
# D3.1.2: The SPACEBOOK City Model

Michael Minock, Johan Mollevik,
William Mackaness, Phil Bartie

Distribution: Public

SpaceBook
Spatial & Personal Adaptive Communication Environment: Behaviors & Objects & Operations & Knowledge
270019 Deliverable 3.1.2

March 1st, 2013


Project funded by the European Community under the Seventh Framework Programme for Research and Technological Development


*The deliverable identification sheet is to be found on the reverse of this page.*

| | |
|---|---|
| **Project ref. no.** | 270019 |
| **Project acronym** | SpaceBook |
| **Project full title** | Spatial & Personal Adaptive Communication Environment: Behaviors & Objects & Operations & Knowledge |
| **Instrument** | STREP |
| **Thematic Priority** | Cognitive Systems, Interaction, and Robotics |
| **Start date / duration** | 01 March 2011 / 36 Months |

| | |
|---|---|
| **Security** | Public |
| **Contractual date of delivery** | M24 = February 2013 |
| **Actual date of delivery** | March 1st, 2013 |
| **Deliverable number** | 3.1.2 |
| **Deliverable title** | D3.1.2: The SPACEBOOK City Model |
| **Type** | Report |
| **Status & version** | Final 1.0 |
| **Number of pages** | 12 (excluding front matter) |
| **Contributing WP** | 3 |
| **WP/Task responsible** | UMU |
| **Other contributors** | UEDIN |
| **Author(s)** | Michael Minock, Johan Mollevik, William Mackaness, Phil Bartie |
| **EC Project Officer** | Franco Mastroddi |
| **Keywords** | City model, GIS, PostgreSQL, PostGIS, OpenStreetMap, Navigation Systems |

The partners in SpaceBook are:

| | |
|---|---|
| **Umeå University** | UMU |
| **University of Edinburgh** HCRC | UE |
| **Heriot-Watt University** | HWU |
| **Kungliga Tekniska Högskola** | KTH |
| **Liquid Media AB** | LM |
| **University of Cambridge** | UCAM |
| **Universitat Pompeu Fabra** | UPF |

For copies of reports, updates on project activities and other SPACEBOOK-related information, contact:

The SPACEBOOK Project Co-ordinator:
Dr. Michael Minock
Department of Computer Science
Umeå University
Sweden 90187
mjm@cs.umu.se
Phone +46 70 597 2585 - Fax +46 90 786 6126

Copies of reports and other material can also be accessed via the project's administration homepage, `http://www.spacebook-project.eu`

# Contents

# Executive summary

The *city model* in the SPACEBOOK project is the main repository of information on objects and regions in the city and the path networks that pedestrians use to navigate and explore. The city data model is extensible to accommodate a growing set of types, properties and relationships that real pedestrians are interested in. The city model is scalable, being able to answer reasonable queries in sub-second time frames to support the SPACEBOOK mobile application.

This report will present our design of our city model. An earlier version of this design was successfully used within the SPACEBOOK prototype that was evaluated on the streets of Edinburgh in August and September of 2012. Our emphasis in this deliverable is to clearly document and discuss the representation choices we made. The emphasis here will not be on how this model is populated with real data over Edinburgh (that will fully described in *D3.3.2:Populated City Model*, which will be published in June, 2013.

# 1 Introduction

Relational databases are often justifiably criticized for not being flexible or extensible. The criticism is that once a conceptual model and schema are committed to, it is difficult to integrate additional types, properties or relationships without significant conceptual redesign and schema evolution. This process is cumbersome, error prone and, if not performed carefully, often results in inscrutable models, with redundant data, poor performance and inconsistencies.

In recent years many have championed semi-structured (or schemaless) approaches to data representation[2] and information integration[3]. Although this has many modern incarnations (e.g. XML without a DTD, RDF, Attribute-value stores, etc.), the basic idea is quite old and stretches back to *semantic networks*[10].

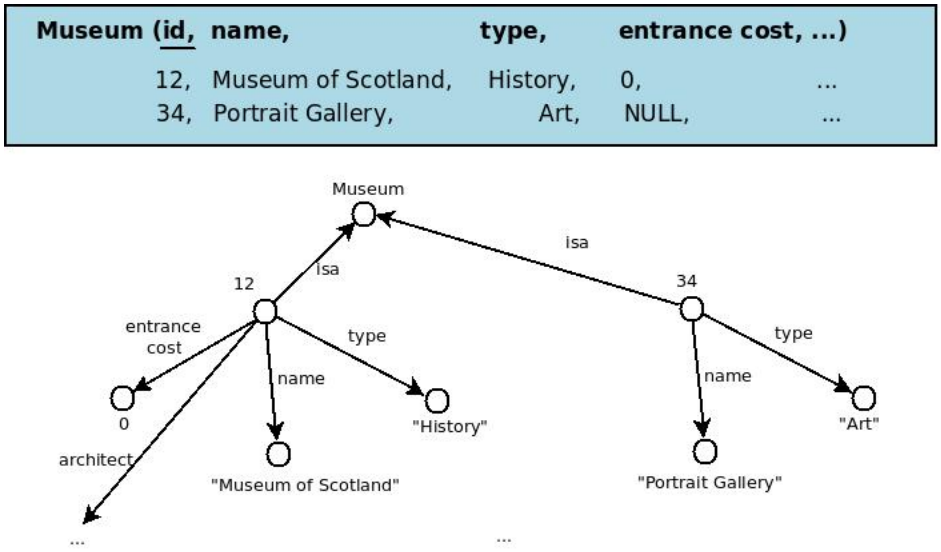


Figure 1: A relational approach versus a semantic network approach

Figure 1 illustrates these concepts in a very simple fashion. At the top of the figure we see museum information being represented as a large relation. In the bottom of the figure we see the same information represented as a semantic network. When we represent museums using a large relation, we must anticipate a finite set of attributes and represent individual museums as tuples, often with `NULL` values when we do not know the attribute's value. In contrast, if we adopt a semantic network approach we can simply add labeled edges willy-nilly to a growing network of nodes that represent concepts, instances and values. Note that the labeled edge `architect` was simply added to node 12 without any declaration of an attribute or schema alteration. This illustrates the notion of a data representation being *schemaless*.

The advantages and disadvantages of using wide relational tables versus schemaless representations are too numerous to recount here, but let us consider a hybrid approach known as *vertical partitioning* [1]. In this approach the entities of the domain are unary predicates. To express each property, we introduce a separate table with entities in the subject place and the property's value in the object place. Figure 2 shows our example represented using such vertical partitioning.

As additional properties (e.g. recording a museum's architect) come up one adds additional tables without being required to alter existing tables. To encourage an orderly extension one could adopt a naming convention such as: `VerbPhraseObject(Subject,Object)`. So, for example, to add information on

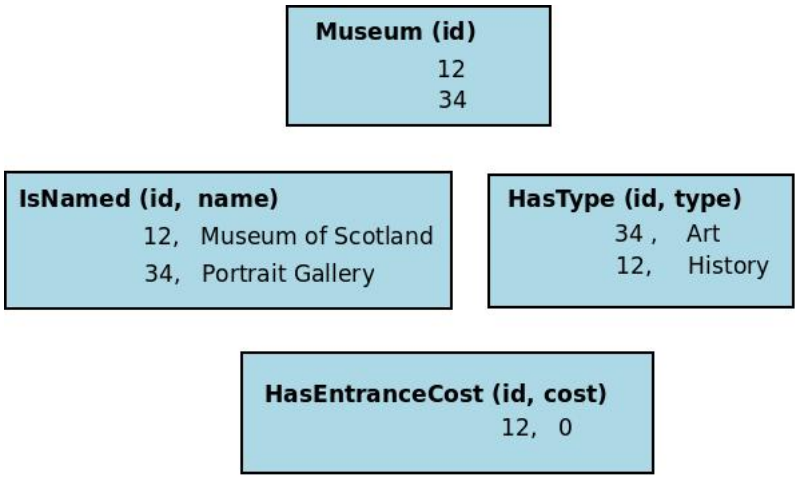


Figure 2: In vertical partitioning, each property gets a separate table.

architects, we add a table such as:

```
wasDesignedByArchitect(BuildingId,PersonId)
```

An important point in such an approach is that it leverages relational databases directly and thus can represent tables of arbitrary arity. This is often necessary when expressing spatial, temporal or other complex properties. So for example one could add `hasOppeningHours(Id,Day,Open,Close)` then we can express the fact that a museum opens at 10am and closes everyday at 9pm except Thursdays and Sundays when it closes at 7pm.

While we initially used this type of *vertical partitioning* [1] approach, we soon found it cumbersome to handle the proliferation of unary tables to represent types. Thus we reified these types in a special table `isA`. In doing so, we make asserting an additional type (e.g. a building is of type `PoliceStation`) on an object as simple as just a tuple insert. Once we took this step, it was natural to continue pushing reification through the approach and thus we adopted a similar approach and likewise reified properties via a `hasProperty` table.

## 1.1 Overview of the city model

The city model represents the objects and regions of the city, their types, geometries, associated descriptions and relationships. Moreover the model represents the groupings of entities as *sites*. The model also represents the *path-segments* and *branching points* of the underlying *path network* of the city upon which pedestrians will be directed (This nomenclature comes from [7]).

We implemented our model using a relational database system extended with spatial operators – specifically a database implemented in POSTGRESQL extended with POSTGIS. In addition to providing SQL select access to the tables of the database, several services are supported from within the database via server-side functions. This includes route finding as well as functions to give analysis reports to support wayfinding instructions.

## 1.2 Organization of this report

Section 2 presents the vocabulary of the SPACEBOOK city data model. Specifically we give the table and attribute names for relations representing the entities (object and regions), groupings of entities (called sites), as well as the path network. Section 3 presents a set of services and tools designed to work with the city model. Aside from a simple server that allows SQL access to the city model, we describe how to access the visibility engine and a route planner. Section 3 also describes a generic tool (`osm2sb`) to import basic entity and network data into the city model from OPENSTREETMAP. Section 4 discusses the overall performance and suitability of the city model and entertains some future ideas. Section 5 concludes.

# 2 Vocabulary

Here we systematically present the *vocabulary* of the SPACEBOOK city model (mostly) following the *reification* approach discussed in the introduction. By 'vocabulary' we mean the set of constant symbols, functions and predicates used within queries. The constants are standard `integer`, `real` and `text` literals as well as POSTGIS geometry literals such as `point`, `linestring`, `polygon`, etc. The functions are the standard arithmetic and string processing functions as well as POSTGIS associated functions over geometry types. In addition there are the standard built-in predicates and a set of POSTGIS spatial predicates over the geometry types.

## 2.1 Entities

| | |
|---|---|
| `entity(id)` | An entity is an object or a region. The argument **id** comes from a common namespace of `integer` values. |
| `isA(id,type)` | Each entity has a set of associated types. A given entity may be a member of multiple predicates. For example a given object could be both a bar and a restaurant. |

## 2.2 Text Valued Properties

An extensible set of text value properties are captured in the following predicates:

| | |
|---|---|
| `isNamed(id,name,namesearch)` | This associates text with an entity. The `namesearch` attribute is a tokenization of the given name represented in a `tsvector`. A postgresql rule on insert and update events maintains this tokenization. The tokenization speeds up partial string matches. |
| `hasProperty(id,property,value)`<br>`hasProperty2(id,property1, propery2,value)` | Thus far we have not been required to populate any facts in `hasProperty2` or beyond. But in principle there are n-ary type properties that would require such tables. |
| `hasVisualDescription(id,type, value)` | The City Model is able to store visual description details for entities. These may be used when assisting the user to identify an object in a view, or to find the destination. This table consists of types: is, has, opposite, nextto. This enables more intuitive ways of describing features in a form that it meaningful at the level of the pedestrian. |

Fullname and Soundex searching. The city model now makes use of the PostgreSQL extensions for fuzzy string matching. The functions are based on:

a) `text_soundex(text)`

b) full text searching - using `tsvector` data type

## 2.3 Geometry Valued Properties

| |
|---|
| `hasPoint(id,geom)`<br>`hasPolyline(id,geom)`<br>`hasPolygon(id,geom)` |
| `hasShape(id,geom)` |

## 2.4 Sites

The database is comprised of polygons, networks and points. A polygon represents an identifiable feature, but many features may also be considered as a single site. For example Edinburgh Castle consists of 326 polygons which include buildings, steps, open spaces, walkways, and other associated polygons. This is scale dependent issue since at one scale it is the totality of the feature that you are looking at, but when close up, it might its sub components you are interested in (eg from the Castle to the Chapel, Armoury, and Barracks that comprise the castle).

| |
|---|
| `sitelink(id,site_id,site_name, site_type, site_area)` |

This partonomy is handled by use of the table `sitelink` which references the polygon members of that composite feature. A polygon may be a member of many sites. For example a polygon may be both part of Scott Monument and also part of East Princes Street Gardens. This enables a rich and intuitive way of referring to the geography through which the pedestrian walks.

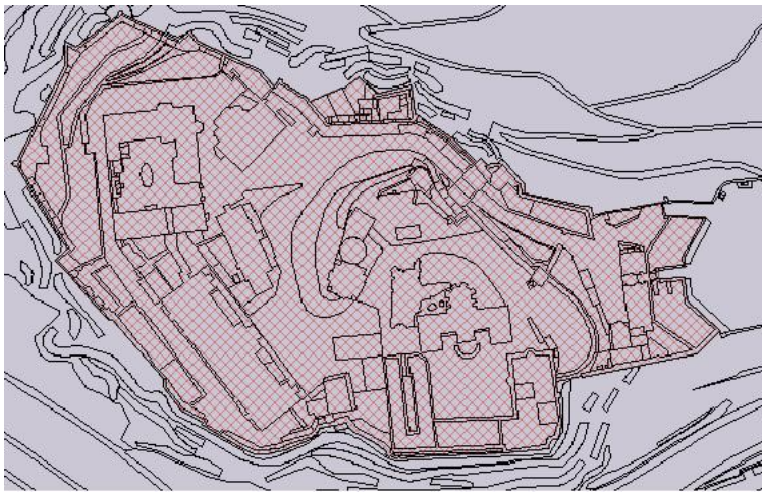

Figure 3: Edinburgh castle site and its component polygon parts

## 2.5 Path Networks

Currently we are using exactly the networks represented in OPENSTREETMAP, built by using the tool `osm2sb` described in section 3.4. These are in turn enriched using a *digital terrain model*(DTM).

| | |
|---|---|
| `node(id, geom, nodetype, z,connections)` | Corresponds to a *branching point*. The `z` value comes from the the DTM. |
| `network (id, geom, pathtype, length2d, length3d, rdlength, startpoint,endpoint, start_id, end_id, sview, midptz, sinuosity, bendpt, bends, bendang)` | Corresponds to a *path segment*.<br>The value for `length3D` is calculated across the terrain surface so includes real walked distance across hills, rather than straight line, point to point, distances.<br>The attribute `sview` is a boolean for if the road is available in StreetView for web based trials, used to change behavior of routing engine (eg not tracks, steps, Princes street and other non-accessible streets to private vehicles).<br>The attribute `midpt` is the DTM elevation at the midpoint along that edge used to calc if youre going up or downhill from a node (which have their own z elevation values)<br>The attribute `sinuosity` is the calculated ratio of edge length against Euclidean distance from start to end node to give a value for how straight or bendy that edge is.<br>The attribute `bendpt` is the greatest bend in the path segment and the `bendang` is the angle of such a bend. |
| `Network_Vertex(id,geom)` | |
| `Network_Edge(id,fromId,toId,geom, type,networkid,networksequence)` | Corresponds to a *elementary segment*. |

## 2.6 Navigation Points

The city model stores many points from various sources (eg OS, OpenStreetMap, Gazetteer for Scotland) which link to name and type data. These points are placed arbitrarily within the polygon. In this manner, the polygon is the container to those points of activities. When seeking a destination the closest street to any one of those points may not necessarily direct the user to the main entrance. In the example below the National Museum of Scotland is next to four roads and the closest road to the corresponding point would lead the user to what is the back of the building.

To overcome this a set of navigational points (navpoints) were generated for those entities with a given street address. Figure 4 shows the navpoints generated for the National Museum of Scotland, which corresponds to the main entrance.

Each polygon contains many points, each corresponding to different features contained within the defined region. A single building might contain a range of stores, public telephones, ATMs, and restaurants. This reflects the fact that a building is multi functional. We argue that this fits with a pedestrians perspective they may see it as a museum, but their reason for visiting might be to have a coffee, meet a friend, or get some money from the ATM. But it would be information overload to list all the functions of that


(a) Navpoint shown for National Museum of Scotland

(b) hasPoint entites for National Museum of Scotland and surrounding region

Figure 4: National Museum of Scotland

building, or to suggest it is an ATM, when most would agree it was a museum! To determine the most suitable name to reference each polygon various crowd sources were used including Flickr, Foursquare, web search engines. From this it was possible to rank the popularity of the various contained entities to determine the most appropriate. In this manner each polygon has a main name, with sub functions contained within it. In this way, a request for the nearest cafe would not exclude candidates that had cafe as a sub function.

| | |
|---|---|
| `navPoint(geom, id, nearest_node, on_roadid)` | where id relates to the hasPoint id `nearest_node` is the closest node id `on_roadid` is the network id that the navpoint is on these are stored in this table for speed purposes so we don't need to repeatedly carry out spatial joins. |

# 3 Services and Tools

The city model may be access through a custom TCP protocol via sending a single line XML-styled request to the server, which will reply with an XML-styled document. Each request requires an authentication value included as part of the outer tag, otherwise the request will not be processed.

## 3.1 SQL select queries over the Edinburgh City Model

The city model can accept SQL select queries over the vocabulary of section 2.

## 3.2 The route planner

We use the well known package `pgrouting` to calculate routes. Once a set of edges is returned from this algorithm, various SQL and Python functions inside the database are used to turn this into wayfinding instructions (detailed below).

## 3.3 Wayfinding Instructions

There are lots of subtle ways by which a route can be described: down the road, over the bridge, the second turning at the T junction. These are just a tiny few of the sorts of adjectives used to differentiate between paths, and avoid ambiguity at junctions. To this end we have added many hooks by which descriptions can be given.

wayfinding function includes details on:

* Topography downhill, slight downhill etc
* pathtype street, path, steps, bridge etc
* bendy or straight section (Eg St Vincent Place has a slight bend)
* the total number of exits at each junction
* the exit to take numbered clockwise from -180 to +180 degrees (UK roundabout style)
* turn angle, distance (3d distance over surface topography)
* junction type (T, Y, X, Multi, inline = change of name or pathtype)

How well known a street is, is based on crowd sourced data (eg FlickR). Well known streets are used to compress route instructions where possible by asking the user if they know how to get to a very well known street, then taking on guidance from that point onwards. This minimizes the set of instructions required to direct the pedestrian, taking into account their knowledge of the city and its streets.

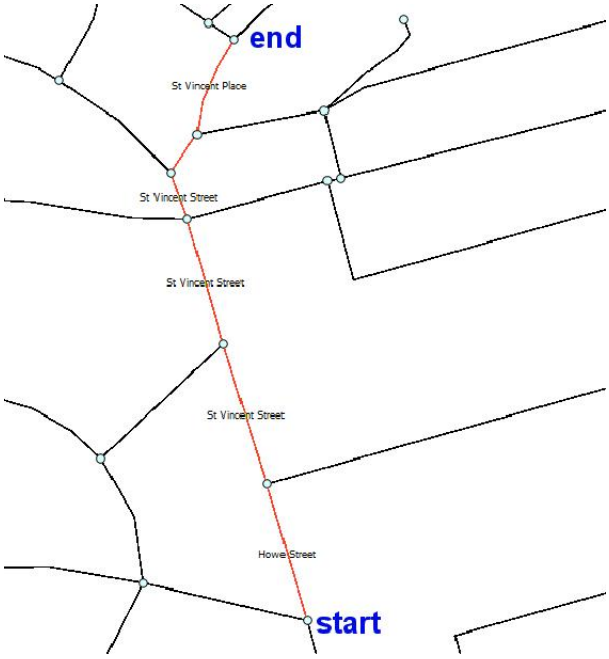


Figure 5: Route

As an example consider the following call to our server-side function `wayfinding_knownstreets_jtype` which takes the source and destination nodes as arguments:

```
select * from wayfinding_knownstreets_jtype(2007687,2007628)
```

The following table is generated, providing the vocabulary for the interaction manager and subsequent text to speech conversion.

| seq num | edge id | name | cost | path type | bendy | known score | node | next turn | next slope | exits | jtype | exit num | exit desc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 803879 | Howe Street | 49.5 | street | - | 163 | 2007658 | -0.91 | downhill | 2 | - | 1 | carry on |
| 1 | 803848 | St Vincent Street | 51.1 | street | - | 28 | 2007610 | 1.39 | downhill | 2 | - | 2 | carry on |
| 2 | 803824 | St Vincent Street | 45.1 | street | - | 28 | 2007568 | -3.83 | slight downhill | 3 | X | 2 | carry on |
| 3 | 803811 | St Vincent Street | 17.1 | street | - | 28 | 2007547 | 53.5 | flat | 2 | Y | 2 | take right fork |
| 4 | 803812 | | 16.2 | street | - | | 2007581 | -25.9 | flat | 2 | Y | 1 | take left fork |
| 5 | 803831 | St Vincent Place | 38.5 | street | slight bend | 4 | 2007628 | 0 | null | 2 | - | - | |

## 3.4 The OPENSTREETMAP Extractor (osm2sb)

We have built a software tool that takes OPENSTREETMAP files as input and return SQL files that build partial city model databases for SPACEBOOK. The OPENSTREETMAP files are generated manually by visiting the site `http://www.openstreetmap.org/`, selecting a portion the data one is interested in and then performing an export to `OpenStreetMap XML Data`. Given such an OPENSTREETMAP file, our tool `osm2sb` may be run to generate the SQL export.

```
osm2sb city.osm
```

This generates an SQL file that defines the tables of section 2 and the facts associated with the particular OPENSTREETMAP datafile. This file is then loaded into the PostgreSQL database extended with PostGIS encapsulated by the city model component. This tool is of course limited to just OPENSTREETMAP data, thus it is only partially used to obtain the data that becomes the full Edinburgh data instance. The full Edinburgh data instance uses a much wider variety of data and will be fully described in D3.3.2 (Populated City Model) which will be published in June, 2013.

# 4 Discussion

## 4.1 Scalability

Because of the critical role that the city data model plays in the SPACEBOOK project, we elected to implement the city data model in a mature relational database system, POSTGRESQL [6] extended with POSTGIS to support geometric representation and querying [8, 9]. Through our experiences we concluded fairly early that the city model

scales very well. The mostly static nature of the city model means we may build a wide variety of GIST-based indexes [4] without having to consider the costs of run-time insertion into the indexes. Still, some care must be taken to write SQL queries that use these indexes during query execution.

In the past 12 months, the Edinburgh City Model has handled 7.8 million queries with an average processing time of 4ms. This includes route finding but excludes visibility engine queries. For all queries including visibility engine the average processing time is 60ms.

## 4.2 Extensibility

We didn't push the reification approach completely through the model because of practical project considerations. For example once other project components became dependent on earlier tables names and structures, it did not make so much sense to push though needless names changes and restructurings to simply conform to an abstract naming convention and uniform approach. Still, that said, we have implemented enough of the reification approach to have gained positive experience with it. Also we note that we performed some work extending the vocabulary to capture and analyze the pedestrian's trajectory [5]. However for the Edinburgh prototype such information in represented in a pedestrian tracker module.

# 5 Conclusions

The SpaceBook project always had the ambition of two prototype systems based on the city of Edinburgh. At commencement the first city model built on theoretical models of pedestrian navigation, and practical know how from previously built systems made prior to the SpaceBook project. The first implementation of the city model resulted in field deployment and evaluation. The evaluation highlighted and further clarified the requirements of the underlying data model.

Modeling of the pedestrian highlighted the fact that unlike car navigation tied to strict topological networks along roads, pedestrians operate in free space, moving between different types of networks of spaces from footpaths, across parks, along pavements, across pedestrian road crossings. This required a model that integrated and serviced these different types of spaces.

In examining the ways that pedestrians navigate that space, it was clear that rich descriptions of space were required in order to unambiguously instruct the pedestrian and to refer to different objects relative to other features in the scene. This required us to model the city at different partonomic scales, and to consider the familiar and the vernacular information gathered from social media sites such as foursquare and flickr.

There is a close link between the positional accuracy and precision of the GNSS within the smartphone on the one hand, and the language, and the detail of the city model needed to guide the pedestrian on the other. The second version of the city model has been designed in anticipation of the vagaries of GNSS (dilution of precision) and the latency of the system. This requires snapping techniques, the grouping of instructions, and the measurement of silences between instructions so that appropriate information can be used to fill these silences without the user feeling overwhelmed, but also comforted by the knowledge that the tracker knows where the user is, and can adequately describe the environment through which the pedestrian is moving.

# References

[1] Daniel J. Abadi, Adam Marcus, and Barton Data. Scalable semantic web data management using vertical partitioning. In *In VLDB*, pages 411–422, 2007.

[2] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: from relations to semistructured data and XML*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[3] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

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

[5] Michael Minock, Johan Mollevik, and Mattias Åsander. 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`).

[6] B. Momjian. *PostgreSQL: Introduction and Concepts*. Addison Wesley, 2001.

[7] Kai-Florian Richter and Alexander Klippel. A model for context-specific route directions. In *Spatial Cognition*, pages 58–78, 2004.

[8] H. Samet. *Applications of Spatial Data Structures*. Addison-Wesley, Reading, Massachusetts, 1990.

[9] S. Shekhar and S. Chawla. *Spatial Databases: A Tour*. Prentice Hall, 2003.

[10] John F. Sowa and Alexander Borgida, editors. *Principles of semantic networks : explorations in the representation of knowledge*. The Morgan Kaufmann series in representation and reasoning. San Mateo, Calif. Morgan Kaufmann, 1991.