative dialogue with databases. *Natural Language Engineering*, 14(3):289–312, 2008.

[9] Philipp Cimiano, Peter Haase, Jörg Heizmann, Matthias Mantel, and Rudi Studer. Towards portable natural language interfaces to knowledge bases–the case of the orakel system. *Data & Knowledge Engineering*, 65(2):325–354, 2008.

[10] Edgar Frank Codd. Seven steps to rendezvous with the casual user. In *IFIP Working Conference Data Base Management*, pages 179–200, 1974.

[11] Ann Copestake and Karen Sparck Jones. Natural language interfaces to databases. *The Natural Language Review*, 5(4):225–249, 1990.

[12] J. Terry Coppock and David W. Rhind. The history of gis. *Geographical information systems: Principles and applications*, 1(1):21–43, 1991.

[13] Robert Dale, Sabine Geldof, and Jean-Philippe Prost. Coral: Using natural language generation for navigational assistance. In *Proceedings of the 26th Australasian computer science conference-Volume 16*, pages 35–44, 2003.

[14] Jurafsky Daniel and H. Martin James. Speech and language processing: An introduction to natural language processing, speech recognition, and computational linguistics, 2008.

[15] Georg Gartner, Andrew Frank, and Günther Retscher. Pedestrian navigation system in mixed indoor/outdoor environment–the navio project. *CORP*, pages 24–27, 2004.

[16] Johan Granberg. Learning natural language interfaces over expressive meaning representation languages. Master's thesis, Umeå Universitet, 2010.

[17] Barbara J. Grosz, Douglas E. Appelt, Paul A. Martin, and Fernando C. N. Pereira. Team: An experiment in the design of transportable natural-language interfaces. *AI*, 32(2):173–243, 1987.

[18] Ralf Hartmut Güting. An introduction to spatial database systems. *The VLDB Journal—The International Journal on Very Large Data Bases*, 3(4):357–399, 1994.

[19] Antonin Guttman. *R-trees: A dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[20] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE*, 7(4):12–18, 2008.

[21] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[22] Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer. *Generalized search trees for database systems*. September, 1995.

[23] Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. Developing a natural language interface to complex data. *ACM Trans. Database Syst.*, 3(2):105–147, 1978.

[24] Srinivasan Janarthanam, Oliver Lemon, Xingkun Liu, Phil Bartie, William Mackaness, Tiphaine Dalmas, and Jana Goetze. Integrating location, visibility, and question-answering in a spoken dialogue system for pedestrian city exploration. In *Proceedings of the 13th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 134–136, 2012.

[25] Johannes Keuning. The history of geographical map projections until 1600. *Imago Mundi*, 12(1):1–24, 1955.

[26] WolframAlpha LLC. Wolfram alpha. `http://www.wolframalpha.com`, 2012.

[27] Hideo Makino, Ikuo Ishii, and Makoto Nakashizuka. Development of navigation system for the blind using gps and mobile phone combination. In *Engineering in Medicine and Biology Society, 1996. Bridging Disciplines for Biomedicine. Proceedings of the 18th Annual International Conference of the IEEE*, volume 2, pages 506–507, 1996.

[28] Andrew J. May, Tracy Ross, Steven H. Bayer, and Mikko J. Tarkiainen. Pedestrian navigation aids: information requirements and design implications. *Personal and Ubiquitous Computing*, 7(6):331–338, 2003.

[29] Michael Minock. A phrasal approach to natural language access over relational databases. In *Proc. of Applications of Natural Language to Data Bases (NLDB)*, pages 333–336, Alicante, Spain, 2005.

[30] Michael Minock. Where are the 'killer applications' of restricted domain question answering? In *Proc. of Knowledge and Reasoning for Answering Questions (KRAQ)*, pages 98–101, Edinburgh, Scotland, 2005.

[31] Michael Minock. Modular generation of relational query paraphrases. *Research on Language and Computation*, 4(1):9–37, 2006.

[32] Michael Minock. C-phrase: A system for building robust natural language interfaces to databases. *Data & Knowledge Engineering*, 69(3):290–302, 2010.

[33] Michael Minock, Peter Olofsson, and Alexander Näslund. Towards building robust natural language interfaces to databases. In *proceedings of the 13th International Conference on Applications of Natural Language to Information Systems (NLDB)*, London, UK, 2008. Springer Verlag (LNCS 5039).

[34] Martin Pielot, Benjamin Poppinga, and Susanne Boll. Pocketnavigator: vibro-tactile waypoint navigation for everyday mobile devices. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, pages 423–426, 2010.

[35] Paul Ramsey et al. Postgis manual. *Postgis. pdf file downloaded from Refractions Research home page*, 2005.

[36] Ehud Reiter and Robert Dale. *Building natural language generation systems*, volume 152. MIT Press, 2000.

[37] Jon Rhein, Jon Rhein Greg Kemnitz, P. O. S. T. G. R. E. S. Group, Eecs Dept, Jolly Chen, Ron Choi, Jeffrey Goh, Joey Hellerstein, Wei Hong, Anant Jhingran, Greg Kemnitz, Jeff Meredith, Michael Olson, Lay-peng Ong, Sunita Sarawagi, Cimarron Taylor, Ra Ghosh, and Jim Frew. The postgres user manual.

[38] Simon Robinson, Matt Jones, Parisa Eslambolchilar, Roderick Murray-Smith, and Mads Lindborg. I did it my way: moving away from the tyranny of turn-by-turn pedestrian navigation. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, pages 341–344, 2010.

[39] Dan A Rosen, Frank J Mammano, and Rinaldo Favout. An electronic route-guidance system for highway vehicles. *Vehicular Technology, IEEE Transactions on*, 19(1):143–152, 1970.

[40] Richard T. Snodgrass, Jim Gray, and Jim Melton. *Developing time-oriented database applications in SQL*, volume 42. Morgan Kaufmann Publishers San Francisco, 2000.

[41] Annegret Stark, Marcel Riebeck, and Jürgen Kawalek. How to design an advanced pedestrian navigation system: Field trial results. In *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2007. IDAACS 2007. 4th IEEE Workshop on*, pages 690–694, 2007.

[42] Waldo R. Tobler. Automation and cartography. *Geographical Review*, 49(4):526–534, 1959.

[43] Andreas Vlachos, Stephen Clark, Tiphaine Dalmas, Robin Hill, Srini Janarthanam, Oliver Lemon, Michael Minock, and Bonnie Webber. D4.1.2 final request analysis component. Public deliverable, SpaceBook Project, http://spacebook-project.eu/pubs/D4.1.2.pdf, August 2013.

[44] Yuk Wah Wong and Raymond J. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. *ACL-07*, pages 960–967, 2007.

[45] William A. Woods, Ronald M. Kaplan, and Bonnie Lynn Nash-Webber. The lunar sciences natural language information system. Technical Report TR 2378, June 1972.

I

A Natural Language Interface over the MusicBrainz Database

Johan Granberg and Michael Minock

Department of Computing Science: Umeå University

Abstract. This paper demonstrates a way to build a natural language interface (NLI) over semantically rich data. Specifically we show this over the MusicBrainz domain, inspired by the second shared task of the QALD-1 workshop. Our approach uses the tool C-Phrase [4] to build an NLI over a set of views defined over the original MusicBrainz relational database. C-Phrase uses a limited variant of X-Bar theory [3] for syntax and tuple calculus for semantics. The C-Phrase authoring tool works over any domain and only the end configuration has to be redone for each new database covered – a task that does not require deep knowledge about linguistics and system internals. Working over the MusicBrainz domain was a challenge due to the size of the database – quite a lot of effort went into optimizing computation times and memory usage to manageable levels. This paper reports on this work and anticipates a live demonstration[1] for querying by the public.

Keywords: Natural Language Interfaces, Relational Databases, MusicBrainz, C-Phrase

1 Introduction

It has often been noted that natural language interfaces to databases suffer when the manner in which data is stored does not correspond to the user's conceptual view of such data [1, 2, 5]. This mismatch between the way data is structured and the user's conceptual model can be for a variety of reasons, but here we speculate that the three most common reasons are:

1. The database is highly normalized (e.g. to BCNF) for the sake of eliminating update anomalies.
2. The database is highly abstracted, including many attributes per relation so as to avoid cost associated with joins.
3. The database is stored in a semi-structured form corresponding to RDF triples.

It is the thesis of this paper that the user would prefer to query the data in a conceptual form similar to what is represented in the entity-relationship diagram

[1] `http://www.cs.umu.se/~johang/research/brainz/public/`

of the database domain (see figure 1). Naturally if we are given a database in one of the three forms above, we assume that it is possible, via standard view definitions, to transform the data so that it may be accessed (and perhaps even updated) via the conceptual model.

This paper explores these ideas and shows some preliminary results over the MusicBrainz database paired with the 50 natural language queries in the shared task for MusicBrainz in the QALD-1 workshop. In section 2 we present our approach to building a natural language interface supporting queries over MusicBrainz. Section 3 discusses our initial results and some anecdotes from our development efforts. Section 4 summarizes our findings and points toward near term and longer term plans, including the fielding of a live interface to MusicBrainz for querying by the public.

2 Approach

2.1 The conceptual model

After looking at the set of 50 natural language queries to be supported over MusicBrainz, we defined the conceptual model appearing in figure 1. This diagram is traditional except that it shows a form of conceptual aggregation. For example a group or a person can release an album, soundtrack or single. A traditional ER diagram would require six relationship diamonds instead of one to depict this.

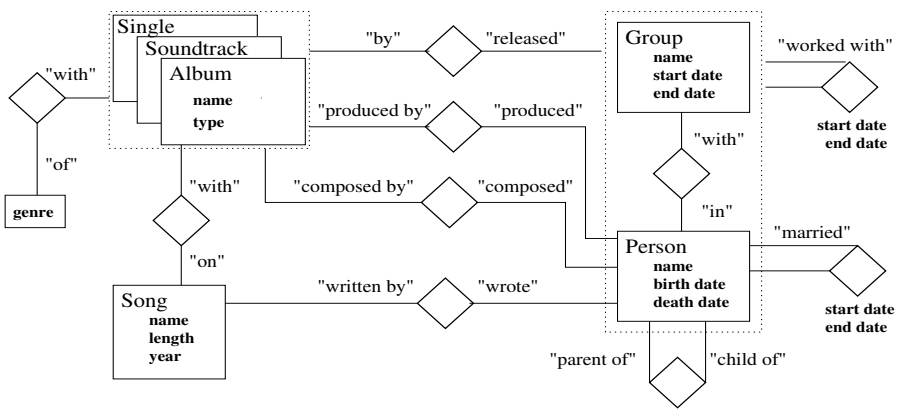

Fig. 1. Conceptual model of MusicBrainz.

2.2 The data source

We had two choices for what we used as the data source for the actual MusicBrainz data. The first was the original data stored in a PostgreSQL

database and the second was the RDF dump of the data built for the QALD-1 shared task. Because of our rooting in relational databases, we chose to simply draw on the data in the original MusicBrainz PostgreSQL database.

The schema of the underlying MusicBrainz database is primarily designed according to option 2 from section 1 above. That is, several highly abstracted relations with many attributes represent abstract entities such as `L_ARTIST_ARTIST` and `LT_ARTIST_ARTIST` which has tuples that represent both individual artists (e.g. Bob Dylan) as well as bands (e.g. REM). The database itself is rather large. There are approximately 10.6 million songs, 0.8 million albums and 0.6 million individual and bands. In our experiment we remove the tuples containing non-ASCII characters and arrive at 9.7 million songs, 0.7 million albums and 0.4 million artists.

2.3 The view definitions

Views were defined in the standard way over the base relations of the MusicBrainz database. One consideration was whether to materialize these views for quicker access to the data. This essentially doubles the database size. Our findings are that this leads to a speed up factor of approximately 3. For example using regular views the test query, "which singles did the Dead Kennedys release?" took 2.0 ms on average. Using materialized views it took an average of 0.7 ms. Considering other performance issues in the system (e.g. natural language parsing times), we concluded that querying through the views is not currently a bottleneck. Thus we did not elect to materialize views.

2.4 Authoring with the C-Phrase administration interface

Once several errors were dealt with (see section 3.1) the authoring process proceeded well. It followed the name, tailor and define method laid out in [4]. Well over half the training set can be authored for within 90 minutes. With our new enhancements to the parser to better handle WH-movement, this may be further reduced. We intend to produce series of YouTube videos that demonstrate this process.

3 Preliminary Results

3.1 Difficulties

There were several difficulties we encountered that, while perhaps anecdotal, are still worth mentioning. To date the C-Phrase system has been applied only over small databases. MusicBrainz is a sizable database, so this brought up some scalability issues that we had not earlier experienced. We assume other NLIs attempting to scale to this size of a database might face similar problems.

Large main-memory hash tables The first issue related to memory management issues in CLISP, the version of LISP that C-PHRASE is implemented over. There are some unfortunate memory bugs that corrupt CLISP memory when utilization climbs over a certain threshold. It is difficult to track, because the corruption generally causes a `Segmentation Fault` at later steps when memory is accessed. Under normal circumstances this problem does not surface. However to allow for named entity recognition, C-PHRASE materializes string values from the database into hash tables to scan for matching values in the user's typed request. In the case of MUSICBRAINZ this means building main memory hash tables containing millions of constants. This was too much for CLISP to handle.

After several false starts, the solution to this problem was to implement a remote hash facility that maintained the main memory hash tables remotely outside of CLISP. The overhead access time for these hash tables is negligible and the implementation is stable and scalable, bounded ultimately by the size of virtual memory.

Limitations of the POSTGRESQL **query optimizer** C-PHRASE maps English to logical expressions in Codd's tuple calculus. From such logical expressions, SQL is in turn generated. Before our experiment we would generate SQL such as the following to answer the query, "Which singles did the Dead Kennedys release?"

```
SELECT DISTINCT NAME
FROM SINGLE AS x
WHERE
 EXISTS(
   SELECT *
   FROM BAND as y1
   WHERE
     x.artist = y1.id  AND
     y1.name = 'Dead Kennedys').
```

Unfortunately such queries are not taken up by POSTGRESQL's optimizer. This query in fact takes 23 minutes to answer on an older Solaris server where we run our database. In contrast the equivalent query

```
SELECT DISTINCT x.NAME
FROM SINGLE AS x,BAND as y1
WHERE x.artist = y1.id  AND y1.name = 'Dead Kennedys'"
```

takes 0.7 seconds on the same server. Here the query time is entirely dominated by the time to establish the connection, the actual query execution is reported as 2 ms. We assume that the speed-up is due to the fact that the optimizer has access to both relations on the second line of the query and can plan accordingly. Instead of pestering POSTGRESQL about 'improving their optimizer', we altered our translator to produce SQL of the later variety.

3.2 Performance

We are still in the process of collecting performance data. The running example query of "Which singles did the Dead Kennedys release?" shows that the performance is adequate in the case of single join queries. We have run additional tests on queries that exercise more joins and are convinced that the current approach is feasible. The time it takes to parse natural language queries is typically in the range of one to three seconds.

3.3 Current coverage

Unfortunately recent extensions to C-Phrase have introduced system instability and have blocked a systemic precision, recall and f-measure study on the 50 unseen queries released as part of the QALD-1 shared task. The problems are mostly due to unanticipated parser bugs and performance problems when extending our parser to handle gap threading. Work continues to resolve these problems. In the meantime we have elected not to read the 50 new unseen queries in anticipation of doing a clean coverage test in the future.

As for the original 50 queries for the MusicBrainz example, in our prior working version of C-Phrase, we covered all but 5 of these queries. The 5 problematic queries (in order of estimated level of difficulty) are those involving implied time intervals ("How many bands broke up in 2010"), types (e.g. "Is Liz Story a person or a group?"), complex time calculations ("Which artists have their 50th birthday on May 30, 2011?"), computed comparison values with ellipsis (e.g. "Which artists died on the same day as Michael Jackson"), and queries involving non-quoted, non-domain string values (e.g. "Are the members of the Ramones that are not called *Ramone*"). Time permitting, we will extend C-Phrase to handle these types of queries. It will be interesting to learn about how other groups at QALD-1 approached these queries.

4 Conclusions

We were very pleased when we heard of the QALD-1 shared task. We decided to focus our efforts on the more closed-domain task of queries over MusicBrainz. We based our data on the original MusicBrainz relational database and, after confronting several technical difficulties in scaling C-Phrase, we managed to build a natural language interface that covered 45 of the 50 training examples in the QALD-1 shared task for MusicBrainz.

Unfortunately technical problems have delayed a systematic evaluation. We still intend to perform and report an evaluation of how well we cover the 50 unseen queries, but perhaps more tellingly we intend to field a natural language interface[2] for real-time querying of MusicBrainz by the public.

[2] `http://www.cs.umu.se/~johang/research/brainz/public/`

References

1. I. Androutsopoulos and G.D. Ritchie. Database interfaces. In R. Dale, H. Moisl, and H. Somers, editors, *Handbook of Natural Language Processing*, pages 209–240. Marcel Dekker Inc., 2000.
2. A. Copestake and K. Sparck Jones. Natural language interfaces to databases. *The Natural Language Review*, 5(4):225–249, 1990.
3. R. Jackendoff. *X-bar-Syntax: A Study of Phrase Structure, Linguistic Inquiry Monograph 2.* MIT Press, 1977.
4. M. Minock. C-phrase: A system for building robust natural language interfaces to databases. *Journal of Data and Knowledge Engineering (DKE)*, 3(69):290–302, 2010.
5. W. Ogden and P. Bernick. Using natural language interfaces. Technical Report MCCS-96-299, Computing Research Laboratory, New Mexicon State University, Las Cruces, May 1996.

II

Context-dependent 'Near' and 'Far' in Spatial Databases via Supervaluation

Michael Minock and Johan Mollevik

Department of Computing Science Umeå University, Sweden
Phone: +46 90 786 6398
Fax: +46 90 786 6126
Email: mjm@cs.umu.se, johang@cs.umu.se

Abstract

Often we are interested to know what is 'near' and what is 'far' in spatial databases. For instance we would like a hotel 'near' to the beach, but 'far' from the highway. It is not always obvious how to answer such nearness questions by reducing them to their crisp counterparts 'nearer' or 'nearest'. Thus we confront the vague and context-dependent relation of *near* (and *far*). Our approach follows a supervaluation tradition with a limited representation of context. The method is tractable, learnable and directly suitable for use in natural language interfaces to databases. The approach is based on logic programs supervaluated over a set of context-dependent threshold parameters. Given a set of rules with such unconstrained threshold parameters, a fixed parameter tractable algorithm finds a setting of parameters that are consistent with a training corpus of context-dependent descriptions of 'near' and 'far' in scenes. The results of this algorithm may then be compiled into view definitions which are accessed in real-time by natural language interfaces employing normal, non-exotic query answering mechanisms.

1. Introduction

A difficulty in natural language interfaces to databases (or knowledgebases) has been an adequate treatment of vagueness. For example when we ask for "a *near by* Indian restaurant", what exactly do we mean? While related questions involving the comparative and superlative forms (e.g. "is Ghandi's *nearer* than Taj Mahal?","which Indian restaurant is the *nearest?*") have crisp answers, which restaurants qualify as 'near' seems open to interpretation and arbitrary. In short, 'near' is vague.

Figure 1 depicts the general situation we model. A speaker asks for objects of type R (e.g. *Restaurants*) that qualify as members in the vague predicate V (e.g. *Near*). As a result a subset A of R is reported back to the speaker using the description C (e.g. "Ghandi's and Taj Mahal are both within 300 meters of your current position."). One obvious truism is that the determination of A

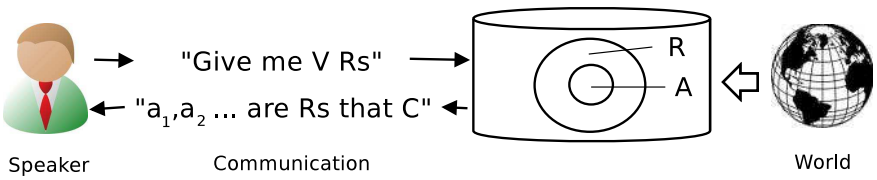

Figure 1: The basic framework

is *context dependent*. For example how much time does the speaker have for lunch? Is the answer different when it is raining? What if the speaker requests near by hospitals rather than near by restaurants? Does requesting hospitals make a difference in the determination of what distance qualifies as near? Does the number of restaurants in the vicinity of the speaker influence the distance threshold of what qualifies as *Near*? These questions hint at the strong role of context in the interaction depicted in figure 1. Of course context is a rather broad notion so let us stipulate the following three types:

> **Speaker context** involves the speaker's goals, capacities and preferences. In the case of 'near', we may ask if the speaker is walking or driving, if they are in good health, if they are under time pressure, etc.
>
> **World context** involves every thing that holds in the world external to the speaker's mind. In practice this will either be recorded in the modeled reality of the database or not be explicitly modeled. For example our database might track the location of the speaker and restaurants, but not track the weather.
>
> **Communication context** involves the actual request of the speaker. Because we restrict the form of communication so rigidly in figure 1, this context is simply what type of objects the speaker is requesting (i.e. R) and the vague predicate (i.e. V).

While representing communication context has been largely handled by restricting our attention to only types of communication depicted in figure 1, we still have the daunting task of representing speaker and world context. Surely we must feel some trepidation [23]. Are we really prepared to build rich models of the user's wants, needs and capabilities? Likewise are we prepared to attempt to formally describe how our database relates to the greater world? Perhaps we could try, but then again perhaps we should just give up and refer to separate contexts in the simplest possible way. That is by merely stipulating that contexts (e.g. c_1) exists, whatever they are, and by giving them descriptive names (e.g. c_1 is named "a 5 minute walk").

In essence we elect this simple approach to context by extending our vague predicate $V(x)$ to $V(x, c)$. Thus, assuming that I am standing at the capital building in Washington DC, we might assert $\neg Near(WhtHouse, \text{'5-min-walk'})$

but $Near(WhtHouse, \text{'afternoon-day-trip'})$. We might elect to say neither $Near(WhtHouse, \text{'15-min-walk'})$ nor $\neg Near(WhtHouse, \text{'15-min-walk'})$ if it is not clear either way. Other questions might be to ask whether there exists a context c_i where $Near(Tokyo, c_i)$? Sure, when $c_i =$ 'inter-galactic-space-travel'. In fact it seems that for any vague predicate, we can say that there is some context that makes it true and some context that makes it false.

Now that we have largely side-stepped context, this paper will focus on representing vagueness of 'near' and 'far' in spatial databases. There has been a wealth of prior work in vagueness stretching from antiquity to comprehensive modern treatments (see [20] for an overview). The present work is informed by these developments, but adopts, in the terms of [6], a *computational* rather than a *cognitive* perspective. That is we seek to support simple aspects of vagueness through leveraging modern relational database systems and theorem provers limited to tractable classes of first-order logic. Our long term goal is to robustly and efficiently support important and well circumscribed classes of vagueness without necessary recovering all the nuances of the phenomena. The work described in this paper is part of this project and addresses the special case of representing 'near' and 'far' in spatial databases. The practical motivation for focussing on 'near' and 'far' is that questions to GISs are often couched in such terms (e.g. "which Indian restaurants are near to the university?", "which hotels are near to the white house but far from a highway?") and answers or descriptions of spatial scenes could be described using such vague spatial predicates (e.g. "It's the Starbucks near to the Chinatown metro stop."). Practically all natural language interfaces to GIS eschew this problem and instead focus on qualitative relations (e.g. give the objects that overlap one another) or they work hard to answer 'near' questions using their crisp counterparts of 'nearer' and 'nearest'.

1.1. Organization of this article

This article is an extension of an earlier conference article [14] and holds a similar structure, but presents at greater depth and with a wider discussion of alternative and future work. The work is also more focussed on the case of 'near' and 'far', and does not follow up on the more extended cases of 'next-to' and 'between' that were briefly entertained in the conference paper. Section 2 provides a review of work in vagueness and in particular vague relations in spatial databases. Section 3 presents our approach basing it on a concrete example for the North Western Washington DC portion of the OpenStreetMap database. The example illustrates the essence of our approach. Section 4 discusses our prototype implementation. Section 5 presents a wide ranging discussion of the work as well as criticisms and future directions. Section 6 summarizes and concludes.

2. Background

2.1. General accounts of vagueness

Vague relations are characterized by *borderline cases* and *inquiry resistance*. Borderline cases means that there are cases that don't seem to be clearly in or out of the relation and inquiry resistance means that no amount of further information can decide the case. Inquiry resistance distinguishes vagueness from *ambiguity* for, in general, ambiguity can be resolved with further dialogue. If a user requests "list the buses that travel down D st." does this include buses that don't actually stop on D street? Once that issue is decided the question essentially becomes crisp, thus it does not resist inquiry. But in a given context if a restaurant is neither 'near', nor 'far' from me, then giving a more accurate measure of the distance does not often help. Although it can be argued that making the context of the question more specific might bring the relation closer to being crisp, we assume here that even under precise contexts vague relations are still inquiry resistant.

A comprehensive treatment of vagueness is Kees van Deemter's recent book *Not Exactly: In Praise of Vagueness* [20]. The book is an informative and entertaining look at vagueness in many of it guises and it reviews the main theoretical approaches to representing the phenomena. The book bases its definition of vagueness on the sorites paradox of Eubulides of Miletus of the fourth century B.C.E. The sorites paradox (also known as the paradox of the stone heap) asks how many stones make a heap. Since one stone does not make a heap, and since in general adding one stone to a collection should not change its status, then paradoxically we should be able to continue adding stones to the collection with it never attaining the status of a 'heap'. A more modern version of this paradox that makes explicit the notion of perceived differences, starts with assuming that we have a person that is 151 centimeters tall that we refer to as 'short'. If we stand this person next to a person that is only a hair's width taller (perceptually the two subjects appear to be the same height), then we must state that the other person is also 'short'. Of course by this line of reasoning we will conclude that people of arbitrarily large height are 'short'. Clearly somewhere the induction step must break down.

More abstractly, in van Deemter's terminology, there are three fundamental properties of vague relations (such as $tall(x)$ or $near(x, y)$) based on gradable measures (such as height and distance): *admissibility*, *tolerance* and *non-transitivity*. In the example of nearness:

> **Admissibility** states that if some object is 'near' at a given distance, then if it were moved to a shorter distance away, then it too would be 'near'.
>
> **Tolerance** states that if an object is 'near', then if it is moved an imperceptible distance away, it still remains 'near'.
>
> **Non-transitivity** states that if object x is 'near' to object y and object y is 'near' to object z, then it is not necessarily the case that object x is 'near' to object z.

We will adhere easily to admissibility and non-transitivity, though as we will see, we will struggle later with tolerance.

Van Deemter's book does a thorough job of presenting and contrasting the linguistic and semantic approaches to handling sorites problems, with special focus on the example of tall. This includes the naive method of specifying thresholds, supervaluation, Kamp's notion of incoherent contexts, introspective accounts, fuzzy logic based approaches, and finally probabilistic logic based approaches. We refer the interested reader to [20].

2.2. Supervaluation

The tradition which guides the present work, is *supervaluation* (see [7, 12] for a philosophical description and [8, 18, 15, 3] for practical approaches that, broadly speaking, can be classified as employing supervaluationistic techniques). From [12],

> According to [supervaluationist] theory, a sentence is true if and only if it is true on all ways of making it precise. This yields borderline case predications that are neither true nor false, but classical logic is preserved almost entirely.

In this paper this is captured by parameterizing the definition of vague predicates with various thresholds constants which are not explicitly set. A *precisification* of the vague predicate is a setting of the relevant threshold parameters to numerical values that are consistent with observations. A statement is *supertrue* if it is true over all possible precisifications. To illustrate, let us define 'near' as $(\forall x)(x < n_1 \Leftrightarrow near(x))$ and assume observations $near(2)$ and $\neg near(10)$. For simplicity, let us assume that our universe of distances is the whole numbers between 0 and 20. Given the observations we have 8 consistent precisifications of our rule. But no matter what the actual setting of n_1 is, the rules above are sufficient to deduce that $near(1)$. Thus $near(1)$ is supertrue, $near(11)$ is superfalse and $near(5)$ is neither supertrue or superfalse, being either true or false with different consistent settings of the parameter n_1. This formulation enforces the property of admissibility and non-transitivity, but does not directly address tolerance. Issues of tolerance aside, what we have is a fairly reasonable formulation for practical application. While one can solve for consistent settings of n_1 in a straight forward way (e.g. $n_1 = 3$ is consistent with the observations), this is not necessary if one wishes to simply use the system to deduce membership or non-membership in vague relations.

There is a limitation of the above system that should be mentioned. Although the formulas given do not determine a specific value for n_1, each model does set a specific value for the threshold. Where this becomes tricky is when we introduce another parameter that determines when a distance is definitely not 'near'. Let us say for example that we introduce the rule: $(\forall x)(x > n_2 \Leftrightarrow \neg near(x))$. Under this system, because for each model a given distance must be either *near* or $\neg near$, we unwittingly induce the constraint that $n_2 = n_1 - 1$. This is unfortunate, because once we determine parameter settings, we would

like for the resulting logic to model inquiry resistance by remaining agnostic about the determination of the vague predicates for values in some gap. For example in the most conservative setting[1] of parameters for the above example is $n_1 = 3$ and $n_2 = 9$. Instead of embracing partial logic, a work around is to simply introduce an 'opposite' vague predicate $far(x)$ so that we have the rule, $(\forall x)(x > n_2 \Leftrightarrow far(x))$ and the rule that determines mutual exclusion: $(\forall x)(near(x) \rightarrow \neg far(x))$. Thus we can now have consistent settings of threshold parameters under classical first-order logic where a distance is provably *near*, provably far or neither. This gives us our 'gaps' that are needed to model inquiry resistance.

There has been a series of practical works that model vagueness via supervaluation. The work in [8] was one of the first attempts to fix parameters in rule based systems by letting domain knowledge interact with rules to give tighter bounds on intervals. The work is similar to the work carried out here, but no attempt to identify tractable classes of problems was undertaken. Other important work in supervaluation is Bennett's work on VAL (Vague Adjective Logic) [2] and Pulman's work on vague predicates and degree modifiers [15].

2.3. Vague relations in spatial databases

Typically spatial databases represent regions by adding geometric types (`POINT`, `LINE`, `POLYGON`, etc.) to the basic attribute types (e.g. `INT`, `VARCHAR`, `DATE`, etc.). Such geometric types (e.g. as proposed in SQL/MM) have a host of well-defined operators (e.g. *overlaps*, *contains*, *disjoint*, *strictly-above*, *does-not-extend-to-the-right-of*, etc.) and functions (*distance*, *area*, etc.) that can be used as conditions or terms in queries. The canonical types of crisp queries are *point queries* (e.g. "what park am I currently in"), *range queries* (e.g. "what are the Chinese restaurants between H and F streets and 9th and 11th streets?"), *nearest neighbors* (e.g. "Where is the nearest ATM from the corner of 9th street and F street?") and *spatial joins* (e.g. "give Indian restaurants within 100 meters of a metro stop."). The use of R-tree indexes [10] and their generalization ([11]) give log-based access to spatially arranged objects, by indexing on the bounding rectangle of polygon regions and line segments. While spatial databases provide a standard set of basic spatial operators and functions we seek to support spatial predicates 'near' and 'far' which are of a vague, context-dependent nature.

While many researchers have pursued fuzzy logic approaches [5, 1, 21] in spatial databases, it should be remarked that most of these approaches have focussed on what could be called *indeterminate* spatial relations [17] of fuzzy objects, rather than focus on the vague relations *near* (and far) that obtain between crisp objects. An indeterminate relation considers an object that has unknown shape, such as a river that may widen during rainy season or a forest whose boundary may taper off into a clearing. While such objects have unknown exact shape, there does exist some shape and thus, in principle, there

[1] A conservative setting of parameters selects the widest gaps between opposite relations consistent with observations.

are answers to basic spatial predicates (e.g. RCC8 relations [16]) and functions (e.g. distance) evaluated over them. A common way to represent such indeterminate regions is through the so-called *egg yolk* method [4]. The greatest possible extent of a region is represented as well as the most compact possible region and the calculus of the exact spatial predicates and functions gives rise to three-valued logics and intervals of possible values. Some recent work that flirted more directly with vague spatial relations is [13]. While they explicitly side step "vague, ambiguous and context dependent expressions" via a controlled language approach, they provide an interesting definition of 'between' that can be thought of as being on the cusp of becoming a vague relation.

3. An Approach to Context-Dependent *Near* and *Far*

Our approach to represent $Near$ is based on supervaluation over multiple contexts. Returning to figure 1, given a user's question for Rs that are V (e.g. "*Restaurants* that are *near*") under context c, we apply the vague predicates at the intensional level. That is the set of answers A is:

$$\lambda c.\{x|R(x) \wedge V(x,c)\} \text{ where } V(x,c) \Leftrightarrow n(c) < val(x)$$

We focus on the case where R is a basic type predicate (for example $Church$, Pub or $Museum$ and V is a binary (as opposed to unary) vague predicate $near$ (or far) extended with a context argument. Specifically $near(x,y,c)$ and $far(x,y,c)$ are true when object x is 'near' to (or 'far' from) object y in context c. The val function represents the 'distance' between the geometries of two objects. For now let us assume this distance function is the straight line distance metric Δ_{sld} and that the function geo maps from object ids to their associated geometries (i.e. points, polygons or lines).

3.1. The definition of the context-dependent Near and Far

Ignoring types for now, the main rules that define the vague predicates are:

$$(\forall x)(\forall y)(\forall c)(near(x,y,c) \Leftrightarrow \Delta_{sld}(geo(x),geo(y)) < low(c))$$
$$(\forall x)(\forall y)(\forall c)(far(x,y,c) \Leftrightarrow high(c) < \Delta_{sld}(geo(x),geo(y)))$$

These rules provide the necessary and sufficient conditions[2] for $near(x,y,c)$ and $far(x,y,c)$. The necessary conditions state that if two objects are 'near' to (or 'far' from) one another in a given context, then the distance between their associated geometries must be less than some threshold low (or greater than some threshold $high$) for the context. The sufficient conditions state that if the distance between two objects associated geometries is less than some threshold low (or greater than some threshold $high$) for a given context, then two objects are 'near' to (or 'far' from) one another in that context. Nowhere

[2] An error in the conference paper [14] was that only sufficient conditions were included.

do we explicitly set these parameters $low(c)$ and $high(c)$, for they are to be constrained by observations.

Note that the functions we are using in our system are partial and have rigid signature restrictions: $geo(x)$ maps object ids to geometries, $\Delta_{sld}(g_i, g_j)$ maps two geometries to a numeric value, and $low(c)$ and $high(c)$ map contexts to numerical values. The relation $<$ is defined over a finite set of numerical values under active domain. The function signature requirements give a finite Herbrand universe for any system specifying a finite set of contexts and a finite set of observations. This along with the fact that rules are limited to the Horn property gives us a tractable algorithm[3].

Finally we model the 'opposite' relation between $near$ and far in a given context via:

$$(\forall x)(\forall y)(\forall c)(near(x, y, c) \Rightarrow \neg far(x, y, c)).$$

3.2. Calculating thresholds from context-dependent observations

By way of example let us consider observations over the contexts c_1 = '10 minute stroll', c_2 = 'five minute sprint in rain' and c_3 = 'short bicycle ride'. In the scene[4] in Figure 2 we may have observations of the following sort:

$$near(m_2, s_1, c_1) \wedge far(m_2, s_4, c_1) \wedge \neg near(m_4, s_5, c_2) \wedge far(s_6, s_2, c_2) \wedge ...$$

Based on a set of such observations, the most conservative setting of the thresholds could be $low(c_1) = 240$, $high(c_1) = 600$, $low(c_2) = 750$, $high(c_2) = 5000$ and $low(c_3) = 780$, $high(c_3) = 4500$. As will be discussed in Section 4, we have a tractable algorithm to calculate these parameters.

3.3. Representing and comparing contexts

While the $near(x, y, c)$ and $far(x, y, c)$ predicates include a context argument, we also wish to associate facts as being true (or false) in contexts. To achieve this we include a predicate $ist(p, c)$ (standing for 'is true') that records that a proposition $p \in \mathcal{P}$ holds in a particular context c. These facts are propositions from an arbitrary finite set $\mathcal{P}$. For example let us say that the propositions are on-foot, raining, day-time, on-bike. Thus we might have the following facts: $ist(\text{on-foot}, c_1)$, $ist(\text{on-foot}, c_2)$ $\neg ist(\text{on-foot}, c_3)$, $\neg ist(\text{on-bike}, c_1)$, $\neg ist(\text{on-bike}, c_2)$, $ist(\text{on-bike}, c_3)$, $ist(\text{raining}, c_2)$. If a context c_i makes no claims about $raining$, then neither $ist(raining, c_i)$ nor $\neg ist(raining, c_i)$ will be asserted.

Given the ist predicate and a set of context dependent observations, we are able to calculate several useful relationships between contexts:

[3]Technically this is a fixed parameter tractable algorithm based fixing the arities of the functions and predicates. See formal definitions and proofs developed in [14]

[4]We use OpenStreetMap data in PostgreSQL extended with PostGIS. The scene in 2 represents a section of Washington, DC and consists of over a thousand unique objects. Each of these has an associated geometry type (e.g. a polygon, line or point) and a set of descriptive attributes that associates name and types.

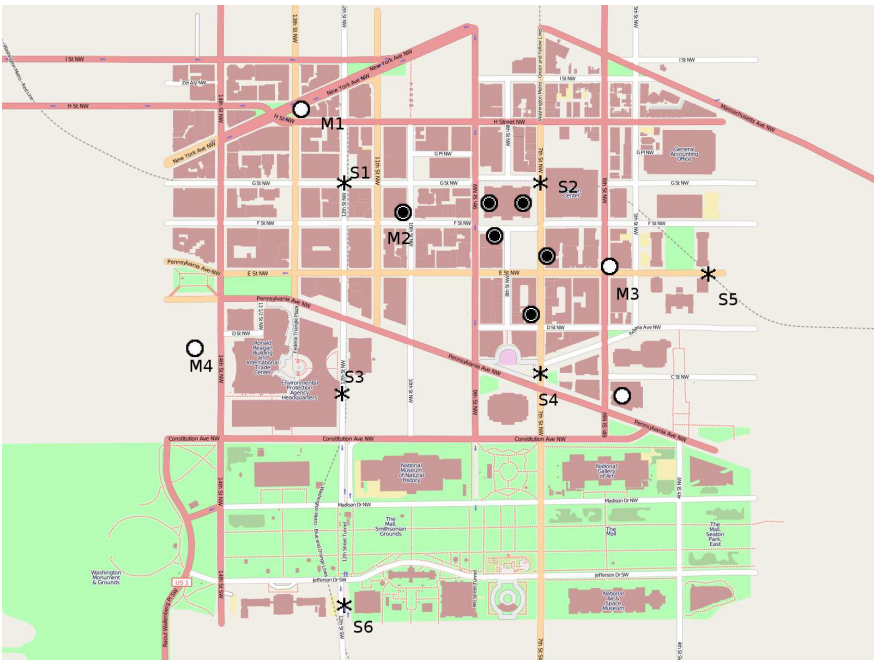

Figure 2: Museums definitely near (•) and possibly near (○) to a metro (∗) in the context of a '10 minute stroll'

Possibly Equal ($c_i \approx c_j$)

When we take into account both observations as well as *ist* facts we may calculate whether two contexts are *possibly equal*. We say that two contexts c_i and c_j are possibly equal (denoted $c_i \approx c_j$) if their equality is consistent with the observations and *ist* facts. This can be tested by merely stating their equality and testing whether the observations and *ist* facts are consistent. In our example *ist* and observations in Section 3.2, none of the contexts can possibly be equal to one another.

Generality ($c_i \sqsupset c_j$)

Ignoring observations we can speak of context c_i being at least as general as a context c_j (denoted $c_i \sqsupseteq c_j$) if and only if $(\forall p)(p \in \mathcal{P} \wedge ist(p, c_j) \Rightarrow ist(p, c_i))$ and $(\forall p)(p \in \mathcal{P} \wedge \neg ist(p, c_j) \Rightarrow \neg ist(p, c_i))$. This can be tested efficiently for each context pair by testing the consistency of these rules with the *ist* facts. A context c_i is more general than c_j (denoted $c_i \sqsupset c_j$) if $c_i \sqsupseteq c_j$, but **not** $c_j \sqsupseteq c_i$. In our example $c_1 \sqsupset c_2$.

More Discerning ($c_i \prec c_j$)

Ignoring *ist* facts, a relationship we calculate is what we call a context being *more discerning* than another context. A context c_i is at least as discerning as

c_j (denoted $c_i \preceq c_j$) if everything that is 'near' in c_i is 'near' in c_j and everything 'far' in c_i is 'far' in c_j. A context c_i more discerning as c_j (denoted $c_i \prec c_j$) if $c_i \preceq c_j$, but **not** $c_j \preceq c_i$. Formally we can compute if $c_i \preceq c_j$ if we add the following rules to above rules and observations and test for consistency:

$$(\forall x)(\forall y)((near(x, y, c_j) \Rightarrow near(x, y, c_i)) \wedge (far(x, y, c_j) \Rightarrow far(x, y, c_i)))$$

In our example above, it is the case that $c_1 \prec c_2$.

3.4. Generating views

Based on the set of contexts, observations, and then the calculated high and low parameters, we can generate views that capture what is 'near' and 'far' in a given context. Thus the view definition of `NEAR(XID, YID, CONTEXT)` provides the objects that are 'near' to one another in the given context. Its form is a straightforward result of pairs $\langle c_i, low(c_i)\rangle$:

```
CREATE VIEW NEAR(id1,id2,context) AS
  (SELECT x.osm_id,y.osm_id,'10 minute stroll' FROM
    planet_osm_point AS X, planet_osm_point AS Y
    WHERE ST_Distance(x.way,y.way) < 240)
  UNION
  (SELECT x.osm_id,y.osm_id,'10 minute stroll' FROM
    planet_osm_polygon AS X, planet_osm_point AS Y
    WHERE ST_Distance(x.way,y.way) < 240)
   ...
  UNION
  (SELECT x.osm_id,y.osm_id,'5 sprint in rain'
   ...
```

This view can be accessed via natural language through the natural language interface system. An analogous view is defined for `FAR(XID, YID, CONTEXT)`

3.5. Multiple distance metrics

In practice, it is often the case that a straight line distance measure is an unreliable metric of 'real' distance. Certainly the existence of a subway system warps the notion of what is 'near'. Modern GIS treatments are getting more sophisticated in calculating such alternative distances. For example taking into consideration the road network and approximate travel times, it is fairly straightforward to develop a metric $\Delta_{driving}$, it is even feasible that we can develop $\Delta_{walking}$ or Δ_{metro}. Given this we could then imagine composing these to generate metric functions such as $\Delta_{walking\&metro}$.

Although we have not yet performed any experiments with these alternative distance metrics, we note here that the impact of this will be to make the distance function definition based on input context. Thus the basic vague predicate definition rules from above become:

$$(\forall x)(\forall y)(\forall c)(near(x, y, c) \Leftrightarrow \Delta(geo(x), geo(y), c) < low(c))$$
$$(\forall x)(\forall y)(\forall c)(far(x, y, c) \Leftrightarrow high(c) < \Delta(geo(x), geo(y), c))$$

This will then necessitate the assignment of distance metrics to various contexts:

$$(\forall x)(\forall y)(\Delta(geo(x), geo(y), c_1) = \Delta_{sld}(geo(x), geo(y)))$$
$$...$$
$$(\forall x)(\forall y)(\Delta(geo(x), geo(y), c_{10}) = \Delta_{driving}(geo(x), geo(y)))$$

While this requires slightly more knowledge engineering, it does not alter the tractability of the approach.

3.6. Type dependence

Based on our intuition that a person being 'near' to a waste paper basket and a person being 'near' to a park should be accorded different distance thresholds, let us entertain the possibility of extending our approach with types (e.g. a restaurant, ATM, park, road, etc.). With respect to the definition of *near* (and far) this would lead to a proliferation of rules including:

$$(\forall x)(\forall y)(\forall c)(near(x, y, c) \Leftrightarrow ATM(X) \wedge Restaurant(Y) \wedge \Delta(geo(X), geo(Y), c) < low(ATM, Restaurant, c))$$

Note that the low parameters now take types as well as the named context as arguments. This is a straight forward extension to the basic case although it results in many more rules – quadratic in the number of types.

A serious limitation with this approach is that monotonicity of logic will result in the most restrictive threshold bounds to be induced between types. This will likely quickly lead to hard to explain inconsistencies that render the approach unworkable. A possible fix would be to stipulate that pairwise disjoints between types. For example the types could included restaurants, ATMs, etc. For example:

$$(\forall x)(Restaurant(x) \Rightarrow \neg Road(x) \wedge \neg ATM(x) \wedge ...)$$
$$...$$

Even if we were able to work with a set of mutually exclusive types, the whole approach suffers from requiring many more observations to fully constrain the thresholds across all type combinations. For this reason we largely dismiss the idea (originally proposed in [14]) of including types in the rules.

4. Prototype Implementation

The approach of section 3 is implemented in an initial LISP prototype integrated with the theorem prover SPASS and PostgreSQL extended with PostGIS over OpenStreetMap data. The implementation consists of two web-based tools: the *teaching tool* and the *query tool*. The teaching tool lets an administrator build and manage contexts (see Figure 4). The output of the learning tool is a set calculated relationships between contexts and the view expressions that define the context-dependent `NEAR` and `FAR` relations. The query tool lets casual users obtain answers to 'near' and 'far' queries via limited natural language dialogue (see Figure 5).

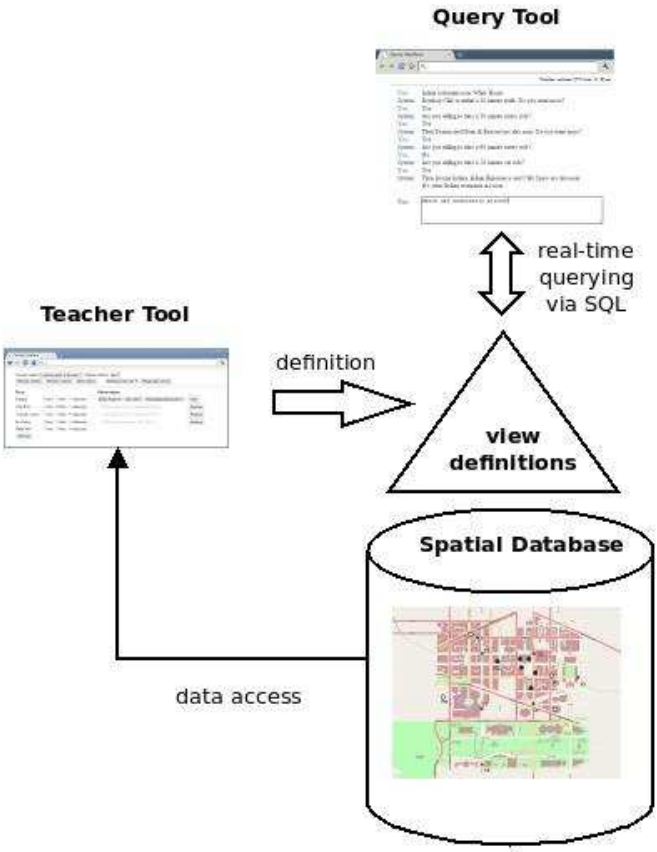

Figure 3: The overall architecture

4.1. The teacher tool

The teacher tool enables an administrator (hereafter referred to as the teacher) to add and name contexts, set associated propositions in contexts to $true$, $false$ or *unknown*, and then to make statements of what is 'near' and 'far' in contexts. Additionally the teacher must specify which distance metric is relevant in the context and must also specify a descriptive name for the context. These descriptive names (e.g. a '5 minute sprint in the rain') will be used to identify the context to casual users.

As the teacher builds up a library of contexts, the system alerts the teacher to related contexts that are possibly equal (see $c_i \approx c_j$) above to a new context that they are asserting. The teacher is encouraged to refrain from defining a context c_i if there already exists a context c_j, where $c_i \approx c_j$ and $c_j \sqsubset c_i$ and $c_j \prec c_i$ (see Section 3.3 for a formal definition of these terms). This captures

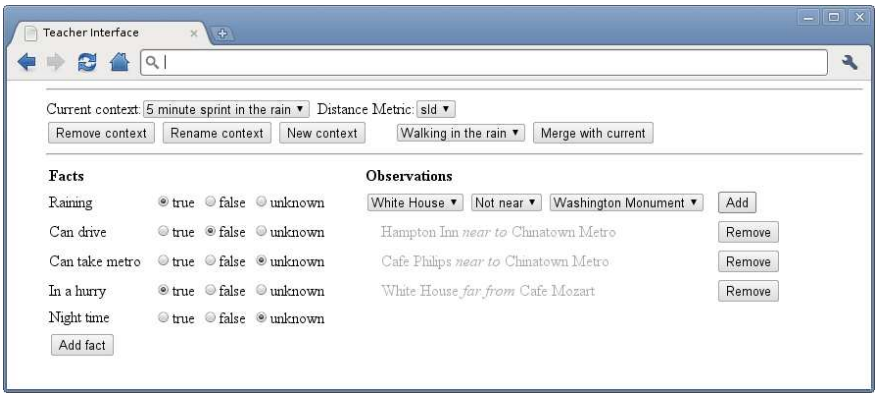

Figure 4: The interface to the teacher tool

the intuition that if a context is more specific, but in being so it widens a gap between opposite predicates, then it is probably irrelevant. Likewise a new context c_i that is more general than a prior context c_j ($c_i \sqsubset c_j$) and more discerning $c_i \prec c_j$ should probably supersede c_j and c_j should be removed from the library. These are just recommendations to the teacher however. The teacher can author contexts as they see fit.

Once the teacher has defined a library of contexts, they generate a view definition for the context library. This starts by calculating the most conservative settings of the parameters, followed by a translation of the entire body of contexts into view definitions. This is achievable in polynomial time (see [14]).

Core reasoning services

The core reasoning services in the teaching tool are carried out by the first order resolution theorem prover SPASS [22]. We encode the vague observations, the *ist* statements, the unique names assumption and $<$ over numerical constants (under active domain) in first order logic limited to Horn clauses. These encodings are confirmed to be consistent via a satisfiability check. One tricky issue worth mentioning around the definition of $<$, is that must discretize the active domain of ordered numerical values $d_1, ..., d_n$. This includes Horn-based constraints such as:

$$x = d_i \Leftarrow x < d_i + 1 \wedge d_i - 1 < x$$

Assuming consistency of the defined contexts and observations, based on the reasoning question we are seeking to answer, we encode constraints such as the value of a parameter, or whether two contexts are equal, or any of the other questions from Section 3.3, and test for consistency. The view compilation routine which finds consistent settings of all the *high* and *low* function values is the most expensive of the operations. However this does not require interaction and, again based on the Horn case it is tractable. Further experiments are underway to improve our prototype's performance.

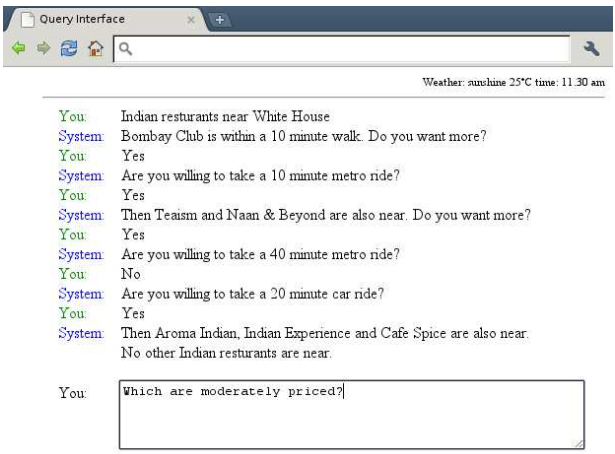

Figure 5: A dialog over the query tool.

Given that we have the view, materialized or otherwise, that defines the vague predicate `NEAR` and `FAR`, the question is how do we use such a view in querying the underlying database. A common use case is that we are unaware of the user's context and they then ask a nearness query. Now given the assumption that the true context of the user matches a context defined in the context library, this will mean that a family of possible thresholds will be possible.

To resolve a more discerning notion of *near* the system will need to obtain from the user their context. This may involve setting predicates based on world state (e.g. the weather or time of day) or via explicit questions (e.g. "Are you driving, walking, or biking?", "Can you take a metro?", etc.) and using this to narrow the set of possible contexts. This could also be based on users stating that objects are definitely near or far, or the user even suggesting the actual threshold values explicitly. The key point is that as the set of possible contexts becomes more constrained, the boundary *low* and *high* parameters will become more and more constrained. The dialogue in Figure 5 follows a strategy where the set of answers common to all possible contexts are identified first, followed by questions that systematically rule out contexts and progressively report all answers in the next less restrictive context, etc. until all possible contexts are addressed or ruled out.

5. Discussion

The original ambition of this article was to cover a wide class of vague spatial preposition including 'between','on','next to','at', etc. As it turned out, we decided to focus our attention on the more limited case of 'near' (and 'far').

Now that we have developed what we feel to be a concrete and practical approach to this simpler case we will be generalizing our approach to more complex and varied cases.

The general interpretation of vague language in this article has been as 'convention'. In general this leads to difficulties. For example consider the non-spatial request for "large cities in Alaska." If we assume that the only relevant gradable measure for a large city is it's population, where would we set the threshold? Obviously this depends on the comparison set. For example do we want a large city by Alaskan standards, or do we want large cities by more conventional standards – in which case perhaps the correct answer is that Alaska does not have any large cities. A possible way to extend vague predicates is with definitions such as "tall means above $n1$ standard deviations from the mean." The $n1$ here remains as a parameter, but one that becomes folded up within a complex calculation that takes the distributional context into account. Formally, in relation to figure 1, this would be:

$$\lambda c.\{x|x \in V^*(\{y|R(y)\}, c)\} \text{ where } x \in V^*(S, c) \text{ when } val(x) \text{ is at least } n^*(c) \text{ standard deviations above the mean } val \text{ for members of } S$$

This paper has based its approach on relational database technology and theorem provers applied over tractable cases of Horn clauses with a finite Herbrand universes. No doubt our approach to context can be criticized for its simplicity given the more sophisticated options [9, 19]. While we slightly extended our notions of context to make certain propositions true or false in context, adding a context argument to predicates is essentially our approach to context in this paper. The main virtue of this is its simplicity and its representation in tractable first-order logic.

Conventional wisdom says that limiting approaches to vagueness to classical first-order logic is too restrictive. In fact Kees van Deemter concludes his book [20] with the analogy of giving up classical logic with the expulsion of Adam and Eve from paradise. It's a painful, but necessary. While we acknowledge that this is probably ultimately true to capture advanced cognitive aspects of vagueness, in this paper we are fighting the expulsion on computational grounds.

6. Conclusions

The approach detailed in this paper treats vagueness in spatial databases as a set of conventions, based on context. The approach is based on definite logic programs with contexts represented as first-class objects and a type of supervaluation over a set of threshold parameters. Given a set of context-dependent rules with open threshold parameters, a tractable algorithm finds a setting of the parameters that are consistent with a training corpus of vague spatial statements. The results of this algorithm may then be compiled into view definitions that may be integrated into normal SQL-based databases. And

in turn such view definitions may be exploited by natural language interfaces employing, normal, non-exotic relational query answering mechanisms.

The need to map vague spatial descriptions to precise logical formulas over spatial databases is a problem that is quite relevant as we develop natural language interfaces for communicating with anything, anywhere. This article has presented a scalable approach to support vague spatial relations for querying spatial databases using natural language phrases such as 'near' and 'far'. In doing so, it has brought to bear work in supervaluation based approaches to vagueness and treatments of context. Experiments have been encouraging and efforts are underway to turn our prototype into a system.

References

[1] D. Altman. Fuzzy set theoretic approaches for handling imprecision in spatial analysis. *Journal of Geographical Information System IBM*, 8(3):271–289, 1994.

[2] B. Bennett. A theory of vague adjectives grounded in relevant observables. In *KR*, pages 36–45, 2006.

[3] B. Bennett, D. Mallenby, and A. Third. An ontology for grounding vague geographic terms. In *FOIS*, pages 280–293, 2008.

[4] A. Cohn and N. Gotts. Representing spatial vagueness: A mereological approach. In *KR*, pages 230–241, 1996.

[5] A. Dilo, R. de By, and A. Stein. A system of types and operators for handling vague spatial objects. *International Journal of Geographical Information Science*, 21(4):397–426, 2007.

[6] M. Duckham and M. Worboys. Computational structure in three-valued nearness relations. In *COSIT*, pages 76–91, 2001.

[7] K. Fine. Vagueness, truth and logic. *Synthèse*, 30:263–300, 1975.

[8] N. Goyal and Y. Shoham. Reasoning precisely with vague concepts. In *AAAI*, pages 426–431, 1993.

[9] R. V. Guha. *Contexts: A Formalization and Some Applications*. PhD thesis, Stanford University, 1991.

[10] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD'84, June 18-21, 1984*, pages 47–57. ACM Press, 1984.

[11] J. Hellerstein, J. Naughton, and A. Pfeffer. Generalized search trees for database systems. In *VLDB*, pages 562–573, 1995.

[12] R. Keefe. Vagueness: Supervaluationism. *Philosophy Compass*, 3(2):315–324, 2008.

[13] S. Mador-Haim, Y. Winter, and A. Braun. Controlled language for geographical information system queries. In *Proceedings of Inference in Computational Semantics*, 2006.

[14] M. Minock. Vague relations in spatial databases. In *Proc. of Applications of Natural Language to Data Bases(NLDB)*, pages 203–208, Cardiff, Wales, 2010.

[15] S. Pulman. Formal and computational semantics: a case study. In *Proceedings of the Seventh International Workshop on Computational Semantics: IWCS-7, Tilburg, The Netherlands, 2007*, pages 181–196, 2007.

[16] D. Randell, Z. Cui, and A. Cohn. A spatial logic based on regions and connection. In *KR*, pages 165–176, 1992.

[17] A. Roy and J. Stell. Spatial relations between indeterminate regions. *Int. J. Approx. Reasoning*, 27(3):205–234, 2001.

[18] P. Santos, B. Bennett, and G. Sakellariou. Supervaluation semantics for an inland water feature ontology. In *IJCAI*, pages 564–569, 2005.

[19] L. Serafini and P. Bouquet. Comparing formal theories of context in ai. *Artif. Intell.*, 155:41–67, May 2004.

[20] K. van Deemter. *Not Exactly: In Praise of Vagueness*. Oxford University Press, 2010.

[21] F. Wang. Handling grammatical errors, ambiguity and impreciseness in GIS natural language queries. *Transactions in GIS*, 7(1):103–121, 2003.

[22] C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobalt, and D. Topić. SPASS version 2.0. In *18th International Conference on Automated Deduction*, pages 275–279, Kopenhagen, Denmark, 2002.

[23] T. Winograd and F. Flores. *Understanding computers and cognition - a new foundation for design*. Addison-Wesley, 1987.

III

University of Umeå
Department of Computing Science
SE-901 87 Umeå, Sweden

UMINF-12.18
October 2012

Toward an Active Database Platform for Guiding Urban Pedestrians [1]

by

Michael Minock, Johan Mollevik and Mattias Åsander

[1]The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 270019 (SpaceBook project www.spacebook-project.eu) as well as a grant through the Kempe foundation (www.kempe.com).

ABSTRACT

We present an Android-based platform for incrementally presenting spoken route directions to guide pedestrians to destinations. Our approach makes heavy use of stored procedures and triggers in an underlying PostGIS spatial database. In fact most of the 'intelligence' of our prototype resides in database stored procedures and tables. As such it represents an example of a challenging real world case study for the use of persistent stored modules (PSM) in a complex mobility application. It also provides a platform to study performance tradeoffs for complex event processing over spatial data streams.

1 Introduction

The automated generation of route directions has been the subject of many recent academic studies [2, 11, 9, 8, 13, 3, 12, 7] and commercial projects (e.g. products by Garmin, TomTom, Google, Apple, etc.). While most focus has been dedicated to automobile drivers, there has also been an effort to provide route directions to pedestrians (e.g. Google and SIRI). The pedestrian case is particularly challenging because the location of the pedestrian is not just restricted to the road network and the pedestrian is able to quickly face different directions. In addition the scale of the pedestrian's world is much finer, thus requiring more detailed data representation. Finally the task is complicated by the fact that the pedestrian, for safety, should endeavor to keep their eyes and hands free – there is no room for a fixed dashboard screen to assist in presenting route directions. We take this last constraint at full force – in our prototype there is no map display; the only mode of presentation is text-to-speech instruction heard incrementally through the pedestrian's earpiece.

We focus here on the problem of providing incremental spoken route directions to guide a pedestrian from their current position to a given destination. Such a problem yields two related metrics of evaluation: *(1) what is the system's effectiveness in actually guiding pedestrians from a given initial position to a given destination position?*; *(2) how many simultaneous users can a system scale to?*. These two metrics most certainly trade off against one another. While our initial focus has mostly been on improving metric 1 measures, metric 2 is increasingly a consideration.

1.1 Organization of this report

This report describes a test-bed prototype that we implemented to explore the pure navigation case. Section 2 of this report introduces the terminology and concepts we use in our work. The terminology is based largely on that of Richer and Klippel [11], although we limit ourselves to only a subset of their terms and adapt the terminology slightly. Section 3 presents the overall architecture of the prototype. Because of the centrality of the SpatialDB in our prototype, section 4 describes the table definitions and dynamic state within the spatial database. These tables correspond to the concepts presented in section 2. Section 4 shows how we implement a 'policy' that maps from complex spatial/temporal state to commands to generate route directions. Section 5 informally reports on how our system initially performs for several tests carried out in Umeå in October 2012, Sweden. Section 6 concludes.

2 Terminology

The *path network* upon which a pedestrian may be directed to travel is made up of *branching points* and *path segments*, as illustrated in figure 1. For example the points labeled '5401' and '5522' are branching points and the line from point 5401 to point 5522 represents a path segment. Path segments are directed and are often not straight line edges, but rather

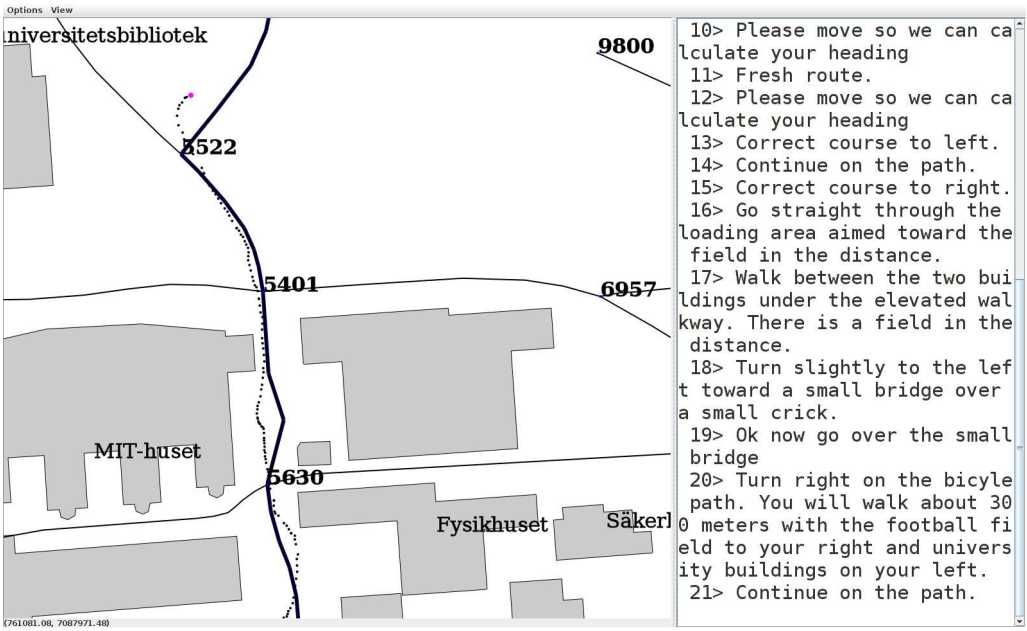

Figure 1: Guiding a pedestrian on a route on Umeå University's campus

are defined as a sequence of directed *elementary segments* that, chained together, represent curves or the meandering of a path segment. The path segment from branching point 5401 to branching point 5522 consists of 3 elementary segments. A *path* is a connected sequence of path segments that would take a pedestrian from some origin branching point to a destination branching point. In figure 1 we see a marked path with an origin and a destination that is off the map. A *route* demarcates a path, consisting of *route segments* and *decision points* which in turn demarcate associated path segments and branching points of the path.

Landmarks are entities in space that have associated point, linear or polygonal geometries. Landmarks also have associated *types* (restaurant, bar, museum, street, park, university_building, etc.) and names (e.g. 'MIT-Huset', 'Fysikhuset', etc.). There is a general linking relation that allows arbitrarily named *properties* and *values* to be associated with landmarks (color, architectural style, etc.).

3 System Overview

In figure 2 we see the main components of our architecture. The key components are the PhoneApp, the SpatialDB, the RoutePlanner and finally the Controller.

The PhoneApp runs on the user's Android phone and logs GPS measurements as longitude, latitude, antenna error triples every second to the SpatialDB. The PhoneApp likewise

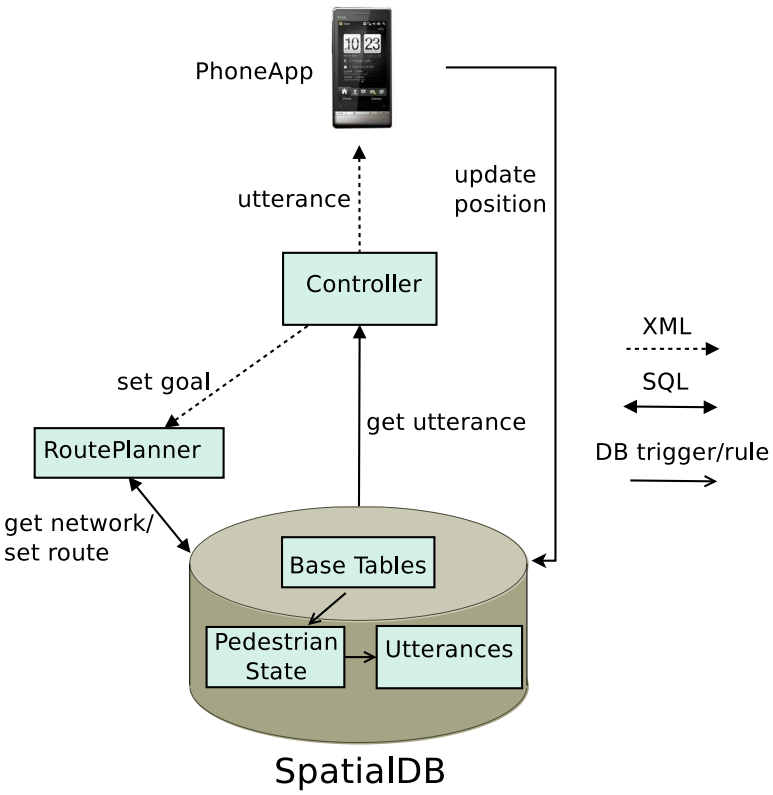

Figure 2: The basic architecture

accepts text messages from the CONTROLLER that are voiced using Google's text-to-speech engine running on Android.

The SpatialDB is the single repository for all *state* in the system. This means that the SpatialDB represents the path network, the landmarks, the GPS measures, routes and a log of previously issued utterances. In addition there is a state table that is dynamically derived via *stored procedures*. These state tuples capture the complex spatial and communication state of the pedestrian through time. Finally the SpatialDB contains a set of communication rules that select the utterance, if any, that should be voiced to the pedestrian. The SpatialDB is implemented within PostGIS/PostgreSQL, using both PostgreSQL triggers and rules with stored procedures implemented in PL/PGSQL.

The ROUTEPLANNER is a simple component that plans a route from the branching point closest to the pedestrian's current position to a destination branching point within the path network. The method used is simply A* search using a *straight line distance* heuristic run over the path network with cost based on path segment length[10]. To set up the search, the

RoutePlanner issues an SQL query to the SpatialDB to bring in the relevant part of the path network. After performing the search, the RoutePlanner inserts the result into the `Route` table in the SpatialDB.

The Controller runs a very simple (less than 20 lines of Java code) control loop that polls the database for what utterance message to send next to the PhoneApp and when it is necessary to invoke the RoutePlanner for a new goal.

There are two infrastructure components that do not appear in figure 2, but should be mentioned for the sake of completeness: the PhoneServer and the IceBroker. The PhoneServer represents the PhoneApp in the back end and shunts GPS position reports to the SpatialDB as well as shunting text message issued by the Controller onward to the PhoneApp for voicing. The IceBroker allows components to publish and subscribe to data streams (e.g. GPS data) or to issue remote procedure calls on other components (e.g. executing SQL queries, etc). It represents a slightly higher level of abstraction and functionality than a pure socket-based client server protocol would support [4].

4 The SpatialDB

Because of the centrality that the SpatialDB plays in our prototype, we describe in some detail the tables in the SpatialDB, how they are initially populated or dynamically generated.

4.1 The base tables

Figure 3 shows the base tables of the database from figure 2 grouped into tables representing the path network, landmarks, routes and the pedestrian name and time series of reported GPS positions[1]. These tables mirror exactly the terminology of section 2. Attributes named `point` or `line` and `geom` are PostGIS geometry types.

The base tables are populated by external processes that add data either at database build time or at run-time. Specifically the landmark and path network tables of figure 3 are populated at database build time by converting OpenStreetMaps XML data[1][2] to tuples in our schema. The pedestrian tables are populated by a very simple pedestrian registration process as well as the run-time GPS logging at one update per second from PhoneApp. The route tables are populated at run-time by the RoutePlanner once a call for a new goal is made by the Controller.

4.2 Pedestrian state table

Pedestrian 'state' is represented as a tuple in a single table with many attributes (`PedestrianState`). The attributes have varied types (boolean, integer, real, PostGIS geometry types, time stamps, etc.) and are described below:

[1]Note that the table and attribute names used here are slightly different in the actual implementation. We document these differences in the README file accompanying our (future) open-source software distribution.
[2]Obtained at `http://openstreetmaps.org`.

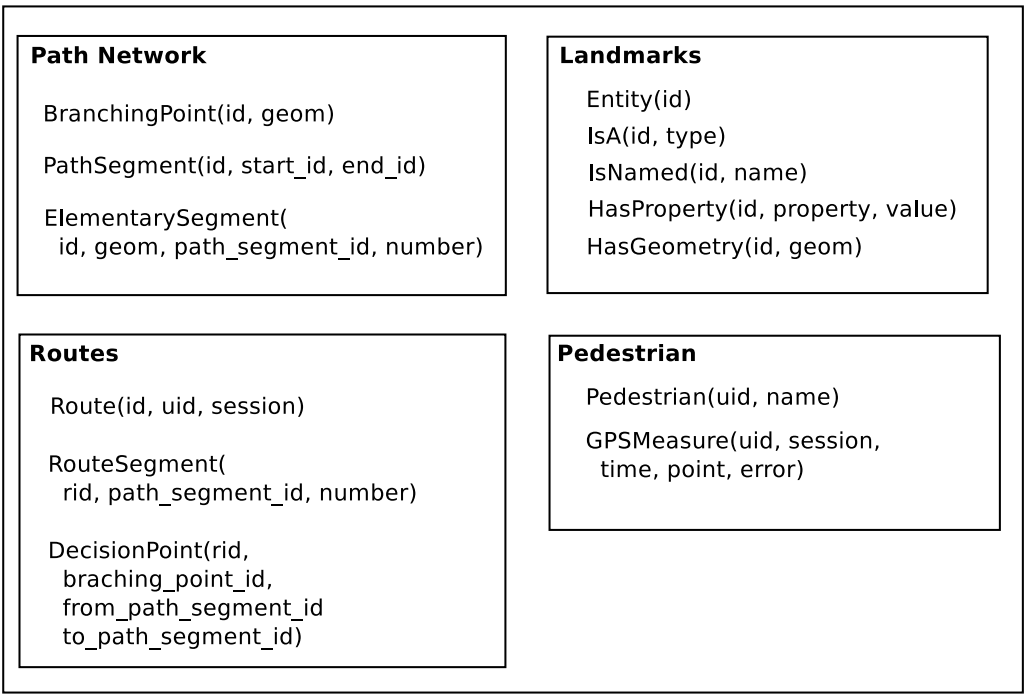

Figure 3: The base tables

`uid`: This is the pedestrian's id.

`session`: This is the session of the pedestrian.

`phone_time`: This is the time-stamp recorded on the phone by PhoneApp.

`insert_time`: This is the actual time at which the state tuple is inserted into the database. The difference between `phone_time` and `insert_time` represents the *position report latency* plus the *clock difference* between the PhoneApp and the SpatialDB.

`position`: This is the smoothed and filtered position that is the best guess as to the actual position of the pedestrian. Currently this smoothing and filtering process is very simplistic.

`GPS_error`: This is the GPS error measure reported by the PhoneApp.

`speed`: This is a smoothed value that estimates the pedestrian's speed in meters per second.

`heading`: This is a smoothed angle value (in degrees, with North at 0°, East at 90°, etc) that represents the direction in which the user is facing under the assumption that pedestrians always face in the direction that they are traveling. Quite often this field has the `NULL` value to represent that we do not have a reliable heading value for example

at the start of a session, when the user is at a stand still, when we are not getting consistent GPS measures, etc.

`on_path`: This is a boolean value that is true if the pedestrian's position is within a distance threshold (currently set at 10 meters) to an elementary segment of a path segment of the current route.

`at_branching_point`: This is a boolean value that is true if the pedestrian's position is within a distance threshold (currently set at 5 meters) to a branching point on the path demarcated by the current route.

`in_path_segment`: This is a boolean value that is true if `on_path` is true and `at_branching_point` is false. In other words `in_path_segment` is true if the pedestrian is positioned on a path segment between two branching points.

`at_goal`: This is true if the pedestrian is within a distance threshold (currently set at 10 meters) of the currently pursued goal.

`standing_still`: This is true if the last three seconds show an average speed of less than some constant, currently .5 meters per second.

`receiving_TTS`: This is true when an utterance is being voiced on the PhoneApp.

`heading_correction`: This is a computed angle that gives the clockwise rotation necessary to align the pedestrians heading with the heading of the elementary segment that they are currently on. In the case that the pedestrian is not on an elementary segment of a path segment, this angle is the 'overland' best correction to their current heading.

`current_goal`: This is the id of the current goal.

`current_tour`: This is the id of the current tour.

`euclidean_distance_to_goal`: This is the current distance 'as the crow flies', from the pedestrians position to the goal.

`path_distance_to_goal`: This is the current summed distance of all elementary path segments between the pedestrian and the goal (plus any additional distance required to get on a path segment in the case that the user is not already there).

`heading_toward_goal`: This is the angle heading that the goal is in from the user as the crow flies.

A PostgreSQL trigger on inserts into the `GPSMeasure` table executes a stored procedure that builds and inserts a state tuple into `PedestrianState`. Although this requires a fair bit of calculation, given that states only need to be calculated once per second, currently the representation is being calculated well under budget in single user trials. Since inserts into the `GPSMeasure` table are once per second, so too are inserts into into the `PedestrianState` table. Thus we record a pedestrian state for every second of their session, when testing the system with a single pedestrian.

4.3 Communication rules

As we monitor pedestrian state, we need to decide when and which utterances to voice to the pedestrian to guide them to their goal. In this initial prototype we model this as a simple *reactive system* implemented in a set of *event-condition-action* (ECA) rules [6] on the `PedestrianState` table. We term these *communication rules* where the *events* are inserts into `PedestrianState` table, *conditions* are queries on the inserted tuple possibly joined with additional tables and *actions* are inserts into an `Utterance` table (see figure 5). The `Utterance` table records exactly which utterances will be, or have been, voiced to the pedestrian.

```
1  CREATE RULE TurnThroughDecisionPoint AS
2  ON insert TO PedestrianState
3  WHERE  NEW.onPath AND NEW.atBranchingPoint
4         AND NOT NEW.receivingTTS
5  DO ALSO (
6    INSERT INTO Utterance(uid, session, time, utterance)
7      VALUES(
8        NEW.uid,
9        NEW.session,
10       NEW.time,
11       currentTurnUtterance(NEW.uid));
12   );
```

Figure 4: An example communication rule

An example communication rule appears in figure 4. Following the syntax of PostgreSQL, rules are named (`turnThroughDecisionPoint` in line 1) with specification of events (e.g. line 2), conditions (e.g. lines 3-4) and actions (e.g. lines 6-11). The purpose of this rule is to direct the pedestrian on to the next route segment as they arrive at a decision point.

Additional rules are:

`continueRouteSegment`: The action is to encourage the pedestrian to continue following the route segment they are in. The conditions for this are that the pedestrian is making good progress on a route segment. An analogous rule is defined over elementary segments.

`correctHeadingToLeft`: The action is to tell the pedestrian to correct their course to the left. The condition is that the pedestrian is veering off the current elementary segment to the right. Analogous rules are `correctHeadingToRight` and `correctHeadingTurnAround`.

`offPathHeadingCorrection` Given to direct the pedestrian toward the most appropriate path segment. The condition is that the pedestrian is not on any path segment of the current route.

`distanceReport` given to report how much further to the goal.

`orientToGoal` given to report that the pedestrian is facing the direction of their goal.

`encourageMovement` given to inform the pedestrian that they need to walk so that a heading can be calculated.

When multiple communication rules simultaneously have true conditions, only one is allowed to generate an utterance. This is guaranteed by implementing a separate PostgreSQL rule on inserts into the `Utterance` table. Lexicographic order on rule names determines an a precedence relation among communication rules.

4.4 Generation and realization of utterances

When a communication rule inserts an utterance into the `Utterance` table, it must be voiced (i.e. *realized*) to the pedestrian. This is achieved by the Controller which polls the the `Utterance` table, shunting new utterances to the PhoneApp. The Controller currently does this once per second.

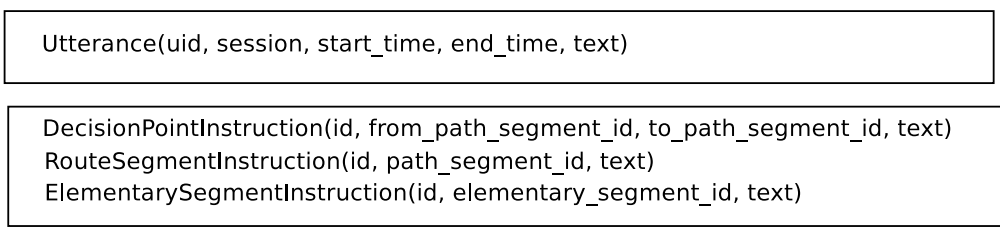

Figure 5: The utterance and pre-generated instructions tables

Currently utterances are pre-generated via an off-line natural language generation package that systematically computes route instructions (possibly including references to landmarks) over all possible decision points and route segments. Authoring tools enable us to override default generated utterances with human written content. For example utterances 16-20 in figure 1 were authored into the system.

No matter their source, pre-generated utterances are stored in the tables `DecisionPointInstruction`, `RouteSegmentInstruction` and `ElementarySegmentInstruction` shown in figure 5. Stored procedures (e.g. `currentTurnUtterance` in figure 4) retrieve these utterances and insert them into the `Utterance` table for immediate realization.

5 Initial Observations

We have implemented the prototype described in this report and conducted a series of pilot tests. Most of our tests were dedicated to validating capabilities and confirming bug fixes.

While we we have not yet run any formal experiments, as of October 2012 we have developed the system to a state that will soon be sufficiently robust and effective for navigation experiments with random human subjects.

5.1 Run time performance and stability

The run time performance of the system is adequate. For a typical test under the conditions depicted in figure 1, the average time to calculate the pedestrian states was well under the 1 second budget. The average lag time of GPS reports are approximately 60 ms.

The stability of the system has improved substantially from our first running prototype (in June 2012) to the prototype at the end of our latest development phase (in October 2012). These stability issues were addressed mostly by redesigning and recoding initial components. In addition we have systematically tracked known problems and feature requests using the REDMINE bug tracker. Our prototype must be very reliable before we commit substantial resources to evaluation.

5.2 Effectiveness of navigation

To be blunt, our initial implementation, before any experiences were gathered and parameters tweaked, would not have been able to reliably guide a user to a goal. For example problems like the quantity and timing of utterances (too much or too little speech, utterances issued too late or too early) and oscillations in the calculation of facing direction led to a frustrating user experience. Thus much effort was directed toward fixing parameters in the underlying system, coding alternative phrasings, adding further communication rules and state variables, etc. In addition we determined that scheduling of utterances in synchronization with user position is a critical capability that is not easily finessed in our purely reactive approach. This orients our future efforts toward the challenging problem of predicting user position and scheduling utterances accordingly.

Testing with the VirtualPedestrian

While figure 1 shows the display of our VIRTUALPEDESTRIAN tool in tracker mode, figure 6 shows our VIRTUALPEDESTRIAN tool in a 'virtual mode'. In this mode the user controls the heading and speed of the pedestrian on a map. The map portion of the tracker shows the plot of the GPS positions along with their inferred position. The portion to the right gives the log of utterances that the system generates. The VIRTUALPEDESTRIAN in 'virtual' mode engages in exactly the same protocol that the PHONEAPP engages in with the system. Moreover the tool can simulate GPS error and the route can be hidden from the human operator and text can be voiced as text-to-speech. In this mode we can run tests and perhaps even evaluation without having to go out in the streets.

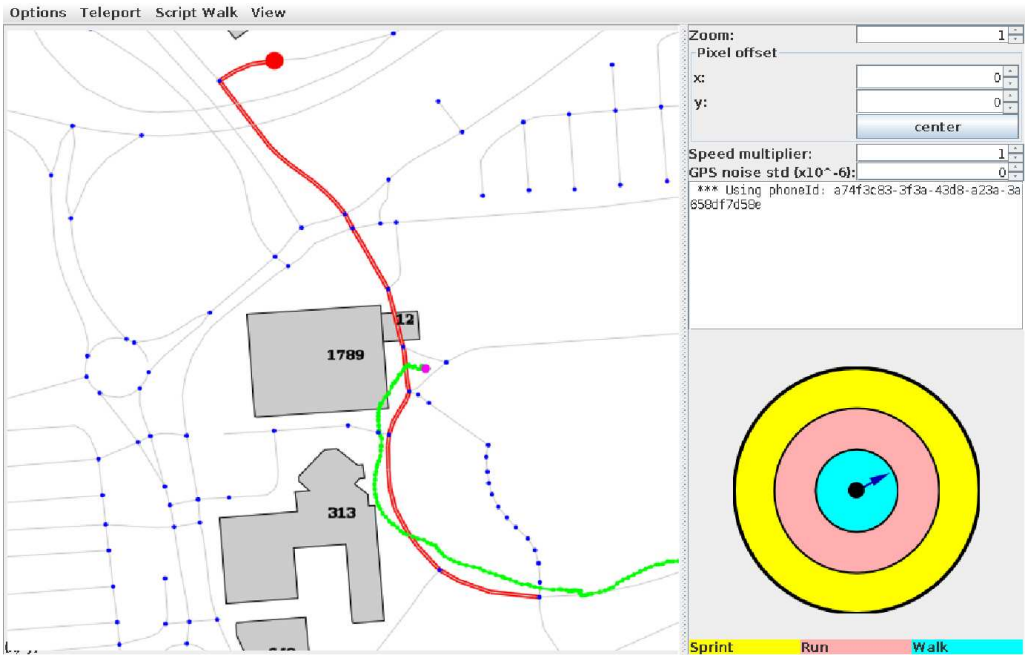

Figure 6: A human subject using VirtualPedestrian to follow a virtual route.

Actual field tests

No matter how many virtual pedestrian tests we run, what counts is how the system actually performs with real users in the field who do not know the system, but can only follow the instructions that the system voices. Our plan here is to start field testing by generating random tours unknown to a single human tester. Since the field tester will be unaware of the destination that they are currently being guided to, the system will need to be effective in guiding the tester through route following instructions. Even if the tester is one of the authors of this report, this will give us insight into the effectiveness of various communication strategies. No doubt other unforeseen issues will also come up. Only after performing this cheaper form of *auto-evaluation* (or testing) shall we consider a larger evaluation with random human subjects.

6 Conclusions

This report has provided a snap-shot description (as of October 2012) of our work on building a pedestrian navigation system based on active database technology. Although we have only performed cursory testing so far, we believe that the system holds out promise as a

scalable platform to effectively guide pedestrians to goals in the city. The ability to author content directly into the system gives a practical approach to override machine generated descriptions. Moreover this authoring approach may underlie a future method by which textual descriptions (or perhaps even audio content) may be crowd sourced.

The work here also brought to the surface some interesting query processing issues. Because of the noisy nature of position reports, we have found statistical time-series queries of particular use. For example time series queries such as "Is the standard deviation of heading less than 10 degrees over the last 10 seconds?" are the basis of determining facts like whether we have a stable heading, which in turn is a condition for rules like `orientToGoal` of section 4.3. One can imagine even more complex time-series queries that could determine, for example, if the pedestrian is likely to be waiting for a traffic light ("Has the user walked straight up to a road and waited longer than 3 seconds in stand still?"), is disoriented ("Has the user returned to this spot after walking in a 'circle' lasting 3-4 minutes?"), is making progress ("Over the last minute has the user moved at least 30 meters nearer to the goal?"), etc.

Thus far we have not yet been forced off a traditional relational approach in favor of a stream-based approach. Since our focus has been on boosting scores on metric 1 (see section 1), we have in fact been limited to running 30 minute tests with single users – in essence isolating our attention to windows of no more than 2000 tuples in the `PedestrianState` table. As we transition to larger scale studies and in particular explore methods to crowd source audio route instructions, we anticipate transitioning to a stream based approach [5]. With our attention firmly focused on our two evaluation metrics, it will be interesting to see how this project progresses.

7 Acknowledgments

Our schema design for the path network and landmarks was developed through earlier discussions with William Mackaness and Phil Bartie of SpaceBook partner Edinburgh University. The basic notion of basing communication actions on dynamic pedestrian state is inspired by the work of Oliver Lemon's group at SpaceBook partner Heriot Watt University. Initially we used a PhoneApp (and server) implemented by SpaceBook partner Liquid Media. Finally we would like to acknowledge the programming work of Markus Karlsson and Linus Närva at Umeå University.

References

[1] S. Coast. How OpenStreetMap is changing the world. In *proc. of W2GIS*, page 4, 2011.

[2] R. Dale, S. Geldof, and J.-P. Prost. Using natural language generation in automatic route description. *Journal of Research and Practice in Information Technology*, 37(1), 2005.

[3] M. Duckham, S. Winter, and M. Robinson. Including landmarks in routing instructions. *J. Locat. Based Serv.*, 4(1):28–52, Mar. 2010.

[4] M. Henning. A new approach to object-oriented middleware. *IEEE Internet Computing*, 8(1):66–75, Jan. 2004.

[5] S. J. Kazemitabar, U. Demiryurek, M. H. Ali, A. Akdogan, and C. Shahabi. Geospatial stream query processing using Microsoft SQL Server StreamInsight. *proc. of VLDB*, 3(2):1537–1540, 2010.

[6] W. Kim, editor. *Modern Database Systems: The Object Model, Interoperability, and Beyond*. ACM Press and Addison-Wesley, 1995.

[7] W. Mackaness, J. Boye, S. Clark, M. Fredriksson, H. Geffner, O. Lemon, M. Minock, and B. Webber. The spacebook project: Pedestrian exploration of the city using dialogue based interaction over smartphones. In *Proceedings of the 8th Symposium on Location-Based Services (LBS)*, Vienna, Austria, 2011.

[8] Y. Miyazaki and T. Kamiya. Pedestrian navigation system for mobile phones using panoramic landscape images. In *SAINT*, pages 102–108, 2006.

[9] C. Nothegger, S. Winter, and M. Raubal. Computation of the salience of features. *Spatial Cognition and Computation*, 4:113–136, 2004.

[10] J. Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[11] K.-F. Richter and A. Klippel. A model for context-specific route directions. In *Spatial Cognition*, pages 58–78, 2004.

[12] M. Roth and A. Frank. Computing em-based alignments of routes and route directions as a basis for natural language generation. In *COLING*, pages 958–966, 2010.

[13] M. Theune, D. Hofs, and M. V. Kessel. The virtual guide: A direction giving embodied conversational agent. In *Proc. of Interspeech 2007*, pages 27–31, 2007.

IV

A Test-Bed for Text-to-Speech-Based Pedestrian Navigation Systems

Michael Minock[1], Johan Mollevik[2], Mattias Åsander[2], and Marcus Karlsson[2]

[1] School of Computer Science and Communication (CSC),
Royal Institute of Technology KTH, Stockholm, Sweden
[2] Department of Computing Science, Umeå University, Umeå, Sweden

Abstract. This paper presents an Android system to support eyes-free, hands-free navigation through a city. The system operates in two distinct modes: *manual* and *automatic*. In manual, a human operator sends text messages which are realized via TTS into the subject's earpiece. The operator sees the subject's GPS position on a map, hears the subject's speech, and sees a 1 fps movie taken from the subject's phone, worn as a necklace. In automatic mode, a programmed controller attempts to achieve the same guidance task as the human operator.

We have fully built our manual system and have verified that it can be used to successfully guide pedestrians through a city. All activities are logged in the system into a single, large database state. We are building a series of automatic controllers which require us to confront a set of research challenges, some of which we briefly discuss in this paper. We plan to demonstrate our work live at NLDB.

1 Introduction

The automated generation of route directions has been the subject of many recent academic studies (See for example the references in [1], or the very recent works [2,3]) and commercial projects (e.g. products by Garmin, TomTom, Google, Apple, etc.). The pedestrian case (as opposed to the automobile case) is particularly challenging because the location of the pedestrian is not just restricted to the road network and the pedestrian is able to quickly face different directions. In addition, the scale of the pedestrian's world is much finer, thus requiring more detailed data. Finally the task is complicated by the fact that the pedestrian, for safety, should endeavor to keep their eyes and hands free – there is no room for a fixed dashboard screen to assist in presenting route directions. We take this last constraint at full force – in our prototype there is no map display; the only mode of presentation is text-to-speech instruction heard incrementally through the pedestrian's earpiece.

We present a system to support eyes-free, hands-free navigation through a city[1]. Our system operates in two distinct modes: *manual* and *automatic*. In manual

[1] The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 270019 (SpaceBook project `www.spacebook-project.eu`) as well as a grant through the Kempe foundation (`www.kempe.com`).

E. Métais et al. (Eds.): NLDB 2013, LNCS 7934, pp. 396–399, 2013.

mode, an operator guides a subject via text-to-speech commands to destinations in the city. The operator, working at a stationary desktop, receives a stream of GPS, speech and camera image data from the subject which is displayed in real time to the operator (see figure 1). In turn the operator types quick text messages to guide the subject to their destination. The subject hears the operator's instructions via the text-to-speech engine on their Android. In automatic mode the human operator is missing, replaced by a programmed controller.

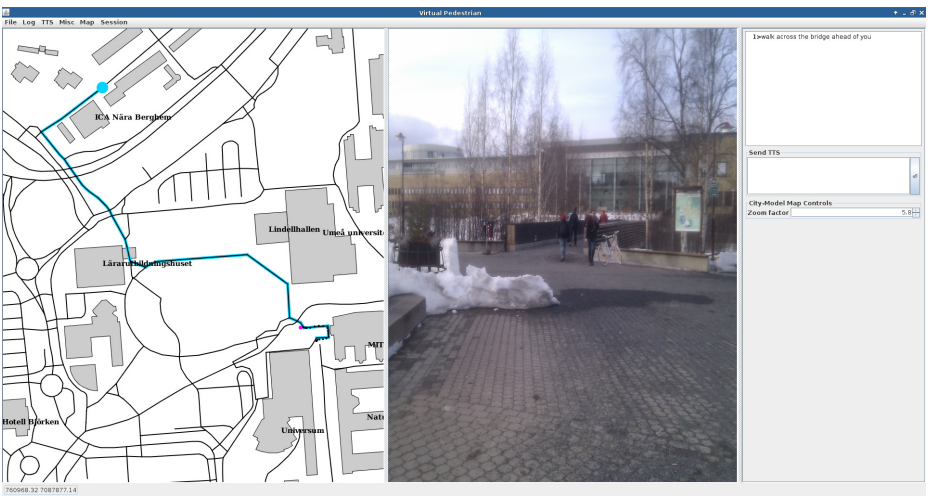

Fig. 1. Operator's interface in manual mode guiding a visitor to ICA Berghem, Umeå

The technical specification and design of our system, with an initial reactive controller, is described in a technical report [1]. That report gives a snap-shot of our system as of October 2012. In the ensuing months we have worked to optimize, re-factor and stabilize the system in preparation for its open source release – working name **JANUS** (Interested readers are encouraged to contact us if they wish to receive a beta-release). We have also further developed the infrastructure to integrate FreeSWITCH for speech and some extra mechanism to handle image streams. Finally we have added a facility that logs phone pictures to PostgreSQL BLOBs, the TTS messages to PostgreSQL text fields, and the audio-streams to files on the file system. Aside for server-side PL/pgSQL functions, the system is written exclusively in Java and it uses ZeroC ICE for internal communication. Detailed install instructions exists for Debian "wheezy".

2 Field Tests

We have carried out field tests since late Summer 2012. The very first tests were over an area that covered the Umeå University campus extending North to Mariahem (An area of roughly 4 square kilometers, 1788 branching points,

3027 path segments, 1827 buildings). For a period of several weeks, the first author tested the system 3-4 times per week while walking or riding his bicycle to work and back. The system was also tested numerous times walking around the Umeå University campus. A small patch of the campus immediately adjacent to the MIT-Huset was authored with explicit phrases, overriding the automatically generated phrases of a primitive NLG component (see the example in [1]). These initial tests were dedicated to validating capabilities and confirming bug fixes and getting a feel for what is and is not important in this domain. For example problems like the quantity and timing of utterances (too much or too little speech, utterances issued too late or too early) and oscillations in the calculation of facing direction led to a frustrating user experience. Much effort was directed toward fixing parameters in the underlying system, adding further communication rules and state variables, etc.

In addition to these tests, in November 2012 we conducted an initial test of our manual interface in Edinburgh (our database covered an area of roughly 5 square kilometers, 4754 branching points, 9082 path segments, 3020 buildings) – walking the exact path used in the Edinburgh evaluations of the initial SpaceBook prototype developed by SpaceBook partners Heriot-Watt and Edinburgh University [2]. With the PhoneApp running in Edinburgh and all back-end components running in Umeå, the latencies introduced by the distance did not render the system inoperable. Note that we did not test the picture capability at that time, as it had not yet been implemented.

Due to the long Winter, we have conducted only a few outdoor tests with the system from November 2012 to April 2013. What experiments we have run, have been in an area surrounding KTH in Stockholm (An area slightly over 2 square kilometers, 1689 branching points, 3097 path segments, 542 buildings), the center of Åkersberga, and continued tests on the Umeå University campus. With the warming of the weather we look forward to a series of field tests and evaluations over the Spring and Summer of 2013.

3 System Performance

Our optimization efforts have been mostly directed at minimizing latencies and improving the performance of map rendering in our virtual pedestrian/tracking tool. There are three latencies to consider from the PhoneApp to the controller (GPS report, speech packet, image) and one latency to consider from the controller to the PhoneApp (text message transmission). We are still working on reliable methods to measure these latencies and, more importantly, their variability. In local installations (e.g. back-end components and PhoneApp running in Umeå) the system latencies are either sub-second or up to 1-2 seconds – a perfectly adequate level of performance. Running remotely (e.g. back-end components running in Umeå and PhoneApp in Edinburgh) appears to simply add a fixed constant to all four latencies.

All the map data is based on XML exports of OpenStreetMap data converted to SQL using the tool osm2sb (see [1]). We have limited our attention

to what may be downloaded as XML exports via OpenStreetMap's web-site. This has covered large enough portions of the city for our purposes. That said, we strongly believe that inefficient access to larger maps is not a significant risk.

4 Some Future Challenges

Evaluations: We have a very natural metric of evaluation: *what is a controller's effectiveness in actually guiding pedestrians from a given initial position to a given destination position?* To minimize expense, we will first employ what we term *auto evaluation*. In auto evaluation one generates random tours, unknown to the subject, over a large number of possible destinations. Because destinations are hidden, even if one of the authors serves as a subject, we will gain insight into the relative effectiveness of various controller strategies. Only after performing this cheaper form of evaluation shall we carry out a larger classical evaluations with testable hypotheses, large cohorts of random subjects, control groups, etc.

Scheduling of Utterances in Synchronization with User Position: Early in our testing we found that scheduling of utterances in synchronization with user position is a critical capability that is not easily finessed in a reactive controller. Thus we have started work on the challenging problem of predicting user position and scheduling utterances accordingly. This is briefly discussed in [4] and will be presented in greater detail in a future conference paper.

Reuse of Operator Utterances in Automatic Controllers: Our current controllers fetch pre-compiled utterances populated via primitive NLG routines run off-line. While we will explore techniques to integrate run-time NLG systems, we are interested in techniques to re-use utterances expressed by human operators (in manual mode) within our automatic controllers. We seek large collections of human authored utterance variations, where, given a large number of user trials, we might learn a policy to select when and where to issue utterances to maximize expected utility over our metric of evaluation.

References

1. Minock, M., Mollevik, J., Åsander, M.: Towards an active database platform for guiding urban pedestrians. Technical Report UMINF-12.18, Umeå University (2012), `https://www8.cs.umu.se/research/uminf/index.cgi?year=2012&number=18`
2. Janarthanam, S., Lemon, O., Liu, X., Bartie, P., Mackaness, W., Dalmas, T., Goetze, J.: A spoken dialogue interface for pedestrian city exploration: integrating navigation, visibility, and question-answering. In: Proc. of SemDial 2012, Paris, France (September 2012)
3. Boye, J., Fredriksson, M., Götze, J., Gustafson, J., Königsmann, J.: Walk this way: Spatial grounding for city exploration. In: Proc. 4th International Workshop on Spoken Dialogue Systems, IWSDS 2012, Paris, France (November 2012)
4. Minock, M., Mollevik, J.: Prediction and scheduling in navigation systems. In: Proceedings of the Geographic Human-Computer Interaction (GeoHCI) Workshop at CHI (April 2013)

V

Prediction and Scheduling in Navigation Systems

Michael Minock
KTH: Royal Institute of Tech.
Stockholm, Sweden
minock@csc.kth.se

Johan Mollevik
Umeå University, Dept of CS
Umeå, Sweden
mollevik@cs.umu.se

ABSTRACT

This position paper makes a case for the need to *predict* pedestrian position and *schedule* communication acts in mobile navigation systems. In our work, carried out in the context of a voice guided city navigation system, we have found that improperly timed route instructions are a major cause of failure in guiding pedestrians in unknown environments. Furthermore, the need to communicate other information while guiding users on routes, as well as complications caused by network latencies, occurs often enough to require that we be able synchronize communication acts with user position as they follow a route. This has led us to focus our efforts on scheduling utterances to maximize route following success.

In this position paper we motivate this problem and present our initial approach and findings which should be of interest to others engaged in similar efforts in both the Geography and HCI communities.

Author Keywords

location-based systems; natural user interfaces; navigation systems; pedestrian interfaces; open street maps

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): Algorithms, Design, Human Factors, Reliability

INTRODUCTION

The automated generation of route directions has been the subject of many recent academic studies (see, for example [10, 9, 4, 8, 11, 5, 2, 6]) and commercial projects (e.g. products by Garmin, TomTom, Google, Apple, etc.). While most focus has been dedicated to automobile drivers, there has also been an effort to provide route directions to pedestrians (e.g. Google and SIRI). The pedestrian case is particularly challenging because the location of the pedestrian is not just restricted to the road network and the pedestrian is able to quickly face different directions. In addition the scale of the pedestrian's world is much finer, thus requiring more detailed data representation. Finally the task is complicated by the fact that the pedestrian, for safety, should endeavor to keep their eyes and hands free – there is no room for a fixed dashboard screen to assist in presenting route directions. We take this last constraint at full force – following [1], in our prototype there is no map display; the only mode of presentation is text-to-speech instructions heard incrementally through the pedestrian's earpiece.

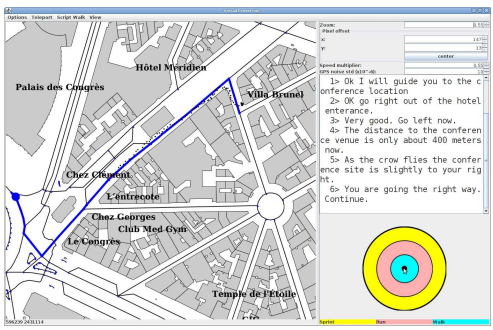

Figure 1. Guiding a pedestrian on a route near GeoHCI's venue.

Thus we focus on the problem of providing incremental spoken route directions to guide a pedestrian from their current position to a given destination. Such a problem yields a direct metric of evaluation: *what is the system's effectiveness, measure in time to destination or minimized deviation from route, in guiding pedestrians from a given initial position to a given destination position?*; The narrow position that we argue in this paper is that measures on this metric will be boosted considerably for systems that explicitly predict user positions and use scheduling to synchronize utterances with user position. We shall ultimately test this position by systematically comparing a *scheduling approach* with a *reactive approach*. The broader position argued for in this paper is that general user position prediction and scheduling of communication actions can be supported in modular components that can, without difficulty, be integrated into a variety of location-based navigation applications.

CONCEPTS

Consider Figure 1. Here we see a map, based on an export of OpenStreetMaps data[3], of a portion of Paris near to CHI's conference venue. We also observe a path that demarcates a route that the user is following (we shall follow the terminology of [10]). On the right-hand-side of Figure 1 we see a record of utterances that have been issued to the pedes-

GeoHCI Workshop at *CHI 2013*, April 2728, 2013, Paris, France.

trian to guide them along their path. Since our goal is to maximize the probability that the user follows the path, there is a mix of simultaneous *strategies* that are employed with little rest between utterances.

While the main strategy is to describe the turning actions (*turn sharp right now!, you will be turning right in about 20 seconds*), another strategy is to give positive feedback to encourage the user that they are pursuing the right path (e.g. *You are going the right way, continue walking.*). Another strategy is designed to inform the user that they are walking in the direction of their goal (e.g. *You are facing directly in the direction of your goal. It is approximately 400 meters away*). Yet another strategy provides descriptions of what the user should be seeing along the way (e.g. *you should see a large white building about 200 meters in the distance.*).

THE SCHEDULED APPROACH

Let us consider how one could support these simultaneous strategies and contrast two primary approaches: a *reactive approach* versus a *scheduled approach*. In the *reactive approach*, the system waits until the user arrives at certain points or, more generally, their trajectories meet certain conditions. At such points events are triggered which result in utterances being generated and voiced on the device. If this is implemented on the server side, then there will be some latency before the actual voicing of the utterance. Also if a particularly long utterance is being voiced, or if the user is moving more quickly than anticipated (e.g. on a bicycle), then the system may in fact miss presenting turn instructions in time. We have implemented a reactive approach earlier [7] and it often had such problems.

We contrast a reactive approach with a more sophisticated, *scheduled approach*. In such an approach, a schedule of future utterances is maintained. The most important utterances are associated with turning actions at decision points. Still, given that often the user will be traversing a path segment, other strategies also have room for their associated utterances to be scheduled. Utterances have start times, durations and pragmatic effects (e.g. enabling the user to correctly turn at a given branching point). The start times are projections into the future for when a given utterance will be issued. Once this time becomes equal to the current time, plus predicted latency, the call to voice the utterance is invoked. Obviously scheduled utterances may not overlap in time.

An interesting aspect of the scheduled approach is that within it one must represent a model of the user from which to generate predictions. This includes predicting the pace that the user will follow the route as well as the effect that utterances will have on their path. Such models can be more or less sophisticated. A simple model, that we term *inertial*, assumes that the user follows instructions perfectly and that their speed is constant (determined by sampling). That is the user continues in their given direction at their given pace, and responds perfectly to turn commands. Strictly speaking, such a model does not compel anything other than the most basic turning utterances. A slightly more complex model, where we assume that the user has a probability of making wrong turns (or failing to make turns), will explain the addition of extra utterances so long as these utterance's pragmatic effect is modeled as decreasing the probability that the user makes a wrong decision. There are many interesting user models to be developed around this problem, and much to be evaluated empirically with real pedestrians. One particularly interesting avenue of work will involve learning probabilistic models of the user from large samples of observed user data.

Other interesting issues are algorithmic and systems oriented. For example, in the most general case, we will wish to calculate a schedule that maximizes expected utility over a probabilistic user model. Note that the given user model may be considered orthogonal to the scheduling algorithmic, so long as it (the user model) is probabilistic. The computational complexity of the scheduling algorithm must be reduced to within bounds that allow real-time deliberation on modern hardware. In addition we will consider what parts of the calculation should occur on the mobile client and which on the server. Finally there are a whole host of issues around deciding when we should call for rescheduling of utterances.

INITIAL TECHNIQUES AND DEMONSTRATION

We have developed an Android-based platform for incrementally presenting spoken route directions to guide pedestrians to destinations. Our approach [7] makes heavy use of stored procedures and triggers in an underlying POSTGIS spatial database. In fact most of the 'intelligence' of our prototype resides in database stored procedures and tables. We have a base line reactive system as well as an initial scheduling approach. The initial scheduling approach uses an inertial user model for predictions. We are actively developing newer more sophisticated user models and we are also improving our scheduling and rescheduling algorithms.

We will be able to demonstrate our system live to interested GeoHCI participants in Paris. That we will guide interested persons on a tour around the region of the conference venue. In addition we will present a video demonstrating how we built the spatial database for Paris including the definition of tours and the authoring of various utterances.

CONCLUSIONS

We have presented here our ideas on the necessity to schedule utterances for users of navigation systems. Embedded within this requirement is developing user models that can be the basis of prediction. We suspect that this argument will also apply to more general multimodal interfaces as well. Finally we anticipate that the scheduling part of the algorithm will cleanly separate from the predictive user model part, and that alternative configurations of these components will suit different architectures, platforms and applications.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 270019 (SPACEBOOK project **www.spacebook-project.eu**) as well as a grant through the Kempe foundation (**www.kempe.com**).

REFERENCES

1. Bartie, P., and Mackaness, W. Development of a speech-based augmented reality system to support exploration of cityscape. *Transactions in GIS 10*, 1 (2006), 63–86.
2. Boye, J., Fredriksson, M., Götze, J., Gustafson, J., and Königsmann, J. Walk this way: Spatial grounding for city exploration. In *Proc. 4th international workshop on spoken dialogue systems, IWSDS'2012* (Paris, France, November 2012).
3. Coast, S. How OpenStreetMap is changing the world. In *proc. of W2GIS* (2011), 4.
4. Dale, R., Geldof, S., and Prost, J.-P. Using natural language generation in automatic route description. *Journal of Research and Practice in Information Technology 37*, 1 (2005).
5. Duckham, M., Winter, S., and Robinson, M. Including landmarks in routing instructions. *J. Locat. Based Serv. 4*, 1 (Mar. 2010), 28–52.
6. Janarthanam, S., Lemon, O., Liu, X., Bartie, P. J., Mackaness, W. A., Dalmas, T., and Goetze, J. Integrating location, visibility, and question-answering in a spoken dialogue system for pedestrian city exploration. In *SIGDIAL Conference* (2012), 134–136.
7. Minock, M., Mollevik, J., and Åsander, M. Towards an active database platform for guiding urban pedestrians. Tech. Rep. UMINF-12.18, Umeå University, 2012.
8. Miyazaki, Y., and Kamiya, T. Pedestrian navigation system for mobile phones using panoramic landscape images. In *SAINT* (2006), 102–108.
9. Nothegger, C., Winter, S., and Raubal, M. Computation of the salience of features. *Spatial Cognition and Computation 4* (2004), 113–136.
10. Richter, K.-F., and Klippel, A. A model for context-specific route directions. In *Spatial Cognition* (2004), 58–78.
11. Theune, M., Hofs, D., and Kessel, M. V. The virtual guide: A direction giving embodied conversational agent. In *Proc. of Interspeech 2007* (2007), 27–31.

Department of Computing Science
Umeå University, SE-901 87 Umeå, Sweden
www.cs.umu.se

ISBN 978-91-7459-777-6
ISSN 0348-0542
UMINF 13.22