Ontology Oriented Storage, Retrieval and Interpretation for a Dynamic Map System*

Igor Wojnicki, Piotr Szwed, Wojciech Chmiel, and Sebastian Ernst

AGH University of Science and Technology, Department of Automatics
wojnicki@agh.edu.pl, pszwed@ia.agh.edu.pl,
wch@agh.edu.pl, ernst@agh.edu.pl

Abstract. This paper presents the Dynamic Map system, one of the key products of the INSIGMA Project. The main focus is on the map and dynamic data storage subsystem, which utilizes a spatial database and is based on ontologies. First, the data models used are described, including the OpenStreetMap-based structure for the static map and the ontology-driven structure for dynamic parameters and events. The approach to generation of database structures from OWL is described in detail, followed by descriptions of the OSM import process, the GPS tracker module, the sensor state analyzer and the event interpreter. Finally, the planned future enhancements are outlined and discussed.

Keywords: ontology, database, map, traffic, events

1 Introduction and Motivation

The paper discusses the main components of the Dynamic Map, one of the key subsystem being the result of the INSIGMA Project [?]. The aim of the Project is development of an Intelligent Information System for Global Monitoring, Detection and Identification of Threats. The Dynamic Map can be considered a composition of a spatial databases storing static, temporal and dynamic data relevant for urban traffic and a set of software modules responsible for data collection and interpretation.

The logical structure of the Dynamic Map is comprised of four layers, as shown in Fig. 1: (1) the digital static map which represents the road network and other map objects, (2) the traffic organization layer, (3) dynamic information about traffic conditions and (4) dynamic and temporary information about events such as traffic jams, accidents and weather conditions. Following the terminology used internally within the project, we will refer to the two lower layers of the Dynamic Map as the static map, and to the upper layers as the dynamic map.

The innovative concept behind the INSIGMA Project assumes that up-to-date information about the current urban traffic conditions is provided by a

* Work has been co-financed by the European Regional Development Fund under the Innovative Economy Operational Programme, INSIGMA project no. POIG.01.01.02-00-062/09.

network of traffic surveillance sensors including: video image processors, inductive loops, acoustic arrays, weather stations and onboard GPS receivers installed in vehicles. Although the system will integrate various types of sensors, we focus on data obtained from cameras, which are treated as the primary source of traffic information. Video streams originating from cameras pointing at selected streets or crossroads are processed and analyzed on-line to calculate various traffic parameters, e.g.: average speed, length of queue of vehicles approaching a crossroad or a time necessary to make a specific maneuver [?]. Results of calculations are sent to the traffic repository, the Storage System, on a regular basis.

Availability of data describing the current traffic conditions is a key requirement, as it will be extensively used by various services provided by the system, including individual route planning (for normal traffic participants, but also privileged users as police, ambulances, fire brigades), urban-wide traffic optimization, current traffic information, traffic monitoring by external entities (e.g. police services, road administrators).

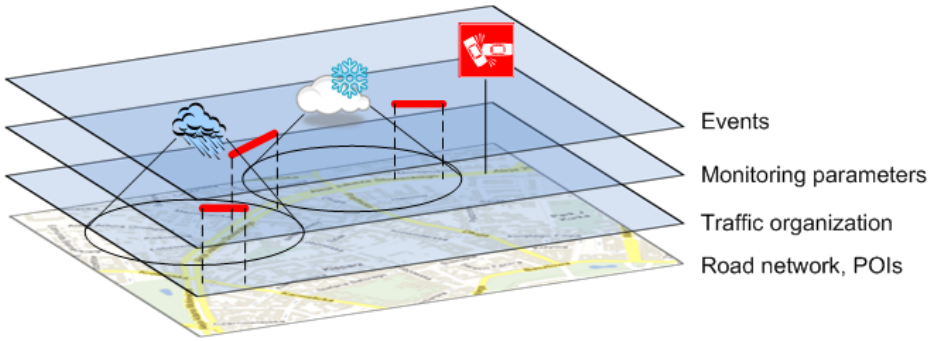

Fig. 1: The logical layers of the Dynamic Map.

The repository stores traffic information that other subsystems, e.g. route planning and traffic optimization subsystems, should be aware of. The data includes measured traffic parameters, weather conditions and events (traffic jams, accidents, etc.). The data can substantially differ in datatypes, ranges and units. Moreover, it must be attached to the underlying road network structure: roads, lanes, crossroads or areas. This implies the need of semantic support which allows to differentiate various types of parameters and to verify whether the data feeds are correct with respect to assumed restrictions. System maintainability is also an important issue: while integrating new types of sensors and measured parameters, the repository should smoothly incorporate the changes without affecting the internal structure. This entails the need for discoverability: a client should be capable of querying about types of measured parameters and their values for an area of interest, i.e. to perform a route planning task.

There are several problems that needs to be faced. A static, topological information data source is not uniform. It is compiled from the road structure

and additional data defining junctions, lanes, special areas etc. The compilation process needs to be precisely defined and has to take map topology changes into account.

The dynamically-changing information needs to be semantically connected both to the static information as well as concepts describing its nature, i.e. what is a sensor, its type, capabilities, what kind of data it feeds in, where it is located, what is the meaning of transmitted data, etc. This semantic connection is provided by the ontology. Such dynamic data is subject to further interpretation, generating even more semantically annotated information, i.e. traffic jam or accident detection.

Another important requirement relates to the performance. The repository should accept an assumed number of data feeds and client requests. Especially, frequent updates need to be taken into account. This leads to the use of relational or NoSQL databases, in spite of their lack of built-in semantic support, as their performance is superior to storage solutions based on RDF triple models, especially for data entry.

An issue that should also be addressed is data reliability. All data originating from sensors is assigned a timestamp and a validity interval. Sensor activities and their data feeds are monitored to detect failures.

The last, but the least, problem concerns interoperability. Web services have been chosen as the primary access method to the repository, as they enable integration with client programs implemented in different programming languages and being executed on various platforms.

2 Map storage system

The proposed Map Storage System consists of two components: the static map and the dynamic map. The system is designed to fulfill requirements of the INSIGMA Project.

The static map provides detailed data regarding the road network and infrastructure. It serves as the basis for visualisation, route planning, and as reference for dynamic data source location. The base layer of the static map is street network data, imported from the OpenStreetMap[1] (OSM) project. As the OSM model, described in detail in Section 3.2, lacks certain elements (e.g. lanes, roadsigns, maneuvers) required in the INSIGMA Project, it has been supplemented by an additional layer. An ontology for the static map, based on map features extracted from the OSM model, has been described in Section 3.1.

The dynamic map holds dynamically-changing data, such as traffic parameters and events. To facilitate future extensions, it has also been based on an ontology (see Section 3.1). The current implementation uses a relational database (PostgreSQL).

Individual components have been described in respective sections. Interactions among them are showed in Fig. 2.

[1] `http://www.openstreetmap.org`

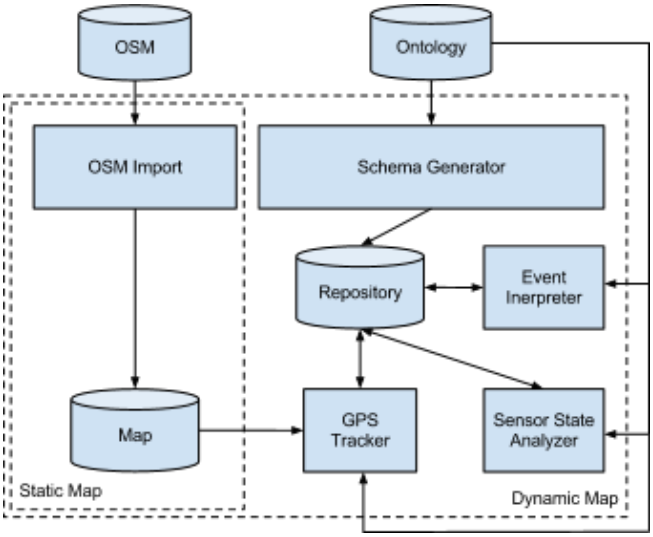

Fig. 2: Interactions among presented components.

3 Data models

This section describes the data models used for various parts of the Dynamic Map system.

3.1 Ontologies

According to a well known classification presented in [3] ontologies can be used in two phases of the software lifecycle: development and exploitation.

Ontologies finding application during the system development phase fall into two categories:

- ontologies formalizing its domain, i.e. classifying physical objects, abstract concepts and events; these ontologies apart from providing a shared reference model can be used for automatic or semi-automatic generation of database schemas or defining data exchanged between components;
- task ontologies describing software functions, services and interfaces that can be used during the system integration.

During the system exploitation ontologies can be used when the domain model or behavior description can not be fully analyzed and specified during the system development; ontologies allow to extend the basic domain model by introducing new concepts, that were not identified during the system development, classification of information is required, and the complexity of the tasks is a prerequisite for the use of advanced semantic tools (reasoners), or some kind of reasoning supporting decision making should be applied.

The system build within the INSIGMA project is at present at the development phase. Several components have been implemented (but still not integrated). The ontologies build during their development constitute a formalized domain model encoded in OWL language. We find that such representation is superior to UML class diagrams, due to the model size and complexity. The model comprises nearly 1 000 classes specifying various objects appearing on maps, types of roads, elements of the road infrastructure, monitoring parameters, sensor, events, traffic organization, types of vehicles and user preferences. These classes are arranged into modules:

1. Ontologies of the static map (`osm-core.owl`, `osm.owl`, `static-map.owl`)
2. Monitoring parameters ontology defining dynamic traffic properties delivered by sensors (`param.owl` and `sensor.owl`)
3. Ontology of events and threats (`event.owl`)

Based on these formal domain specifications the database schemas for three data repositories were semi-automatically generated: Static Map, Monitoring Parameter and Events. The repositories are implemented as PostgreSQL relational databases; the choice of relational representation was driven by the performance issues. The rules of ontology TBox translation are discussed in detail in the section @xxx. An important factor related to the translation, is that it should be revertible. Assuming that the records in the database represent individuals in ontology (ABox), the database schema should provide additional semantic information that can be used to reproduce information on individuals and their relations in form of RDF graph that is required as an input for reasoners.

3.2 Static map

The static map is based on the OpenStreetMap (OSM) map model, which consists of the following elements:

- **nodes:** single points with geographic coordinates and unique IDs,
- **ways:** sequences of node elements, used to represent roads as well as other features, such as building shapes,
- **relations:** sets of way and/or node elements which represent a given entity (e.g. a tram line or a complex junction).

The semantics of each OSM element is defined by attached tags; for instance, every way element representing a road has a *highway* tag with a value determining the road type (motorway, local road, etc.) [1].

The OSM layer needs to be extended to fulfill the needs of the INSIGMA project. Therefore, the OSM layer has been supplemented with an extension layer, introducing the following elements: the *Crossroads* class, which defines crossroads by specifying all of its entry and exit roads; the *Turn* class, which can be used to model maneuver restrictions; the *Lane* class, used to represent individual lanes; the *SMNode* class, which allows for definition of "nodes" which do not exist in OSM and their relation to existing OSM elements.

The components are defined in the static map ontology. Part of the database schema used to store data imported from OSM is based on the OSM Sinple Schema, as described by the `pgsimple_schema_0.6.sql` file, which is a component of the Osmosis import tool (see Section 4.1). Additional components have been modeled using the static map ontology.

3.3 Dynamic map

The Dynamic Map consists of the Repository, which is a database core component, and software modules supporting it. The Repository database schema is synthesized by the Schema Generator. The main goal of the Repository is to efficiently store and retrieve facts for given ontological classes. Other modules use the Repository, retrieving, interpreting, processing or generating new facts. They can also optionally access the Ontology.

Since there are certain performance requirements the Repository schema is synthesized from OWL in a two stage process. The first stage regards flattening desired class structure, the second stage generates actual relations for the flattened structure.

The flattening reduces number of records needed to represent single ontological fact. Other approaches such as [4], which store information about each class as separate relation require multiple records for storing a single fact. Assuming that a single fact is a set of property values of given class to be stored, each property value is stored at different relation plus some additional records need to be generated if there is a class hierarchy involved (in case of an *is-a* relationship between classes). The proposed flattening represents a tree of classes interconnected with is-a relationships as a single class. Additionally in order to preserve information about sub- and superclasses some metadata has to be stored along with the new class instances as well, so-called instance metadata.

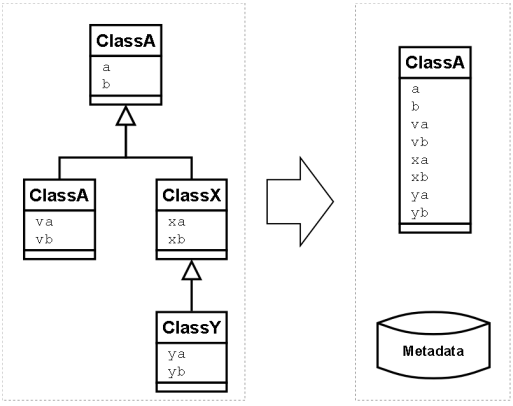

Fig. 3: The First Stage: Class Flattening.

An example of the flattening operation is presented in Fig. 3. On the left side there is a class hierarchy which is transformed into a single class on the right side. If there are any instances of the new class the Metadata contains information that translates them back to proper original classes.

The second stage, translating classes from the first stage into relations is given in Fig. 4. For each flattened class a relation is created. Such a relation has a primary key desined as *id*. Depending on particular property's arity and the class, the property belongs to, different actions are taken. In case of property *a*, it corresponds to an attribute (*table_classA.a*) since it regards a base datatype class of maximal arity of one. The property *c* is translated into the attribute *table_classA.c* which is a foreign key to *table_classB*, since *c* is of maximal arity of one and it is of *classB*. The properties *b* and *d* are represented with use of the auxiliary relations *table_classA_table_classB* and *table_classA_integer* respectively, since they are of any arity.

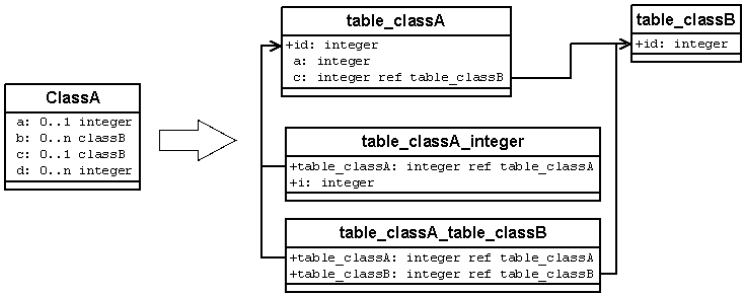

Fig. 4: The Second Stage: Relational Schemas (+ stands for Primary Key).

Additionally appropriate metadata allowing to recreate class instances from the relational model is also provided, so-called structure metadata. It is given in Table 1 as *meta_param* relation. Its attributes are interpreted as: *cls* – class name, *prop* – property name, *tbl* – relation name, *att* – attribute name, *ref_tbl* – name of the relation which holds property instance (optional, for properties of classes other than basic types), *key_att* – foreign key attribute name, (optional, for arities greater than 1), *ptr_tbl* – name of the relation that implements a foreign-primary key relationship (optional, for arities greater than 1).

The instance metadata is implemented as a three-attribute relation: *meta_param_pk_tbl_cls(pk, tbl, cls)*. It allows to map any record (identified by *pk*), in any relation (identified by its name: *tbl*), to be assigned to appropriate class (identified by its name: *cls*).

The proposed procedure if fully reversible. Database records can be extracted and presented as ontological facts by using the metadata. In order to find a correspondence between a record and an ontological class the following procedure has to be considered: (1) Find appropriate correspondence between the class and the relations using the structure metadata: the *meta_param* relation, to

Table 1: Structure Metadata Relation: meta_param

cls	prop	tbl	att	ref_tbl	key_att	ptr_tbl
classA	a	table_classA	a			
classA	c	table_classA	c	table_classB		
classA	b	table_classA_table_classB	table_classB	table_classB	table_classA	table_classA
classA	d	table_classA_i	nteger		table_classA	table_classA

issue a database query. (2) Find apropriate records in the relations. (3) Identify which classes or subclasses the records belong to, using the instance metadata: *meta_param_pk_tbl_cls*.

The proposed database schema and the structure metadata relation depend on the ontological classes and their relationships. The first stage metadata relation depends on class instances stored in the database. Storing a single instance of a given class requires one record for all base type class properties, one record of instance metadata and one for each property with arity greater than one. Comparing with other approaches mentioned before it gives reduction of the number of records. The number of records needed to represent an instance of a class with multiple properties is reduced proportionally to the number of properties.

The proposed data model allows efficiently store and retrieve ontological instances. They can be either accessed as plain database records or, with use of the metadata, as full ontological knowledge regarding both classes and instances. The process is fully reversible, efficient generation of appropriate RDF or OWL is possible.

4 Description of selected modules

This section describes selected modules of the proposed dynamic map system.

4.1 OSM Import

OpenStreetMap has been selected as the data source; therefore, a system had to be implemented for efficient import of selected map fragments. The import procedure has been presented in Fig. 5.

OSM makes its data available in the form of one huge file, `planet.osm`. Since downloading and processing of this file is unfeasible, for the time being a decision has been made to make use of ready-cut data files available at the GeoFabrik website. The import procedure is as follows:

1. Cut planet.osm to extract the map of Poland (this step is performed by GeoFabrik).
2. Download the `poland.osm` file from GeoFabrik.

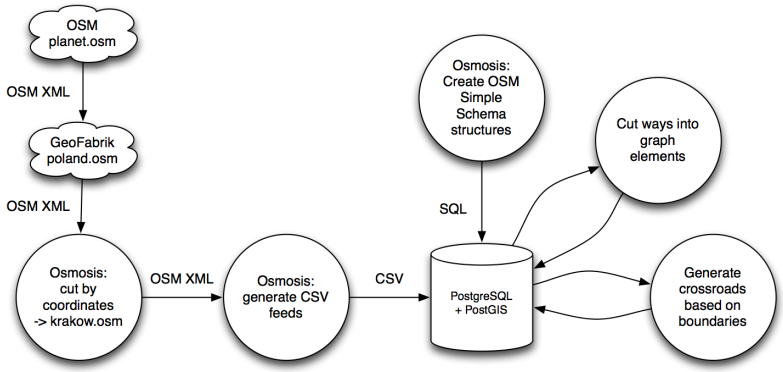

Fig. 5: Overview of the OSM import procedure.

3. Cut the `poland.osm` to extract the desired region (e.g. Kraków) based on coordinates; this step is performed using the Osmosis[2] tool.
4. Generate CSV files with feeds, also using Osmosis.
5. Implement the OSM Simple Schema in the database, using Osmosis accompanying SQL files.
6. Import CSV files generated in step 4 into tables.
7. Cut way elements into individual graph edges.
8. Generate crossroad data using predefined crossroad boundaries.

Boundaries of each crossroads is stored as a closed *linestring*, thus defining an area. All processing is performed using PostGIS; each node which lies within the area of a crossroads and takes part in a way which intersects with the crossroads boundary is labeled as an *input*, *output* or *input/output* node, depending on the direction of the way. All nodes which lie within the crossroads area and have not been labeled otherwise are labeled as *internal* nodes.

4.2 Schema generator and repository

The Generator selects particular ontological classes and provides appropriate database structures to support storing and retrieving their facts. Other modules use the schema, the facts, and optionally the Ontology to perform their actions.

Initially the Repository schemas were generated for the monitoring parameters. Currently they are being extended, to include other concepts defined by the Ontology, such as events. This extension can be easily performed by fine-tuning the Generator.

The Generator acts according to the two-stage process presented in Section 3.3. It analyzes the ontology, expressed as OWL, and generates SQL queries which, in turn, create appropriate tables. It also generates the structure metadata, providing appropriate table schemas and filling it with records. The instance metadata table is created, however no records are generated. It should be

[2] `http://wiki.openstreetmap.org/wiki/Osmosis`

populated by other modules while storing facts in the database. The generated SQL queries comply with the SQL2 standard.

Since the process presented in Section 3.3 is formally defined with rules, the module is implemented in Prolog language. Expressing rules in Prolog is straight forward. To access and process the ontology Thea2 OWL library[3] is used.

4.3 GPS Tracker

The GPS Tracker module collects information on positions of multiple vehicles registered in the system as data sources. Periodically entered GPS coordinates with accompanying timestamps are combined into vehicle trajectories, which are the basis for calculating values of several monitoring parameters related to lanes and turns.

The analysis of GPS coordinates is is a two-stage procedure. The goal of the first stage is to estimate state variables of a vehicle: location, current velocity and acceleration. The estimation is based on the position observations and uses the Kalman filtration method; In the second stage, a trajectory (a vector of state variables and their occurrence times) is projected onto the network of routes defined in the static map, then several values of monitoring parameters are calculated, assigned to lanes and turns and finally send to the repository as values of volatile parameter instances.

As we mentioned earlier, lanes and turns constitute an intermediary layer serving as a bridge between OSM data structures specifying physical course of roads and, stored in the separate repository, dynamically changing information about current traffic and traffic related events. We we adopted lazy evaluation approach, when reporting values of monitoring parameters based on the interpretation of vehicle trajectories. We start with an initial small set of lanes and turns (manually introduced during configuration of fixed sensors: cameras, inductive loops) and, if needed, dynamically create new ones when a vehicle passes successive streets and intersections.

The correctness of trajectory interpretation pose several challenges. First, it is difficult to distinguish between intentional vehicle stopping and stopping it due to a jam or other traffic obstacle. To cope with this problem, we use additional semantic information about the user profile (e.g. municipal transport bus) and the context information (e.g. location of bus stops and terminals). Second, It is assumed that information about location of lanes is unsure (in fact in OSM only numbers of lanes happens to be specified). In consequence, monitored parameters (e.g. average speed) are attributed to all lanes (with a lower confidence if several lanes exist). In some cases, the proper lane can be identified based on user profile (privileged users as buses, emergency vehicles are expected to use a separated lane). Finally, there is an issue concerning assumptions about what is known about future user behavior. If trajectory interpretation is limited to observations

[3] `http://www.semanticweb.gr/thea/`

only, it is possible to calculate monitoring parameters based on past events. This introduces delays, which should be avoided. [4]

4.4 Sensor state analyzer

Each value of a monitoring parameter entered to the repository is represented by a record specifying the time it was measured and the validity period. In normal operation mode it is expected, that before this time elapses, a new value will replace the old one. If a new value does not appear within the validity period, the old value can still be used, however its utility gradually diminishes until the moment, when it should be ignored by route planning algorithms as apparently outdated. From the perspective of application in route planning, it is important that such outdated values and inactive parameter instances are hidden and not visible for discovery services. This regards route planning components deployed at the server side, as well as those running in mobile devices and dynamically downloading traffic data from GSM network. Moreover, long-term absence of updated traffic data may be caused by a sensor failure and should trigger servicing and maintenance actions.

The goal of the sensor state analyzer is on-line detection of long-term inactivity observed at the input interface with respect to a monitoring parameter instance. If such inactivity is detected, the state of the instance is changed to exclude it from the discovery mechanisms. If all monitoring parameter instances that are linked to the same sensor are inactive, the analyzer module changes the sensor state and sends the notification to the human operator, who decides if the inactivity represents a system or hardware failure or is caused by a natural condition, e.g. a snow fall or a heavy rain.

4.5 Event interpreter

The goal of the event interpreter is to identify event occurrences by analyzing the changes in the monitoring parameter values and their trends. Internally, it implements the *Observer* design pattern [2]. Several observers, e.g. TrafficJamObserver, HavyRainObserver, SnowFallObserver aimed at detecting particular event types are registered as recipients of notifications about changing values of relevant monitoring parameters. When a notification arrives, the observer makes a decision, whether a new event should be created and stored it in the event repository or properties of an existing event should be modified (changing state to terminated, duration prolonged, location extended to a wider area etc.).

[4] To give an example, in the presence of a traffic jam, the time required to travel a road section and take a left turn can be calculated as equal to 30 min, after the turn is eventually made. Considering the usability of the obtained information for the route planning task, it can be stated that the information is outdated by 30 min. During this period several hundreds of vehicles could have been directed to the jammed area. However, if it is known that a user intends to make a maneuver consisting in turning left at the end of the road, the same parameter value can be estimated about 30 minutes earlier from registered speed and displacement.

To avoid immense flow of insignificant notifications, an observer during the registration indicates conditions (rules) specifying when it should be notified about the changing parameter value. The conditions are defined with use of thresholds or ranges and can be applied directly to parameter values, differences between old and new values, change directions and their speed.

5 Conclusions and Future Work

This paper introduces an outline and interactions among components of the proposed Storage System. The system handles georeferenced static data as well as dynamically changing monitoring parameters. The parameters are subject to interpretation to detect more abstract concepts, such as traffic jams, accidents etc. The proposed architecture is highly modularized. Each module regards particular functionality. There are five modules introduced. They are responsible for populating and synchronizing static data (OSM Import), generating appropriate database schemas complying with given Ontology (Schema Generator). Furthermore there are auxiliary modules interpreting raw data such as: detecting sensor conditions (Sensor State Analyzer), detecting events (Event Interpreter), or assigning monitoring parameters based on mobile sensors (GPS Tracker).

The presented solution provides a complete and uniform data source describing a dynamically changing map. The source is ontology driven. All data, while stored in a relational database, is semantically tagged. It enables RDF or OWL synthesis of gathered facts at any time.

Future work regards tweaking the Schema Generator to support broader set of ontological classes. Several other modules, interpreting gathered data are also considered. Furthermore, implementation of full RDF and OWL export capabilities, as a separate module, is also taken into account.

References

1. OpenStreetMap wiki: Map features. `http://wiki.openstreetmap.org/wiki/Map\_Features` (Retrieved Jan 2012)
2. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1995)
3. Guarino, N.: Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy. IOS Press, Amsterdam, The Netherlands, The Netherlands, 1st edn. (1998)
4. Lependu, P., Dou, D., Frishkoff, G.A., Rong, J.: Ontology database: A new method for semantic modeling and an application to brainwave data. In: Proceedings of the 20th international conference on Scientific and Statistical Database Management. pp. 313–330. SSDBM '08, Springer-Verlag, Berlin, Heidelberg (2008), `http://dx.doi.org/10.1007/978-3-540-69497-7_21`