Ontology-Driven Wrappers for Navigation Services

Dirk Redbrake and Martin Raubal
Institute for Geoinformatics
University of Münster
{redbrak | raubal}@uni-muenster.de

SUMMARY

The growing number of information services and data sources requires the development of standards for interoperability. Ontologies are one possibility to describe domains and their properties for interoperable use. Those ontologies that are conformant to the standards can be used as underlying models for different GIS and data sources that deal with the same domain. Otherwise, non-interoperability between systems is a likely consequence. Wrappers solve this problem. They aim at transforming information from one domain into another without semantic loss or errors. In this paper, the wrapper-based navigation services 'route calculation' and 'route guidance' are developed as part of an Ontology-Driven Geographic Information System. External ontologies define the transformation results. Solving this task requires a knowledge base consisting of ontological axioms and their translations. If semantic translation is impossible then error codes are returned as the translation result for the axiom.

KEYWORDS: *ontology, wrapper, Geographic Data Files, TeleAtlas MultiNet™, OpenGIS® Simple Features Specification, DARPA Agent Markup Language, navigation service*

INTRODUCTION

Throughout the past years, the growing abilities of computers and the Internet caused the problem of non-interoperability among information services and data sources. If interoperability between services and data sources is successful, then information transmission works in both ways without creating semantic loss or errors.

In this paper, semantic interoperability between the wrapper-based navigation services *route calculation* and *route guidance* and a geographic data source is analyzed. The motivation for this choice is to investigate how one data source can deliver different semantically correct information for various travelling modes and user groups, such as hikers and car drivers, by redefining the semantics of the data. An example for a semantic misunderstanding was the likely cause for a car accident that happened after a navigation service gave the driver a wrong instruction. This driver drove his car into a river after being instructed by his electronic navigation service to continue driving over a supposed bridge, which was a ferry connection in reality. A detailed description of this case can be found in (Kuhn & Raubal 2003) and (Raubal & Kuhn 2004).

One approach for achieving semantic interoperability is the use of ontologies. If services or data sources communicate with the help of ontologies or commit to the same ontology then semantic interoperability becomes possible. In GIScience, the ISO Geographic Data Files (GDF) and the OpenGIS® Abstract Specification (OAS) are two standards for interoperability between GIS and data sources. Ontologies that are conformant to such standards can be used as models for different GIS and data sources that deal with the same domain to increase their semantic interoperability. In this paper, the data source for the navigation services is GDF conformant while the services are OAS conformant.

Wrappers are one way to aim for communication between services and data sources without semantic loss or errors. If services are implemented as a software system that commits to ontologies then the term Ontology-Driven Geographic Information System (ODGIS) is used (Fonseca et al. 2002). A wrapper that is ruled by external ontologies will be developed in this paper. It requires a knowledge base consisting of

ontological axioms and their translations. The knowledge base includes semantic translations of each axiom that allow a well defined conversion from the data source into the properties of an implemented navigation service, which commits to an external ontology.

Our main contribution is the practical development of a wrapper-based ODGIS that is designed for maximum reusability with the help of external ontologies. Other research investigated about general concepts how to develop a wrapper (Lu and Mylopoulos 2002; Thiran 1999; Zhang et al. 2000; Zaslavsky et al. 2000). This paper focuses on the design and implementation of one pragmatic example. Two standards are taken for the wrapping process, therefore wrapper reusability is important. This is achieved through external ontologies.

REQUIREMENTS FOR THE NAVIGATION SERVICES

A navigation service needs to calculate a route from a start to an end point. It depends on the service what kind of preferred path, e.g., shortest, most scenic (Golledge 1995), can be calculated. This paper deals with shortest paths. Guiding hikers and car drivers along these routes is done by the service *route guidance*.

Route calculation

Algorithms for calculating shortest paths are based on graphs consisting of nodes and valued edges. To avoid accidents like the one described, these basic geometries require additional attributes that define restrictions. A geometry with additional attributes is called a feature. In this paper, only line features are considered. Traffic networks and shortest routes are modelled as feature collections. In other words, routes are lists of line features. Every edge in a line feature includes two points. Traffic networks are split into layers. Here, a route can only be calculated in a layer that supports roads and ferries. The operation that a client sends to a server has the following signature:

```
RouteCalculation :: LayerId -> StartPoint -> EndPoint
                    -> FeatureCollection
```

Route guidance

This service receives the feature collection from *route calculation* as input and analyzes each edge that belongs to a feature. Every feature has a comment attribute where error codes and warnings were previously saved by the wrapper. The service *route guidance* guides the user feature by feature. Attribute values and error codes warn the user or give well defined orders how to behave. Street names and lengths of feature geometries are also added. Its operation has the following signature:

```
RouteGuidance :: FeatureCollection -> ListOfInstructions
```

ONTOLOGIES

An ontology is an explicit specification of a conceptualisation (Gruber 1993). It describes concepts and relationships for the benefit of knowledge sharing and reuse. The practical use lies in commitments among members of communities to particular ontologies. Sharing the same ontologies makes them use equal vocabularies and definitions. If services and data sources commit to shared ontologies then semantic interoperability is possible. In this paper, ontologies for GDF and the OpenGIS® Simple Features Specification (SFS) are developed. The ontologies are described in the DARPA Agent Markup Language (DAML) (DARPA 2003).

GDF Ontology

The GDF standard was developed to describe as many geographic objects and relations between them as possible (ISO 2001). This paper deals with navigation services, therefore many details of the standard not necessary for routes are ignored. We consider *Roads* and *Ferries*. GDF offers three levels of representation. On Level-0, the basic graphical building blocks are defined. They are nodes, edges and faces, or alternatively, dots, polylines and polygons. Level-1 represents simple features such as *Road*

Element, *Junction* and *Ferry Connection*. A *Junction* is a feature that bounds a *Road Element* or *Ferry Connection*. On Level-2, complex features are represented. We define elements from Level-0 and Level-1 in a DAML ontology.

SFS Ontology

Software engineers are enabled by OGC's Abstract Specification to create and document conceptual models that are sufficient to create Implementation Specifications like the OpenGIS® Simple Features Specification (OGC 1999a,b). The specification of geometries is based on the ISO 19107 Spatial Schema (OGC 2001). Compared to GDF, OAS offers more possibilities to increase interoperability concerning modelling geometries. To define SFS geometries in an ontology, parts of the Geometry Class Hierarchy of the Geometry Object Model (OGC 1999a) are used. The final selection of metadata entities and elements to associate with each Feature and Feature Collection is left to each data producer or geospatial information community. This means that each software engineer is allowed to define individual feature attributes. Our ontology-based wrapper needs a SFS ontology, but if there exist individual feature definitions for each client then it is also necessary to define a feature ontology for each client.

WRAPPER

The task of a wrapper is to translate a request fragment from the mediator's language to that of the information source and transform the results provided by the information source back to the mediator's language. In this way, wrapping each information source into the translation software makes the protocol diversity of particular sources manageable (Zaslavsky et al. 2000). A mediator is designed to visit the distributed data servers automatically and gathers the metadata. These data are stored in the mediator's Meta database (Zhang et al. 2000).

Wrapping a system is the process of defining and restricting access to the system through an abstract interface. A wrapper for information services accepts queries in a given format, converts them into one or more commands or sub-queries understandable by the underlying information service and transforms the native results into a format understood by the application (Lu and Mylopoulos 2002). Constructing wrappers is a diverse process because the input/output descriptions depend on the level of abstraction and on the details of the descriptions.

ODGIS ENGINEERING

The need for a wrapper occurs when a service intends to use a new data source that commits to a different standard. Figure 1 demonstrates such a situation.

System design

The perfect way to wrap information as presented in Figure 1 would be that the source codes of the old services need to be only minimally altered. The geometry classes used here are specified by SFS and therefore, the interfaces, attributes and algorithms are clearly defined. Although different clients may use different implementations of these algorithms, the interfaces and operation results have to be equal. Therefore, the geometry classes of the different clients are treated equally because the individual implementations are not important for the wrapper itself. Feature collections and their features are not specified by SFS and therefore, the wrapper must cope with differing features. The wrapper must also deal with features in a most general way because a large number of applications may use the same wrapper for the same data source. This is one more important reason why the use of ontologies is a good solution. Defining the details for GDF, the Geometry Classes and the individual features as ontologies turns the wrapper into a general information translator. Such translator can handle an increasing set of information axioms, such as new GDF features, SFS classes and feature collections of new clients through an extension of the ontologies. The wrapper may only require the implementation of new DAML axioms that are needed for new information structures in the extended ontologies.

The data sources may differ in their location, file format, etc. In this paper, the data source is the text file Havellnd.gdf of the record-based Tele Atlas MultiNet™ database. It is already modelled as GDF data. Data fields of the single records are specified in Tele Atlas NV (2001). Only those data relevant to the navigation services are analyzed. Warnings and error codes must also be defined in the knowledge base for cases of difficult or impossible wrapping. Finally, the knowledge base requires translation rules for all individual features.

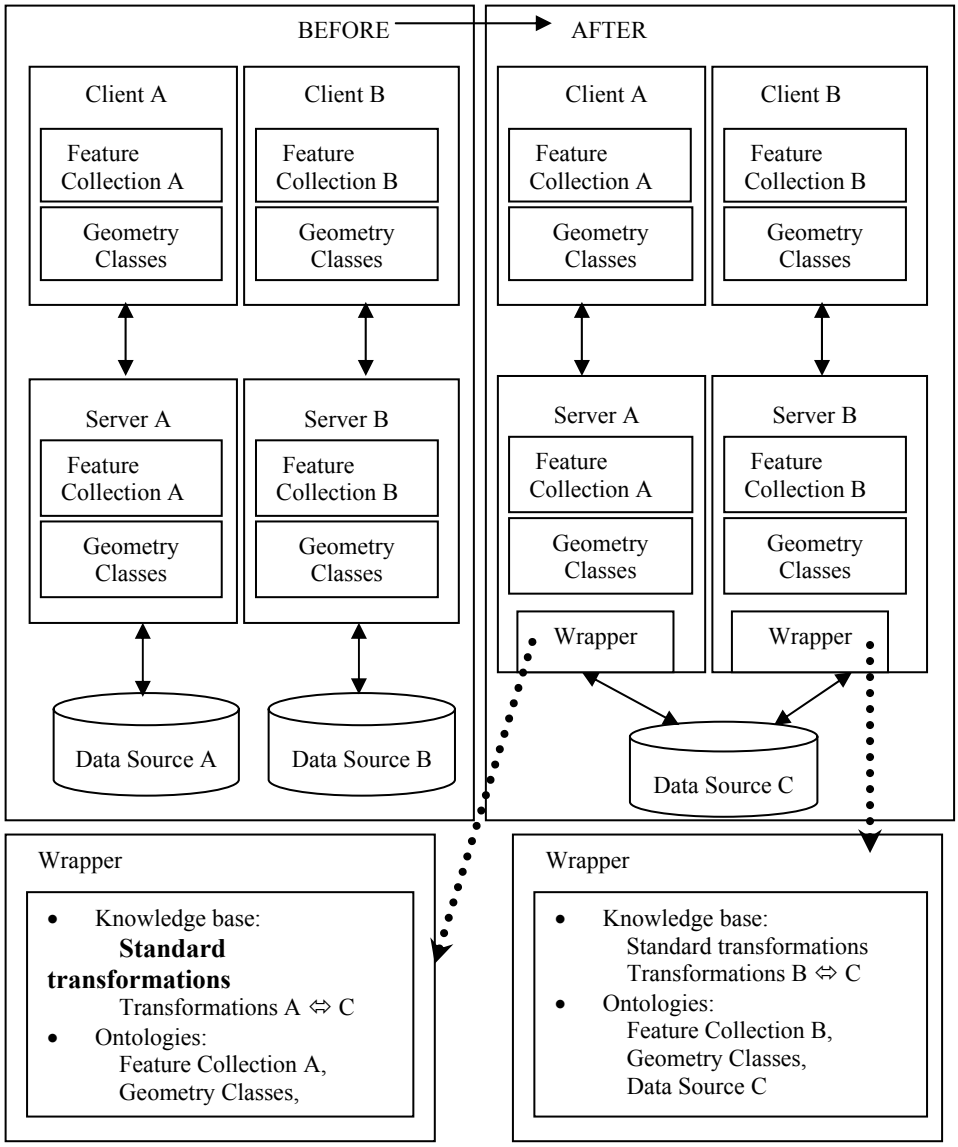

Figure 1: System architecture before and after introducing a wrapper.

Tasks of the wrapper

The wrapper developed here executes several tasks that are independent from each other. Text files are first imported. The component OntologyFactory imports the ontology files and parses the DAML code. Here, an ontology comprises two sets of classes and properties. Points and line features are classes while identifiers, coordinates and feature attributes are properties. A class is a set of property restrictions. Each referenced property is set to a fixed single value or restricted by a cardinality. Using these classes, the OntologyFactory creates ontology classes that include equal information like it is defined by the DAML ontology. The wrapper receives these two implemented ontologies for the external GDF and SFS ontologies. After importing the knowledge base, the data source is read by ignoring those data records that do not fit to the requested themes *Roads* and *Ferries*. After that, the wrapper tries to translate the data. There are three types of rules available for this wrapper. The easiest type are single data fields that work for most feature attributes because their translation is clearly defined in a single line feature record. Impossible translation leads to a well-defined warning or error message that the internal feature attribute *comment* stores. As second type, finding route segment names requires moving through attribute records and finally text records to find the real name. This requires a more complex rule description and wrapper access implementation. Third, wrapping the geometry turns out to be most difficult because in addition to navigating through data records, classes also have to be created while continuing to move through the data records. Finally, the wrapper creates a layer with the wrapped data.

Rules for a knowledge base

Developing a knowledge base that fits the requirements of an ODGIS is very difficult. Semantic translation does not only require a data model. There must be definitions which objects and properties from one domain equal classes and properties of another domain. Furthermore, algorithms how to access and translate information are necessary. The more complex a request is, the more difficult this task is. The main question here is whether a wrapper-based ODGIS can be developed using a wrapper that has no detailed information about GDF and SFS at all in its source code. Any action must be indicated by external ontologies and external knowledge rules. After testing different approaches, it was necessary to define at least general rules that are considered to be reusable for other applications. This means rules are specified and at least the names of such rules and their basic notation must be known by the wrapper source code directly. Therefore, rules have the following structure:

```
:Rule=name
[sequence of axioms]
:Rule end
```

Every *name* of a rule is static and not allowed to be changed like the earlier described *Single Data Fields*. Knowledge rules must deliver enough information about where a wrapper can find data and how the data has to be used to receive the correct semantic translation.

The wrapping process

The wrapping process is described with the help of attributes that occur again in the use cases of the next section. Unique identifiers for features are being wrapped from the MultiNet™ file into attributes of RouteSegment features. In the knowledge base, the rule *Single Data Fields* includes the axiom:

```
52:3:12#GDF:LIFEREC:LINE_ID#DrivingFeatureCollection:RouteSegment:
featureId
```

It is the easiest example for wrapping an attribute. The # signs subdivide the rule into three parts. First, it is defined where a GDF attribute is situated in the text file. A line feature has the record description code 52. These codes always appear as the first two characters of a record formatted as a string. In the MultiNet™ file, the data field of the line feature identifier begins at character 3 and ends at character 12. After extracting these 10 characters, part two is started. It describes the ontological axiom of the GDF ontology. The GDF name for the conformant record is LIFEREC and the suitable attribute name is

LINE_ID. Now the wrapper can locate the attribute LINE_ID through the DAML class LIFEREC in the GDF ontology. It is defined as:

```
<daml:DatatypeProperty rdf:ID="LINE_ID">
 <rdf:type rdf:resource="[…]/daml+oil#UniqueProperty"/>
 <rdfs:range rdf:resource="[…]/XMLSchema#positiveInteger"/>
</daml:DatatypeProperty>
```

This simplified notation reveals the requirements for the identifier LINE_ID, which must be unique and have a value of at least 1. After converting the 10 characters into an integer, the restrictions must be evaluated. If the identifier already exists then the wrapper stops dealing with this record. If the type is incorrect then it must be decided whether the entire wrapping process stops or this record can be ignored. Here, a record is always simply ignored when an invalid identifier occurs. Finally, the last part of the wrapping process begins, namely the attribute is inserted into a SFS or feature class. In the ontology DrivingFeatureCollection, the DAML class RouteSegment is accessed by the wrapper to find the definition of featureId. It has the same structure as LINE_ID. Again, the restrictions are evaluated and after a successful test, the attribute value is taken to create an instance of the class RouteSegment with this value as featureId. The other attributes of the class are also translated. The next section explains two examples.

TEST RESULTS

The wrapper-based ODGIS from Figure 1 is implemented and tested with Java™ 2 SDK. After the server is launched, the wrapper imports the ontologies and knowledge base rules from text files and starts importing the data records from the MultiNet™ text file one by one. First, the nodes and edges for the geometry classes are imported and wrapped into Java geometry classes. Every client uses these geometry classes. Depending on the individually defined features, Line Feature Records and Attribute Records from GDF are wrapped into feature attributes. As a demonstration of the test results for wrapping individual features, two use cases, features for hiking and car navigation systems, are presented. A route made of line features is defined as a HikingFeatureCollection or DrivingFeatureCollection consisting of RouteSegments. One simplified possibility to describe the attributes of segment classes for both feature collections in Java notation is:

```
// Hiking feature(data type; attribute name; attribute description)
 double featureId;  // Identifier
 string comment; // Warnings and errors
 double maxSlope; // Maximum slope allowed in degree
 double segmentLength; // Length of line string
 double segmentName;  // Name of line string
 double zCoordStartPoint; // z-coordinate for start point
 double zCoordEndPoint; // z-coordinate for end point
 LineString geometry;  // SFS geometry for this feature

// Driving feature(data type; attribute name; attribute description)
 double featureId;  // Identifier
 string comment; // Warnings and errors
 int segmentType; // Segment type: 0=Road; 1=Ferry
 double segmentLength; // Length of line string
 double segmentName;  // Name of line string
 LineString geometry;  // SFS geometry for this feature
```

LineString is a SFS class that contains two spatial and linearly interpolated points. Wrapping into a RouteSegment for hiking services creates warnings and errors. A list of error and warning codes for difficult or wrong semantic translation results is defined in the knowledge base rule *Comments*. Codes

and their descriptions are always written into the internal feature attribute *comment*. This enables the clients to decide how to deal with warnings and errors. No fixed definition exists whether a translation problem has to be evaluated as a warning or a critical semantic error. The following axioms of the rules *Single Data Fields* and *Comments* define the warnings and errors that occur during wrapping into a hiking RouteSegment:

```
Critical single data fields:
W1#0#HikingFeatureCollection:RouteSegment:maxSlope
E1#0#HikingFeatureCollection:RouteSegment:segmentLength
W2#0#HikingFeatureCollection:RouteSegment:zCoordStartPoint
W3#0#HikingFeatureCollection:RouteSegment:zCoordEndPoint

Warnings:
W1:MaxSlope does not exist in MultiNet GDF 3.0 data.
W2:Z coordinates for start points are not available in MultiNet GDF
 3.0 data.
W3:Z coordinates for end points are not available in MultiNet GDF
 3.0 data.

Error:
E1:Semantic error occurs if SFS-defined method for 2-dimensional
 segment length is performed for 3D-points (z coordinates <> 0).
```

The values for maximum slope allowed help the *route calculation* service to express whether the real slope of a line string is higher than allowed to be part of a suitable hiking route. But this value is neither represented in GDF nor can it be calculated. Slopes require the z-coordinates of points, which are not defined in SFS. Although the z-coordinates exist in GDF, the SFS method get_Length within the geometry classes only accesses x- and y-coordinates. Therefore, get_Length calculates the distance between two points in two dimensions only. This leads to different results compared to the real-world distances if z-coordinate values differ. The MultiNet™ text file does not include values for z-coordinates (although they are specified in GDF) and therefore the results for get_Length are equal by chance, because undefined coordinate values are assigned to 0 by default. When all z-coordinates are 0 as it is the case for the used MultiNet™ file then such data are practically not useful for hiking routes.

The RouteSegment for driving features is an example for a correct semantic translation. A warning is expressed because the segment length does not belong to GDF, but the geometry classes include the method get_Length, which returns the correct length of the segment. The following axioms define the segment type of a route segment as case sensitive *Single Data Fields*:

```
52:18:21#GDF:LIFEREC:FEAT_CODE(4110)#DrivingFeatureCollection:
 RouteSegment:segmentType(0)
52:18:21#GDF:LIFEREC:FEAT_CODE(4130)#DrivingFeatureCollection:
 RouteSegment:segmentType(1)
```

Wrapping the feature code (FEAT_CODE in GDF) into the segment type (segmentType) defines whether a route segment is a road element (code 4110 → code 0) or a ferry connection (code 4130 → code 1). This value transformation is based on simplified DAML axioms taken from the GDF ontology and the DrivingFeatureCollection ontology:

```
<daml:DatatypeProperty rdf:ID="FEAT_CODE">
 <rdfs:comment>
 FEAT_CODE belongs to the GDF record LIFEREC.
 </rdfs:comment>
 <rdfs:range rdf:resource="[…]/XMLSchema#nonNegativeInteger"/>
 <daml:oneOf rdf:parseType="daml:collection">
 <FEAT_CODE rdf:ID=4110/>
```

```
 <FEAT_CODE rdf:ID=4130/>
 </daml:oneOf>
</daml:DatatypeProperty>

<daml:DatatypeProperty rdf:ID="segmentType">
 <rdfs:comment>
 segmentType is an attribute of the feature RouteSegment.
 </rdfs:comment>
 <rdfs:range rdf:resource="[…]/XMLSchema#nonNegativeInteger"/>
 <daml:oneOf rdf:parseType="daml:collection">
 <segmentType rdf:ID=0/>
 <segmentType rdf:ID=1/>
 </daml:oneOf>
</daml:DatatypeProperty>
```

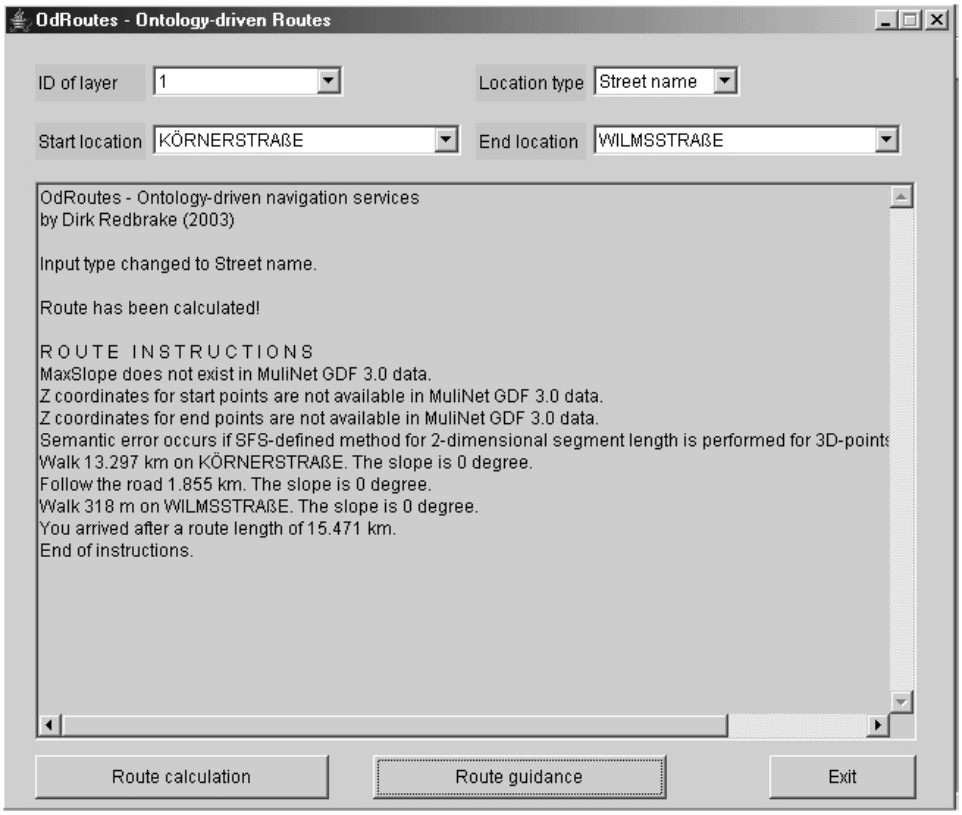

Figure 2: OdRoutes GUI showing instructions based on a hiker ontology.

The implemented ODGIS is named OdRoutes that stands for "Ontology-driven Routes". Figure 2 shows the results after executing both navigation services for the hiker ontology. Locations can be selected as spatial points or street names. The instructions include the warning and error messages as well as calculated values for slopes and lengths of route segments. Referring to the car accident, the hiker ontology could not avoid such an accident if a car driver would follow these hiking instructions because the 1.9-km-long route segment is a ferry connection in reality.

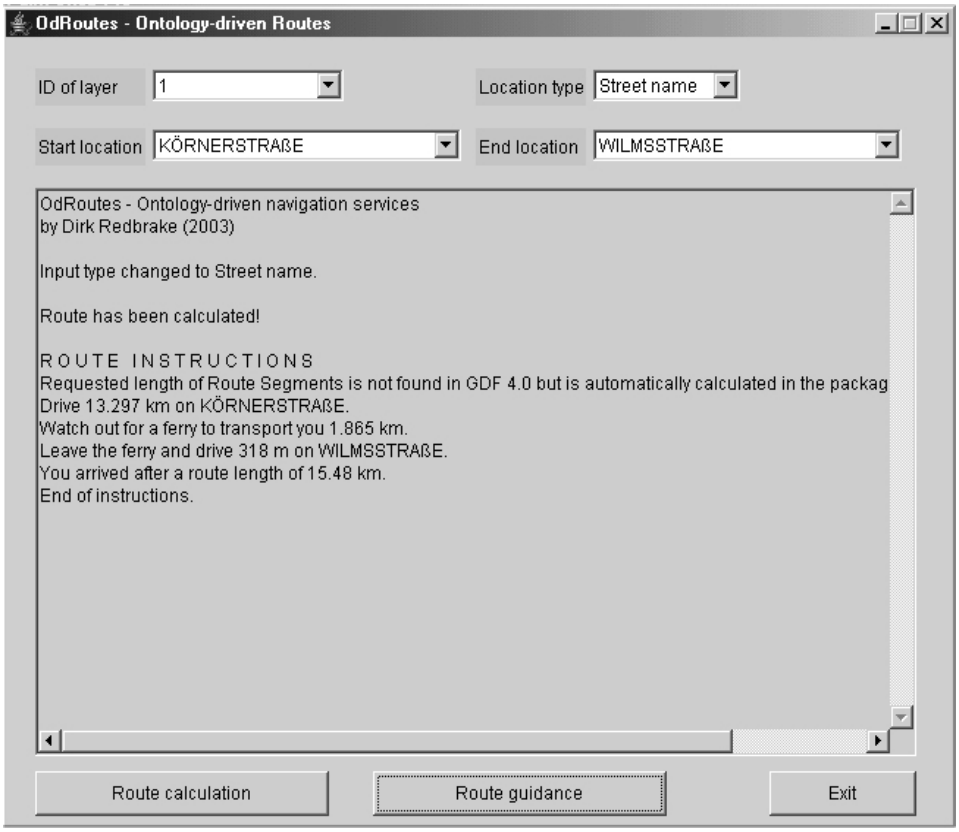

Figure 3: OdRoutes GUI showing instructions based on a driver ontology.

Figure 3 presents the instructions based on the driver ontology. This time, the ferry connection is identified and the driver is correctly instructed.

But there is a big problem in extending this wrapper's abilities. Implementing to find a segment name was already chaotic because it seems that an algorithm is difficult to develop. Therefore, the first approach was to implement rules and Java codes completely iterative. This caused long Java code hoping to discover a strategy how to find the required information. But although many Java command sequences reappear again and again, there is often a small difference so a general algorithm does not seem to exist. When the wrapping process of translating a line string geometry was finished, there was a huge and bad-to-read method with many logical switches. It was a question of time when one little switch would cause a wrong logic and ruin the entire wrapping process. Finally, such mistakes could be found but imagining to wrap complex features appears almost impossible. The dependency between knowledge base rules and basic algorithmic strategies for the wrapper is so strong that rules for complex features must turn into pseudo-code before a correct wrapping is possible.

Solving this implementation problem raised the idea that it was a wrong decision to focus on one pragmatic wrapping example. It appears possible to design and implement the wrapper on a more abstract level. Organising rules in hierarchies may clearly define the algorithms how data have to be connected to form semantically correct information. This could even abstract from spatial domains and enable the wrapper to translate information between two completely different domains that have just a few attributes

in common. Marketing software in a business domain, for example, may require spatial points for a market analysis. An ODGIS may deliver wrapped spatial points then.

CONCLUSIONS AND FUTURE WORK

The ontology-based wrapper presented is a powerful way to cope with different GI-models that are supposed to be connected. The wrapper only needs to collect suitable knowledge and wrap the information automatically because detailed knowledge is "outsourced" into ontologies and a knowledge base. Other standards can easily be added by creating a new ontology. The practical justification for developing an ontology-based wrapper depends on the expense of alternative software designs and implementations with a static wrapper or no wrapper at all. Our ODGIS might be too small for justifying the creation of ontologies, a knowledge base and an entire wrapper, because the expense is much smaller with a few classes that simply access the data source directly. For complex systems though, ontology-based wrappers should practically be a reusable standard software that could be adapted for different applications. The less ontological axioms are standardized such as the features in SFS, the more work it is to create individual ontologies and knowledge base rules for each client. From this point of view, an ontology-based wrapper is an indicator that shows how semantically interoperable a model or standard is. The more individual ontologies and rules must be created, the less semantically interoperable a model is.

For future work, the usability of an ontology-based wrapper could be increased by developing more wrapper intelligence and implementing entire standards as ontologies. The prototype described here reacts to static axioms and understands only a small part of the mentioned standards. It is a problem that the expense for developing and implementing ontology-based wrappers for SFS conformant services theoretically increases endlessly, because attributes for features are not standardized. Furthermore, it must be investigated how the shown problem with features relates to problems with other SFS classes. It is necessary to develop one specification that defines all allowed interfaces, algorithms, attributes and their restrictions. Uniting all advantages of GDF and SFS would result in a reusable and interoperable standard. But the larger the ontologies become, the longer the wrapping process will take. Therefore, the runtime behaviour of such ontology-based wrappers may turn into an important topic. This also includes the choice of development software like the implementation language as there are much faster languages than Java.

Referring to the test results, a new version of this wrapper should be abstracted from spatial contents. A general ontology-driven wrapper is much more reusable by applications from any domain. Such a new version requires redesigning the wrapper and knowledge bases. Knowledge bases may also be defined as DAML ontologies.

ACKNOWLEDGMENTS

We would like to thank Tele Atlas Deutschland GmbH for providing the Multi Net™ data used in this research.

BIBLIOGRAPHY

DARPA (2003) The DARPA Agent Markup Language Homepage. http://www.daml.org.

Fonseca, F., Egenhofer, M., Davis, C., & Câmara, G. (2002). Semantic Granularity in Ontology-Driven Geographic Information Systems. *Annals of Mathematics and Artificial Intelligence, 36*(1-2), 121-151.

Golledge, R. (1995). Path Selection and Route Preference in Human Navigation: A Progress Report. In A. Frank & W. Kuhn (Eds.), *Spatial Information Theory-A Theoretical Basis for GIS* (Vol. 988, pp. 207-222). Berlin-Heidelberg-New York: Springer.

Gruber, T. (1993) A Translation Approach to Portable Ontology Specifications. *Knowledge*

Acquisition, 5(2), 199-220.
ISO (2001) *GDF - Geographic Data Files - Version 4*.0. ISO/TC 204 N 34, Technical Report ISO/CD 2001-02-14.
Kuhn, W., & Raubal, M. (2003). Implementing Semantic Reference Systems. In M. Gould & R. Laurini & S. Coulondre (Eds.), *AGILE 2003 - 6th AGILE Conference on Geographic Information Science* (pp. 63-72). Lyon, France: Presses Polytechniques et Universitaires Romandes.
Lu, J., & Mylopoulos, J. (2002) Extensible Information Brokers. International Journal on Artificial Intelligence Tools 11 (1): 95-115.
OGC (1999a) OpenGIS® Simple Features Specification For OLE/COM. Tech. Rep. 99-050, Open GIS Consortium, Inc.
OGC (1999b) The OpenGIS™ Abstract Specification, topic 11: Metadata. Version 4. Tech. Rep. 99-111r1, Open GIS Consortium, Inc.
OGC (2001) The OpenGIS™ Abstract Specification, topic 1: Feature Geometry (ISO 19107 Spatial Schema). Tech. Rep. 01-101, Open GIS Consortium, Inc.
Raubal, M., & Kuhn, W. (2004). Ontology-Based Task Simulation. *Spatial Cognition and Computation* 4(1): 15-37.
Tele Atlas NV (2001) MultiNet™ Standard Format Specifications ASCII Sequential, version 3.1.
Thiran, P., Chougrani, A., Hick, J.-M., & Hainaut, J.-L. (1999) Generation of Conceptual Wrappers for Legacy Database, LNCS 1677, Springer, 678-687.
Zaslavsky, I., Marciano, R., Gupta, A., Baru, C. (2000) XML-based Spatial Data Mediation Infrastructure for Global Interoperability. *4th Global Spatial Data Infrastructure Conference* , Cape Town, South Africa, 13-15 March 2000.
Zhang, J., Javed, M., Shaheen, A., Gruenwald, L. (2000) Prototype for Wrapping and Visualizing Geo-Referenced Data in A Distributed Environment Using XML Technology. ACM-GIS 2000: 27-32.