An Ontology-Based GeoDatabase Interoperability Platform

Serge Boucher and Esteban Zimányi
Université Libre de Bruxelles, Belgium

Abstract: This chapter presents an ontology-based platform enabling automatic translation between a large number of geographical formats and data models. It explains the organizational motivations for developing this system, the technologies used, how its architecture and processing components were developed, what it achieves and where it still needs improvement. Since current off-the-shelf description logic reasoners are unable to process the large ontologies involved in this system, this platform uses a custom mapping algorithm that scales gracefully and still computes the required information to effect translation between supported data formats. The authors believe that the lessons learned during this project and discussed in this chapter will prove especially useful to interoperability practitioners contemplating the use of semantic technologies for enabling large-scale integration across organizational boundaries.

Keywords: Information Systems, Research and Development, Cartography, Emerging Information technologies, Military IS, IS Integration, IS Architecture, Information Exchange, Knowledge Integration, Geographic Information Systems, Internet Technologies, Web Technologies, Data Integration, Geospatial Data, Spatial Data, Spatial Database, Ontologies

INTRODUCTION

Achieving efficient data conversion and integration between information sources has always been crucial for extracting maximum value from database assets, and is one of the most important areas of research for us at the CoDE department of ULB. This case presents one of the results of this research, a platform for interoperating geographical information sources, developed upon request from one of our industrial partners.

BACKGROUND

The client for this interoperability platform is responsible for the evaluation and development of the weapons systems used by one of Europe’s national armies. Historically, the geographical databases used in the various weapon systems deployed by the defense forces were the responsibility of each arm. For example, the Army was solely responsible for procuring the cartographical data used to feed moving map devices installed in tanks and other land attack vehicles. Warships, on the other hand, are equipped not with moving maps but with chartplotters, and the data they used was the responsibility of the Navy. While from a technological perspective both these devices are very similar to each other, (as well as to GPS receivers now present in
many cars) for historical reasons they tend to use completely different data formats. The Army
and the Navy were thus simultaneously tasked with the development of data sets to accommodate
subtly different use cases and physical formats. Predictably, they came up with schemas that had
large intersecting areas of expressivity but yet presented many subtle differences that made them
completely incompatible. Over the years, this problem was repeated again and again, and today
the terabytes of cartographical information critical to the army’s operations are stored in dozens
of different formats, with each pair of them exhibiting design differences, some fundamental and
some gratuitous, which makes data conversion between them extremely difficult.

The previous paragraph might give the impression that the army managers and executives who
allowed such a situation to occur have been incredibly shortsighted. This, however, couldn’t be
farther from the truth. As is so often the case in the computer science industry, a long series of
sensible decisions has led to a collection of legacy systems that are ill adapted to current and
future needs. A little background knowledge on the history of weapon systems helps explain why.

**Military Cartography**

Cartographical systems are used by the military at all stages of operations, sometimes directly by
humans and sometimes as an input to a largely unsupervised computer process. These various
cartographical systems have to accommodate a large disparity of user interface and timing
constraints, from relatively slow systems limited by the speed of human reasoning, to the most
stringent real-time constraints current technology can offer. At one end of this scale lie the maps
and charts used by higher command for planning strategic operations. These have relatively lax
timing constraints, and are thus supported mostly by paper charts on which physical markers are
laid, due to the very convenient user interface these offer. A bit further down the scale, we find
the systems that power *situation rooms*, where tactical operations are directed. Response times
here are measured in minutes, and computerized systems are expected to show a representative,
real-time updated view of the tactical situation in a field hundreds of kilometers wide. Here the
main bottleneck remains human reaction time. Space constraints being relatively lax, powerful
computers and large screens can be used to their full potential. Still further down the scale, tank
pilots use the aforementioned moving maps to get a better understanding of the tactical situation
in their close vicinity. Reaction times here are measured in seconds, and space is extremely
limited. Towards the end of our scale we find terrain-following autopilots, used in many
warplanes and cruise missiles. Those are quintessential real-time systems: any delay in retrieving
needed information or computing the appropriate trajectory leads to either overshooting the
altitude bound and risking detection by enemy forces, or failing to pull up in time and crashing
into the ground.

Computerized cartographical information systems started being designed for the military in
the nineteen-seventies and eighties, when available computing power was extremely scarce by
today’s standards. At that time, getting such a system to work reliably was a big enough
challenge. Its suitability for future needs and compatibility with other existing systems were,
quite simply, out of scope. For example, designing an embedded moving map that could
accommodate many different data formats and be upgraded in the future to support other formats
was deemed impossible. However expensive converting available datasets to the format imposed
by the system might prove, at least it was technologically feasible.

In addition to the technological constraints of the time, operational realities help explain why
so many different formats came to exist. The various embedded systems were often *designed* in
isolation because they were expected to *operate* in isolation. A moving map inside a tank shows a
representation of the tank’s surroundings. A computerized situation map in a command and control room shows a representation of the entire theatre of operations. While it is obviously desirable that these two views be coherent with each other, it is much more important for them to be as close as possible to reality. Given the different technological constraints imposed on the two systems, it makes sense that the data sets they rely on would be created and updated independently from each other, and it would fall on the users to resolve any incoherence that may reveal itself during usage.

These circumstances have largely changed. Computing power is now much more plentiful, and human work comparatively more expensive, leading all industrial IT departments to shift more and more toward automation. This is especially true in the military of European nations, who are under growing political pressure to become more efficient. While in the past independence between military branches justified the work duplication described above, military commands are now expected to pool their resources at the national and trans-national levels. Indeed, NATO powers have made tremendous efforts at technology sharing: for example, weapon systems like air-launched missiles are now designed to be interchangeable between member armies, and global infrastructure projects are now pooled across nations, as in the Skynet satellite network (Amos, 2007). In this context, armies are now expected to make use of data sets available from allied countries rather than recreating them in-house. Obviously the organizational factors that lead to multiple incompatible formats were equally present in those other countries, which means that the number of different cartographical formats used by all NATO countries is substantially greater than that used inside any single country. However, there now is a rationale for enabling interoperability between all those formats.

Given this situation, our client had over the past decade launched many small research projects exploring the possibilities for interoperating some of those formats, leading to the project studied in this case. Most of these projects centered on problems found in specific conversions, e.g. converting VMAP2i data to VMAP1 (“Vector Product Format,” n.d.) is difficult because they store data at different scales and automatic geometrical generalization is still an open research problem. (Chan, 2002) While these small projects revealed that in some cases automatic translation could never be expected to produce satisfactory results, they also showed that the general approach was worth pursuing further. It was thus decided to study approaches to provide a unified solution to all the agency’s data cartographical data translation needs. This case discusses the results of this study.

CASE DESCRIPTION

Having described the organizational and managerial context of this case, we now present the technical specifications for the system we designed and its implementation. We start with a very brief primer on geographical databases, then present useful previous work on the subject of geographical database interoperability. We then describe the high-level specification of our translation service and finally discuss how semantic technologies were used to effect the actual translation.

Geographical Databases

In the most general sense, a geographical database is any structured collection of information that has some geographical component. Many subcategories exist: geographical coordinates can be two- or three-dimensional, and the information can be stored either as raster or vector data. The
distinction is similar to the one between bitmap and vector image formats: raster formats are
better suited for representing information sampled on a closely spaced grid (e.g. patterns of rain
over an area) while vector formats work better for resolution independent data (e.g. a map
showing buildings and roads). Efficiently storing and processing significant amounts of
geographical data is a challenging problem, which has lead to the development of Geographical
Information Systems, or GIS, which serve the same purpose for geographical data that RDBMS
do for relational data. Popular systems include Oracle Spatial and ArcGIS among others. (“Oracle
Spatial,” n.d.)

Our system processes two-dimensional vector geographical datasets representing
cartographical data. Conceptually, these can be thought of as unordered lists of features, where a
feature is a representation of some real-world object, e.g. the mountain Everest or the Golden
Gate Bridge. Figure 1 presents a small example dataset containing three features: a coastline, a
lighthouse and an industrial area.

![Figure 1. Sample Geographical Dataset – Graphical View](image)

Each feature in such a dataset has an associated spatial extent, which describes its geometry. In a
2D database, this extent can have up to 2 dimensions. Features with 0 dimensional (resp. 1, 2)
spatial extents are called point features (resp. linear, area features). On Figure 1, the lighthouse is
a point feature, the coastline a linear feature and the industrial area an area feature. In addition to
the spatial extent, most features have a number of properties, which store additional information
about the feature. Examples of such information are: the name of a city, the height of a tower, or
the type of activity that occurs in a given area.

While some GIS allow the creation of an arbitrary list of features, geographical databases are
vastly more useful when they follow some kind of schema or data model. A geographical
database schema specifies a list of feature types that features can belong to, e.g. Lighthouse, Coastline, Ocean, and a list of properties that can be applied to these features with associated
domains and ranges. The domain of a property is the set of feature types that it can be applied to,
e.g. features of type Tower can have a property Height, while features of type Ocean cannot. The
range of a property describes the kind of value that this property can take. This range can be
either a data type, e.g. accepted values for the property Name are alphanumerical strings, or a
discrete set of enumerated values, e.g. the values allowed for TypeOfIndustrialArea are:
Production, Refining, Manufacturing, etc. In the rest of this chapter we will use the term concept
to mean any of feature type, property, or enumerated property value. The set of concepts defined
in a data model and the relationships between them determines the expressiveness of this data
model, i.e. the set of physical realities that it can accurately represent.
Below is a simplified excerpt showing how a geographical information system would store the dataset drawn in Figure 1 above:

<table>
<thead>
<tr>
<th>FeatureType</th>
<th>SpatialExtent</th>
<th>Coords</th>
<th>Name</th>
<th>Height</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lighthouse</td>
<td>Point</td>
<td>35,53...</td>
<td>Punta Almina</td>
<td>148</td>
<td>22</td>
</tr>
<tr>
<td>Coastline</td>
<td>LineString</td>
<td>35,5...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WaterBody</td>
<td>Polygon</td>
<td>35,3...</td>
<td>Strait of Gibraltar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Industrial Area</td>
<td>Polygon</td>
<td>35,53...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Geographical Database Heterogeneities**

Like relational databases, two different cartographical data formats can exhibit different levels of heterogeneity. The most obvious is the physical level: some formats are stored in (collections of) flat files, others in relational databases, still others in XML. This level is the one most easily dealt with: writing converters between physical data formats is tedious, but it only has to be done once for each physical format, and either the conversion can be done without loss of information, or the information losses are easily found, described and understood. Next comes the logical level, which describes differences in structure between two data models. While quite troublesome for relational databases, this doesn’t appear for the formats we had to process because they all share a common structure: the feature-attribute-extent hierarchy described in the previous paragraph.

Geographical databases have a unique heterogeneity level that is not present in traditional relational databases: the geometric level. Equivalent features in two different formats sometimes are described with different types of spatial extents. For example, one format may allow arbitrary curves for linear features while another only allows straight lines. One format may store the location of a building as a simple point, while another would store its entire footprint as an area feature. While dealing with those heterogeneities is an active research problem that may never be completely solved in the general case, it is entirely orthogonal to the problem of semantic heterogeneity, so we won’t discuss it further here. The semantic level is the last and most important level of heterogeneity. It deals with the actual meaning of the terms used in the schema specification. Consider the term “lighthouse”, which appears in many cartographic format specification: many land cartography formats define (sometimes implicitly) a “lighthouse” as any tall building that was once used for marine navigation, focusing on the building’s aspect as seen in daylight from a short distance. On the other hand, a maritime format such as AML defines lighthouse as any structure that supports a light currently used for navigation, focusing on how the object appears at night from the sea. Some formats don’t define “lighthouse” at all, considering that real-world lighthouses can be modeled using a more general feature type such as Tower or Building. Solving the translation problems caused by these semantic heterogeneities in an automatic fashion was the main goal of the research project discussed in this case.
Related Work

Substantial work has been done on using semantic technologies to help process geographical data, starting with quantitative spatial reasoning using predicates based on Allen calculus (Allen, 1983) and its 2D extensions. (Cohn, 1992) The long-term goal of spatial reasoning is to find a formal logical representation for general spatial relationships that can be used for inference and reasoning, a goal that has proven quite difficult to attain. (Dutta, 1989) More recently, focus has shifted to exploring ways of embedding geographical information in Web 2.0 and semantic web applications. Standardized geographical ontologies are being engineered (Defne, 2004; Goodwin, 2005) and research on building the geospatial semantic web is active. (Aufaure, 2008) None of these are however addressing database interoperability per se. There has been significant research on using ontologies for spatial database integration, often to help domain experts find feature mappings that will be used for building the converters. Our work focuses on a different problem – using ontologies built by domain experts to enable automatic translation – on which much research still needs to be done. (Hakimpour, 2003)

Also related to this case are results on automatic ontology alignment, both for geographical (Bucella, 2007; Dolbert 2008) and general ontologies. (Euzenat, 2007) These however tend to focus on automatically mapping concepts based on their linguistic and structural similarity, which is not our focus here. We rely on domain experts to define concepts and focus on streamlining the conversions between multiple formats. One pioneering application of ontology-based interoperability was developed for the description of manufacturing processes, leading to the standardization of the Process Specification Language. (“Process Specification Language”, 2003) Ontologies have also been used to facilitate (non-geographical) relational database interoperability, most notably in the QuONTO project. (Acciari, 2005)

As far as we know, the system presented here is the first attempt to develop an automated translation platform for cartographical systems based on semantic web technologies.

System Design

The goal of this research project was to prove that universal data access and conversion could be implemented across all studied formats at reasonable cost, and to evaluate the level of data quality that could be expected from an entirely automated process. This means that the user interface specifics of the final visualization system were out of scope: to prove that some AML data can be shown on an embedded VMAP1 viewer, we don’t need access to the actual viewer. We only need to convert the data to VMAP1, ensure that it meets the VMAP1 specification and evaluate the resulting data quality on a viewer program that allows visualizing VMAP1 data. We were thus able to make our system’s interface extremely simple. It is implemented as a Web Service that accepts one single type of request, containing the location of the source data set and an identifier for the target format. The result of this request is the converted dataset. A high-level view of this process is presented in Figure 2 below.
Translation is performed using a two-layer approach, dealing first with physical then with semantic heterogeneities. The first step consists of translating the source data to the Geography Markup Language, (“GML”, n.d.) which is an XML language standardized by the Open Geospatial Consortium. It has been designed to provide a universal physical representation of geographical data to enable interchange between different GIS applications.

We use different tools for the two translation directions. To convert the source data to GML, we use a Web Feature Service. This is another OpenGIS standard that specifies a web service that outputs GML data. Current WFS implementations are available for most common geographical databases, including all the formats that we had to process. The other side of the conversion, from GML to target data format, was less straightforward. Using Safe Software’s Feature Manipulation Engine (“FME Server”, n.d.), we implemented a converter for each target format that takes GML as input. At the end of the translation process, the main Web Service controller calls the appropriate converter, which outputs the final dataset in the target data format.

Having resolved physical heterogeneities using GML converters, the hard task of converting the source GML to another GML that matches the target format semantics remains to be done. Since GML is an XML format, the most natural tool to process it is XSLT. The problem, however, is to economically generate one XSLT stylesheet for each pair of source and destination formats when the number of formats is large. This is where making the semantics of each data format explicit becomes invaluable. The remainder of this section describes this process in detail.
Semantic Translation

The goal of this project was to enable automatic translation between any two datasets stored in a known, but large, number of different formats. Figure 3 below illustrates two naïve approaches for achieving this goal.

![Figure 3. Two Approaches for Translating Between Many Different Schemas](image)

The first solution, pictured on the left, is to manually implement a converter for each pair of formats. This solution allows the implementer to tweak each and every translator and thus potentially provides the best translation quality. However, it has the obvious drawback that for $n$ different formats it requires $n(n-1)$ individual translators, making it prohibitively expensive when the number of formats is large and likely to grow in the future. Our case study currently has dozens of formats with new ones added each year, making this option unviable.

Another naïve possibility is to build a central pivot format, which is expressive enough to encompass all the others and to translate each format to and from it, as depicted on the right-hand side of Figure 3. This approach sounds very attractive in theory, since it seems to solve the problem with comparatively little work: the only requirement is to define a single new data schema and to implement a two-way converter for each of our original formats. However, closer examination reveals that it doesn’t actually work: even setting aside the huge complexity involved in creating and maintaining the very expressive pivot schema, this approach fails to solve the underlying translation problem.

Inasmuch as there are intersections between the expressiveness of the original formats — and if there are none, data translation is both trivial and useless — a pivot format expressive enough to represent any possible data from the original formats will necessarily afford many different ways to represent the same information. For example, if format A represents a lighthouse using a specific feature type called Lighthouse while format B instead relies on the generic feature type Building with a qualifying attribute BuildingFunction valued as Lighthouse, then the pivot format needs to define both Lighthouse and Building feature types as well as a BuildingFunction property with Lighthouse as one of its possible values. Now, translating towards the pivot format is easy: we just map each original feature type to the equivalent pivot feature type. However, the opposite transformation is much more complicated. Since a real-world lighthouse could be represented in the pivot format using either its specific Lighthouse feature type or using the generic Building with an associated BuildingFunction of Lighthouse, the pivot-to-target converter has to handle both cases. More generally, each pivot-to-format converter has...
to be aware of all other existing formats, and thus the pivot format does not solve our translation problem at all.

Thus, neither of these two approaches work: manually implementing individual converters requires a workload that rises quadratically in the number of formats, and while the pivot approach seems to require only linear workload it fails to address the actual translation problem. However, studying these two failed solutions suggests an attractive possibility: what if there was a way to build the $n(n-1)$ converters with only linear workload?

Our system attempts to do exactly that. By describing each individual format using mappings from its concepts to those defined in a domain ontology — which can be done for each format independently and thus requires only linear time in total — it becomes possible to automatically generate data converters between any given pair of formats. The remainder of this section describes the entire process.

**Ontology Building**

The cornerstone of our approach is a domain ontology, which is an OWL ontology that describes all real-world concepts relevant to the domain of cartography. It is grounded on three disjoint taxonomies: one for feature types, one for properties, and one for property values which describes the generalization/specialization relationships that exist between those real-world concepts. For example, the Factory feature type is a special kind of Building, the hasIndustryCategory property is a specialization of the hasUse property, and the OilRefining property value is a special case of the “Industrial” value. Figure 4 below shows a small fragment of this ontology.

![Figure 4. Fragment of Domain Ontology](image)

The purpose of this ontology is to provide a formal definition for concepts that appear in each of the formats to be translated. These formats’ specifications are most often flat, i.e. they define a list of feature-types with only implicit relationships between them. A format might define
different feature-types for Lighthouse, Tower and Building, and while it is obvious from reading the specifications that a Lighthouse is a special kind of Tower which is a special kind of Building, this knowledge is not present in the original schemas in machine-readable form. Hence we spell it out in the Domain Ontology.

With this knowledge formally stated in the domain ontology, it can be used to describe all features and properties appearing in each format. After expressing each format schema in OWL form, by creating a “format ontology” that defines an OWL concept for each feature type, property, and property value defined in the format, we write a mapping between these format-specific OWL concepts and the concepts defined in the domain ontology. This step gives tangible meaning to the feature types specified in each format: if a land cartography format defines the feature type Lighthouse as a tower-shaped building that sits near the sea but may not be used for marine navigation anymore, this feature type will be mapped to the domain ontology concept HistoricLighthouse. A Lighthouse feature type defined in a maritime cartography format might however refer to any building that contains a light source used for navigation: this feature type would be mapped to the domain concept LandBasedNavigationalAid. A feature type defined as having the shape of a traditional lighthouse and being used for navigation would be mapped to the domain concept Lighthouse, which is subsumed by both previous concepts. An example of such format-to-domain mapping is shown in Figure 5 below:

![Figure 5. Format-to-Domain Mapping](image)

The way these ontologies are built does not matter much. Best practices dictate that the domain ontology should be created by trained ontology engineers with input from domain experts, and each format-specific ontologies and mappings should be created by specialists in each of these formats with support from the engineers who created the domain ontology. Our team developed the prototype ontologies used for this project after close examination of all format specifications and with occasional support from experts in these formats. In the end, the only important thing is for the domain ontology to ensure sound translations of feature types when direct format-to-format mappings don’t exist. Each “A subsumes B” relationship in the domain ontology means that while translating from a format that defines B to one that does not, A is a reasonable substitute.
Translation Algorithm

Given an ontology formally defining domain concepts and mappings linking format-specific concepts to those definitions, designing an algorithm to translate data between any pair of formats is reasonably straightforward. The basic idea is as follows. Feature types, properties and property values are treated separately but similarly. Each of these concepts needs to be mapped to some concept in the target format, if at all possible. When a fully equivalent concept exists in the target format, the mapping is trivial. Otherwise, we look for a more general concept by walking up the domain ontology breadth-first until we encounter a concept that is defined in the target format. If we find one, it is mapped to the source concept. If we don’t, then a mapping cannot be found, thus information about this concept is not expressible in the target format and we are forced to drop it. The following describes the translation process in more detail.

As seen earlier, the very first step is to convert the source dataset from its native format to GML, which is a straightforward process, handled by an industry-standard Web Feature Server. Then, this GML dataset is converted to OWL instances belonging to the source format ontology. This OWL dataset is then processed to obtain another OWL dataset where instances belong to the target format ontology. It is then again straightforward to convert this dataset to GML and then to the target format. This process is illustrated in Figure 6. Obviously, the difficult step is the conversion between source and target OWL datasets, i.e. how can we express the knowledge defined according to the original concepts using only concepts defined in the target format.

Our first approach was based on description logics reasoning. In theory, loading the original OWL dataset in a DL reasoner along with the source, domain and target ontologies reduces the conversion problem to an instance-retrieval problem: doing a series of instance retrieval queries on all the concepts defined in the target format would yield a dataset containing all the knowledge from the original dataset that is expressible with the concepts defined in the target format. For example, if the source data set contained a Lighthouse, and the destination data format defined a Building feature type, and the domain ontology stated that Building subsumes Lighthouse, then
loading all ontologies in a reasoner, classifying the merged ontology and doing an instance retrieval on the Building concept defined in the destination schema would return our Lighthouse from the source data set. Repeating the process with all the other target concepts would give us all the information from the original dataset that can be expressed in the destination schema.

While this solution is elegant and works well with toy examples, current reasoning technology has scaling problems that make it impractical for our purpose. The cartographical formats we studied define thousands of feature types, properties and property values. Combining format ontologies with the domain ontology yields a data structure that is much too large to be classified by current tableaux-based reasoners.

There is, however, a workaround. We designed an algorithm that walks the source format ontology tree to generate a mapping between source and destination format, and store this mapping as an XSLT file. Once this is done, data conversion only requires executing this XSLT with the OWL source instances as input. The main drawback of this approach is its lack of generality. Firstly, our algorithm only processes the OWL constructs we used and thus can’t handle general OWL-DL ontologies. Secondly, given that XSLT is not specifically an OWL processing technology, and that a given OWL ontology has many different XML serializations, our XSLT has to make a number of assumptions on how the OWL was serialized. Thus it cannot handle general OWL/XML files. In practice, it only works on OWL files generated by our GML-to-OWL converters.

The payoff for this lack of generality is that our algorithm does not suffer from the combinatorial explosion of all-purpose OWL-DL reasoners and thus is orders of magnitude faster in our case. Due to the way they’re engineered and the characteristics of our domain, our ontologies have a very clean structure that allows very efficient processing if the processor is aware of this structure. This is not the case for tableaux-based reasoners, which don’t make any assumption on how ontologies are built. Our algorithm works as follows:

```
foreach source_feature in source_format:
    domain_class ← getDomainClass(source_feature)
while domain_class != #Feature:
    if domain_class has equivalent target_feature:
        mapping_found(source_feature, target_feature)
    else:
        domain_class ← parent_breadth_first(domain_class)
```

For each feature type present in our source format, we walk the domain ontology to try to find a mapping with a feature type in the destination format. If the domain class equivalent to our source format feature type has an equivalent destination format feature type, then we have a direct mapping. If this is not the case, we walk the domain ontology upwards and breadth-first, starting at source feature-type, until we find a domain class with an equivalent destination feature class. If we encounter #Feature (the super-concept of all feature types in the domain ontology) before we’ve found a mapping, then this source feature type is not expressible in the target format.

A similar procedure is followed to generate property and property-value mappings. All three sets of mappings are then serialized as XSLT templates which, when executed, translate source OWL instance data into target OWL instance data.

**Extensions**

While the algorithm described above is the essential idea behind our translation platform, as presented it can only handle simple one-to-one feature mappings, which is too simplistic for
many of the conversions we had to handle, so a number of extensions were created to handle the more complicated cases. This subsection presents some of these implemented extensions.

The first extension deals with a type of mapping we call *conditional mapping*. Some formats use the same feature types for conceptually different entities and distinguish between them using different values for a qualifying property. For example, a lighthouse is represented in VMAP2i by using the feature type Building and setting the property hasBuildingFunctionCategory to 82. This is expressed in our format-domain mapping by using an OWL restriction, as shown on Figure 7 below. When writing the XSLT transform for such a mapping, our algorithm generates an appropriate selector (when the restriction appears in the source format) or writes out the property value and inhibits transforms on conflicting properties (when the restriction appears in the destination format).

![Figure 7. Conditional Mapping](image)

While this type of conditional mapping is easily expressible in OWL, others are not. Some formats require conditions on DataType properties, e.g. a tower higher than 150 meters should be represented using the Skyscraper feature type. To deal with these mappings, we extend the OWL restriction syntax as shown below:

```xml
<owl:Restriction>
  <owl:onProperty rdf:resource="height"/>
  <owlext:hasValueSuperiorTo>150</owlext:hasValueSuperiorTo>
</owl:Restriction>
```

Lastly, we have only discussed here the semantic aspects of data conversion. Our translation engine also deals with units of measures as well as geometric aspects such as specialization and generalization, scale-dependent linear approximations and point-area conversions, but these are out of scope for this chapter.

**DISCUSSION**

This section explains in more depth the choices we made while designing this system and experiences we learned during early prototype testing. The first subsection discusses our choice of ontology modeling language. The second subsection explains at length the differences between the role of our domain ontology and the data model of pivot format approaches. While our approach has the obvious benefit of making cartographical formats interoperate with an amount
of work that rises linearly in the number of formats, instead of quadratically, it does have a few drawbacks, which we discuss in the third subsection. Finally, the fourth subsection discusses the performance of our system and strategies for optimization.

**Semantics**

It should be noted that our use of semantic technologies differs from the one envisioned in the early views of the semantic web on at least two points. Firstly, we don’t reuse any existing ontologies. Secondly, our use of OWL is somewhat unorthodox: although we only use a relatively small subset of the OWL vocabulary, some of its constructs are given slightly different semantics than they have in OWL-DL, which might be misleading for someone accustomed to working in this language. Obviously, the right design for a computer system is the one that is best at performing its current and future tasks, not the one that is “purer” in the technologies it uses, but given the intent of this book and the ongoing debate on those issues in the semantic web community we think it worthwhile to explain the reasoning behind those design choices.

Our decision not to reuse existing ontologies is easy to explain: no suitable ontology could be found. While the number of formally defined and publicly available OWL and RDF files is certainly rising, there are still many domains that are not adequately covered by available ontologies, and military cartography is one of them. Although the ontologies published by Ordnance Survey (“Ordnance Survey Ontologies,” n.d.) were initially identified as good candidates for reuse, they are heavily tied to the data models used by the organization and reusing them for our domain ontology ultimately would have caused too many problems.

Our use of OWL deserves further discussion. In any project using semantic technologies, careful consideration should be given to the ontology language used. Some projects are best served by RDF/RDFS. Others will require some level of OWL. Some models are best expressed in OWL-Full, but standard DL reasoners only work with strict OWL-DL. In any case, it is obviously desirable to respect the standard semantics of OWL constructs: this makes the ontologies easier to document and facilitates integration with other semantic web projects.

In our case, the original plan was to use OWL-DL — originally one of our goals was to evaluate how description logics reasoning could help resolve geographical database heterogeneities. As explained above, the size of our ontologies made this approach difficult, but this was not the only problem: the constraints of OWL-DL made some modeling situations tricky. For example, some properties accept enumerated values in some formats but unstructured string values in others. The logical way to model this in OWL is to use an ObjectProperty in the first case and a DatatypeProperty in the second. When these are linked to the domain ontology, we get a sub-property relationship between an ObjectProperty and a DatatypeProperty. This is not allowed in OWL-DL, and for a very good reason: lifting this restriction leads to an undecidable logic. However, in our case this modeling choice is reasonable and its meaning is easy to understand.

Because of those problems, using off-the-shelf reasoners on this project would have required substantial pre- and post-processing steps, adding significant complexity to the system with little benefits in return. We thus decided to stick with the more natural models and find a way to exploit them to their fullest, leading to the translation algorithm that is the main topic of this case.

It can be argued that our system uses a good part of the OWL syntax while giving it different semantics, which might be confusing for people accustomed to pure OWL-DL. However, a more accurate view is that our ontologies are really RDF ontologies that happen to use many components of the OWL vocabulary — after all, any OWL ontology is also an RDF ontology.
OWL provides many constructs that are perfectly suited for our modeling purposes. For example, OWLRestriction is a natural fit for our conditional mappings, and while it would certainly have been possible to achieve the same result using ad hoc RDF predicates, reusing OWL ensures that our ontologies can be edited using standard OWL editors and eases the learning curve for any one familiar with OWL. Our system gives to nearly all OWL constructs the semantics they have in OWL-DL — the only exception is subPropertyOf, for the reason explained above, and this is thoroughly explained in the project’s internal documentation.

It should be noted that the practice of using OWL notation without OWL-DL semantics is used in many well known projects, notably Friend-of-a-friend. (http://www.foaf-project.org/)

**Pivot Formats and Domain Ontologies**

We gave a number of talks on the project and approach described in this case, and one question came up often enough that it deserves some discussion in these pages: how is our approach different from the pivot format approach shown in Figure 3? Looking at the translation process described in Figure 6, one can’t help noticing that the domain ontology in our system acts very much like the pivot format of Figure 3. Indeed, it contains all concepts that might appear in all supported formats and is the central reference point for generating inter-format mappings and thus translating data between different formats. How, then, is our approach different than the pivot format one and why doesn’t it suffer from the same drawbacks?

The answer has to do with the fundamental definition of an ontology, and specifically with what an ontology is not. An ontology is a formal specification of knowledge, not a data model, which is what the pivot format is. A data model is only a set of building blocks that can be used to describe the world. An ontology, on the other hand, actually describes the world, in a machine-readable way. To illustrate, let’s consider what a domain ontology and a data model might say about the real world concepts of Building and Lighthouse.

---

*Figure 8. Excerpt from Domain Ontology and Related Pivot Format*
As we can see on Figure 8 above, both ontology and data model acknowledge that in the world of cartography there exist things known as buildings and lighthouses. The data model provides two corresponding feature types that can be used by a human modeler to represent instances of these structures present at some location in the physical world. This, however, doesn’t help at all when we are given a Lighthouse instance in some format and we want to express it in another format that only defines the Building feature type.

On the other hand, the domain ontology does more than acknowledge the existence of these two feature types: it formally states that all Lighthouses are Buildings. This knowledge proves very useful when we want to represent a Lighthouse in a format that only defines Building.

The advantages of the domain ontology are even more obvious when we consider that other formats can be added to the system some time after the initial implementation. Assume that we add a format that defines the feature type Tower, which does not exist in any previous format and is thus not defined in our domain ontology or in the pivot format. Obviously they have to be modified, yielding the revised version of Figure 9 below.

![Figure 9. Adding a Tower Concept in Domain Ontology and Pivot Format](image)

With the pivot format approach, we are in trouble: we have to revise all existing pivot-to-target-format converters so that they accept instances of the Tower feature type as Buildings. With the domain ontology, however, nothing changes: since the subsumption relation is transitive, it still contains the knowledge that all Lighthouses are Building (as well as Towers), and conversions from formats that define Tower to those that don’t will work as expected.

More formally, the kind of knowledge stored in our domain ontology is *monotonous*, i.e. adding new concepts from new formats to the system does not invalidate previous knowledge.

It sometimes happens that the new format uses the name of some existing domain ontology concept with a different meaning. Imagine our system only deals initially with terrestrial formats, and the domain ontology contains a Lighthouse concept, a sub-concept of Tower that represents all tower-shaped buildings situated near a coastline that were originally built to facilitate nighttime maritime navigation. While this is the definition generally used for land maps, in most
maritime cartography formats a Lighthouse refers to a currently active land-based navigational light, regardless of the shape of the building it sits in. When adding such a maritime format to our system, obviously we can’t map its Lighthouse concept to the domain ontology concept of the same name: we have to define a new concept that matches the definition of Lighthouse as used in maritime cartography, which we might call MaritimeLighthouse. One could argue that the original Lighthouse domain ontology concept was badly named, and that the right solution is to rename it to LandLighthouse, change all existing format-to-domain mappings, and create a new Lighthouse concept that only applies to features that fit both maritime and land definitions. Such a change obviously breaks monotonicity: adding a new format broke existing knowledge.

Obviously, nothing forces us to do this: while the original name of the Lighthouse concept might have been badly chosen, its semantics are well defined. It is perfectly okay to leave the concept as it is, add a comment to the ontology documenting the fact that it only refers to Lighthouses as defined in land cartography, and create the new concept as LandAndMaritimeLighthouse. After all, the system does not care about the names given to domain ontology concepts. On the other hand, if this issue occurs often, it seems doubtful that we can keep adding comments to misnamed existing concepts and creating new equally misnamed concepts for the sake of backward compatibility without at some point getting stuck with a domain ontology that is impossible to maintain.

This problem of imperfect concept names is revealing of a larger issue: the domain ontology is a software artifact of significant size and complexity, and it is bound to evolve. As ontology evolution is still an active research area (Noy, 2004) and best practices for documenting ontologies are still being developed, keeping the domain ontology up-to-date and well documented while many formats are added to the system will likely raise some challenges. While dealing with change in a complex system is always difficult, the fact that ontology engineering is less mature than similar disciplines like relational data modeling might lead to unpredictable problems. How significant these will be and how best to overcome them will be studied while the client tests our prototype.

**Limitations**

Our approach to inter-format mapping is based on one core assumption: if a source concept has no equivalent in the target format, it should be mapped to a more general concept, or not at all. For example, if the source format defines a feature type for towers, and the destination format has no feature type for tower but has one for buildings, then Tower instances are converted to Building instances. If none of the parent concepts of Tower in the domain ontology are defined in the target format, then Tower instances are simply dropped during translation.

While this seems like a sensible approach, it should be noted that it cannot handle cyclical mappings. Imagine that there exists three concepts A, B, C defined in three different formats, and that a domain expert tells us that A should be translated into B, B into C and C into A. Our system cannot effect those translations: it is impossible to structure the domain ontology in such a way that the translation algorithm generates those mappings. It should be noted that we did not encounter this situation in our case, and that we cannot even fabricate a reasonable example of such features in the cartography domain. However it is conceivable that such a situation might arise if the same approach was to be used in other domains.

Another core assumption is that property value mapping is largely independent from feature type mapping. Thus our system is a bit awkward at mapping properties whose meaning depends on which feature they’re associated with. Consider a translation from format A to format B, with
features $A1$ and $A2$ mapping to features $B1$ and $B2$, and some object property $ap$ defined in format A with a domain that includes $A1$ and $A2$. Assume that property value $ap1$ should map to property value $bp1$ when it is applied to $A1$, but to $bp2$ when it is applied to $A2$, as is shown on Figure 10 below.

![Figure 10. Feature-Dependent Property Value Mapping](image)

Our system can handle such a case, but only if we create two ad hoc domain ontology concepts: one that maps to $A1$ with property $ap$ equal to $ap1$ and to $B1$ with property $bp$ equal to $bp1$, and one that maps to $A2$ with property $ap$ equal to $ap1$ and to $B2$ with property $bp$ equal to $bp2$, as seen on Figure 11.

![Figure 11. Domain Ontology for Feature-Dependent Property Value Mapping](image)

If there are two such properties then we need four domain ontology concepts, and the number rises exponentially. If this situation occurs often, we risk a class explosion in our domain ontology, which would become very difficult to maintain. We would argue that having the semantics of property values depend on their subject type is bad data model design, and it seldom
occurred in the formats we used for our case study, but this consequence of our approach could still be a problem in some cases.

**Performance**

Performance is an important concern in any translation architecture that deals with huge amounts of data. Our system converts data by using XSLT, a technology that is not known for its scalability, and any conversion requires three successive transforms, so for large datasets it certainly isn’t the fastest approach. In practice this turned out to be not much of a problem. Due to the nature of our datasets, each unique feature can be translated independently. Moreover, the XPath expressions appearing in our stylesheets only refer to very local XML nodes — typically the current node’s direct children or parent. These two factors mean that the translations run much faster than one would expect and can handle very large datasets: in our tests, the overall time spent on XML processing was small compared to the final conversion from GML to the target physical format.

It should also be noted that nothing in our translation architecture forces us to use XSLT. Mappings are generated as object-oriented data structures, and these could conceivably be converted to any translation mechanism. We currently support two: XSLT, which given the availability of our data in XML format was deemed the easiest to implement, and a plain text format that is not executable but has proven invaluable for evaluating the quality of our mappings and communicating with domain experts. If the system needed to be optimized past the point where XSLT became a bottleneck, it would be reasonably easy to use e.g. STX, SAX or JavaCC (https://javacc.dev.java.net/) instead. If even that proved too slow, we could even get rid of GML altogether without changing the mapping algorithm. This would allow substantial performance gains in the cases where source and target physical formats happen to be the same. These performance optimizations, however, were out of scope for our prototype implementation.

**CURRENT CHALLENGES**

The demonstrator system described above is currently being tested by our client and will be iteratively modified to correct deficiencies found during testing. While the general approach is sound, adding functionalities in the main algorithm and refining both the domain ontology and the different format ontologies will augment the usefulness of the system by improving the translations’ data quality.

Since the domain and format ontologies were engineered by computer scientists with limited domain knowledge, it is fair to say that they are currently far from perfect. While this is mainly a consequence of this project’s limited budget and its exploratory nature, some of the difficulties encountered seem like a direct consequence of our usage of semantic technologies. The consensus among knowledge representation experts is that ontologies should be created by collaboration between domain experts and ontology engineering specialists, (see for example: Gómez-Péres, 2004) however the specifics of such collaboration are rarely spelled out. It is easy to conceive of many arrangements that sound as if they should work. One possibility is for domain experts to assume most of the ontology creation work after having been briefed by ontology engineers on the rudiments of ontology engineering and associated tools. Alternatively, ontology engineers can build the ontology from some semi-formal description of the target domain, eventually asking domain experts for clarification whenever they need it. We mostly used the latter strategy, which required a lot of back and forth between ontology engineers and domain experts and thus made
the process of ontology engineering slower than initially expected. We made some attempts to use the first strategy but these were frustrated by the relatively steep learning curve of RDF and OWL. It is very easy to underestimate the misunderstandings that will arise both in the ontology engineers’ conception of the domain and in the domain experts’ understanding of the fundamentals of ontology engineering. After years of working in computer science, we were often surprised at how difficult it is for smart people who have little computer science background to grasp the precise semantics of even simple relations such as is-a or owl:Restriction. OWL and RDF seem like difficult languages to learn even by computer science standards, and while we can hope that at some point tools will be developed that lower the learning curve of ontology engineering, these tools do not exist yet. It seems to us that any project that requires substantial ontology engineering effort and deals with a complex domain will benefit from very careful study of these issues.

A future challenge for the prototype is support for real data integration, i.e. going beyond 1-to-1 data translation and offering the possibility of merging information stored in different data sources. While the system’s architecture was designed with this goal in mind, and limited ad hoc merging can be performed, the current implementation does not support it in the general sense. Specifically, there is no way to get all the information relevant to a given region of space in a single query. Supporting this use case would require permitting the user to express such a query in a dataset-independent way, querying all the available datasets with adequate translations of this query, translating all results to the destination format and finally merging them. The current system only supports the “translation” step of this process. Both merging geographical datasets without introducing duplicate features and ensuring good performance of such distributed queries are difficult problems.

**CONCLUSION**

All in all, the implemented prototype has shown that building and maintaining an industrial-strength system to support universal data access across all the formats used by our client will require a lot more study, design and implementation work. It has also opened the way for more ambitious use cases like data integration, which will also require significant advances to meet operational requirements. The goal of this project, however, was not to build a perfect system, but to explore ways in which current state of the art information technology could be used to allow cost-effective access to geographical databases across numerous formats and database schemas. While languages and tools for formal description and processing of the semantics of a database schema are still in their infancy, this project has shown that their use in an industrial setting can in some cases provide efficient and cost-effective solutions to large-scale infrastructural problems that initially seem intractable. While it is fair to say that knowledge representation still has a lot of room for improvement, it is also true that it is now mature enough to be used in an industrial context. Those who are prepared to invest enough time and effort to climb the inevitable learning curve of this emerging technology will likely find in it tremendous value for improving existing data infrastructure systems.
REFERENCES


---

i Even if we wanted access to the actual hardware, we would not have gotten it. The precise capabilities of some of these systems are classified and the army only issues security clearances when absolutely necessary.

ii More precisely, the specialization relationships hold not between the feature types themselves but between the real-world entities they’re meant to represent. Since in the geographical database each Feature has only one feature type, the sets of Lighthouses and Towers in the database itself are distinct, even though in the real world all Lighthouses are Towers. Ironically, format specification documents often encourage modelers to always use the most specific feature type suitable to represent a given real-world feature, but they almost never spell out the specialization relationships that exist between feature types, perhaps assuming them to be obvious.

iii The word “format” in “pivot format” is somewhat unfortunate in this context. A data format generally defines both a data model and a physical storage format. Since all translations in this project start and end in the same physical format (i.e.: GML), the pivot would likely use it as well, and its only relevant aspect would be its data model.

iv Note that this is not true of general OWL-DL ontologies, which can be made inconsistent and thus unusable by adding new axioms. Pure RDF/RDFS ontologies, however, are monotonous.

v Graphical ontology editors such as Protégé and TopBraid help the reader abstract himself from OWL syntax but still require a deep understanding of description logics to be used successfully.