Transactions in GIS

Transactions in GIS, 2012, 16(4): 455–476

Research Article

Geospatial Information Integration for Authoritative and Crowd Sourced Road Vector Data

Heshan Du
School of Computer Science
University of Nottingham

Suchith Anand
Centre for Geospatial Science
University of Nottingham

Natasha Alechina
School of Computer Science
University of Nottingham

Jeremy Morley
School of Geography
University of Nottingham

Glen Hart
Ordnance Survey
Southampton, UK

Didier Leibovici
Centre for Geospatial Science
University of Nottingham

Mike Jackson
School of Geography
University of Nottingham

Mark Ware
Faculty of Advanced Technology
University of Glamorgan

Abstract
This article describes results from a research project undertaken to explore the technical issues associated with integrating unstructured crowd sourced data with authoritative national mapping data. The ultimate objective is to develop methodologies to ensure the feature enrichment of authoritative data, using crowd sourced data. Users increasingly find that they wish to use data from both kinds of geographic data sources. Different techniques and methodologies can be developed to solve this problem. In our previous research, a position map matching algorithm was developed for integrating authoritative and crowd sourced road vector data, and showed promising results (Anand et al. 2010). However, especially when integrating different forms of data at the feature level, these techniques are often time consuming and are more computationally intensive than other techniques available. To tackle these problems, this project aims at developing a methodology for automated conflict

Address for correspondence: Suchith Anand, Centre for Geospatial Science, University of Nottingham Innovation Park, Triumph Road, Nottingham NG7 2TU, UK. E-mail Suchith.Anand@nottingham.ac.uk

doi: 10.1111/j.1467-9671.2012.01303.x

resolution, linking and merging of geographical information from disparate authoritative and crowd-sourced data sources. This article describes research undertaken by the authors on the design, implementation, and evaluation of algorithms and procedures for producing a coherent ontology from disparate geospatial data sources. To integrate road vector data from disparate sources, the method presented in this article first converts input data sets to ontologies, and then merges these ontologies into a new ontology. This new ontology is then checked and modified to ensure that it is consistent. The developed methodology can deal with topological and geometry inconsistency and provide more flexibility for geospatial information merging.

1 Introduction

The context of this article is the need to address the separation of national and international spatial data infrastructures, such as the European INSPIRE SDI (European Commission INSPIRE, 2011), from crowd-sourced geospatial databases, such as OpenStreetMap (OpenStreetMap 2011, Anand et al. 2010). Crowd-sourced data sources rely on volunteers to collect data. Although typically not as complete in its coverage or as consistent in its geometric or metadata quality as authoritative data, crowd-sourced data may provide a rich source of complementary information with the benefit of often more recent and frequent up-date than is the case for authoritative data (Jackson et al. 2010). Crowd-sourced communities and governmental agencies can both benefit through communicating and collaborating to improve the overall quality (richness, consistency, accuracy, timeliness, and fitness for purpose) of geospatial information. Furthermore, in the ever-changing world, there is an increasing need for the representation of knowledge of objects to be fluent, changing during its use (Bundy and McNeill 2006). This article describes research undertaken by the authors on the design, implementation, and evaluation of algorithms and procedures for producing a coherent ontology from disparate geospatial data sources. It builds upon previous work on ontology-based geographical data integration that investigated feature matching using a geo-semantic algorithm for position and high level ontological description (Du et al. 2011).

Ordnance Survey and OpenStreetMap were used as examples of authoritative and volunteered data sources, respectively to carry out this research. Ordnance Survey is Great Britain's national mapping agency, which provides the most accurate and up-to-date geographic data (Ordnance Survey 2011). OS MasterMap is a digital map product launched by Ordnance Survey. It includes several "layers", and one of them is the Integrated Transport Network (ITN) Layer, consisting of vector data on transport features, such as roads. OpenStreetMap (OSM) is a free map of the whole world, and it allows everyone to view, edit, and use geographical data in a collaborative way (OpenStreetMap 2011). OpenStreetMap is an open initiative to create and provide free geographic data such as street maps to anyone who wants them. The wiki-style data collection allows it to capture changes in the physical world quickly, but also explains the possibility of inconsistency of data. The OSM vector data has three layers (point, line and polygon), among which the line layer consists of the data for roads.

The research has two main objectives. The first is to develop appropriate methodologies for geospatial information linking, and the second is to develop techniques for merging geospatial information from disparate sources. Firstly, "linking" refers to finding the correspondence relationship between data in different data sets, which may have different conceptual, contextual, topological representations. As a simple example, consider two data sets, one of which refers to road names using the label "name" while the other uses the label "ROADNAME". In order to link equivalent road objects using this label it might be necessary to make the basic formats uniform (e.g. by converting all labels to uppercase) followed by a comparison ("NAME" vs. "ROADNAME"), which would be required to recognize that different labels may have the same meaning in a particular context (e.g. "NAME" and "ROADNAME" have no difference in the meaning here). Secondly, "merging" refers to combining information from disparate sources into a consistent ontology. Consistency (together with accuracy, timeliness, richness of information, and fitness for purpose) is one of the important aspects of information quality. Consistency here means there are no logic conflicts with regard to facts about the same attribute of an object.

The remainder of the article is set out as follows. Related work is discussed in Section 2. Section 3 outlines the system design processes giving details about the system architecture, the ontology design, consistency definitions, algorithm design for geospatial merging of crowd sourced and authoritative road vector data sets, as well as the user interface design for the developed software. A prototype implementation of the system is described in Section 4. Experimental results, evaluation and discussion on design choices, and limitations of the system are presented in Section 5. The article concludes in Section 6 with a summary and discussion of future work.

2 Related Work

In philosophy, ontology studies the categories of things that exist or may exist in some domain (Sowa 2000). In information science and Artificial Intelligence, it refers to a formal representation of knowledge by a set of concepts and their relationships within a domain. Ontologies play an important role for knowledge sharing and are widely used to structure domains of interest conceptually (Stumme and Maedche 2001). Ontologies can exist in many different forms, which makes it difficult when one wants to use them together. Thus, some work needs to be done to find correspondence among ontologies, and judge which concepts are similar, overlapping or unique (Noy and Musen 2000). However, currently most of the work, including ontology linking and merging, is performed manually. Ontology linking or alignment refers to finding correspondences between concepts, which have the same meaning, from different ontologies. Ontology merging is creating a single coherent ontology which contains information from all the sources. Without intelligent support, both are quite difficult, labor intensive and error prone (Stumme and Maedche 2001). Thus, it is desirable to automate the process fully or partially. According to a Choi et al.'s survey (2006), tools for ontology linking and merging include SMART (Noy and Musen 1999), PROMPT (Noy and Musen 2000), OntoMorph (Chalupsky 2000), Chimaera (McGuiness et al. 2000), Anchor-PROMPT (Noy and Musen 2001), and FCA-Merge (Stumme and Maedche 2001).

SMART is a semi-automatic ontology merging and alignment tool, which generates an initial list of suggestions based on linguistic similarity of class names, and performs automatic updates based on users' selections, and creates new suggestions (Noy and Musen 1999). Noy and Musen (2000) also developed PROMPT, providing a semi-automatic approach to ontology merging and alignment, based on a general OKBC-compliant knowledge model. PROPMPT uses some strategies to guide users to the next point of merging and asks them to select operations, then performs selected operations automatically. Stumme and Maedche (2001) argue that previous approaches do not offer a structural description of the global merging process, and propose a new bottom-up method FCA-MERGE for merging ontologies. It follows three steps: linguistic analysis of the text returning two formal contexts; merging two contexts; and semi-automatic ontology creation. According to the survey by Flouris et al. (2008), since all current tools for ontology linking and merging are manual or semi-automatic, it is still a research challenge to explore to what extent the merging process can be automated and to try to automate this process. The most recent work on ontology merging is by Robin and Uma (2010), who proposed a novel algorithm for fully automated ontology merging using a hybrid strategy, which consists of Lexical Matching, Semantic Matching, Similarity Check, and Heuristics Functions as sub-strategies. However, this work only focused on text information and did not capture the uniqueness of geospatial information.

Map matching, a fundamental research area in GIS, is developed for mapping positioning data to spatial road network data (roadway centerlines) to identify the correct link, on which a vehicle is travelling and to determine the location of a vehicle on a link (Quddus et al. 2007). Quddus et al. (2007) presents an overview of map matching algorithms and their limitations. These techniques can be adapted when integrating geospatial data. In 2010, a position map matching algorithm was developed for integrating OS ITN and OSM road vector data, and showed promising results (Anand et al. 2010). However, especially when integrating different forms of data at the feature level, these techniques are often time consuming and are more computationally intensive than other techniques available.

In our previous work, an ontology-based feature matching algorithm was developed to integrate road vector data (Du et al. 2011). It used both geometry and attribute information to find the correspondences between input data, and designed a weighted function to calculate the probability of two features being the same. Compared to geometric matching, this approach requires less computation time, and seems more efficient and effective, especially when the completeness of data is high. It showed promising results for an Ordnance Survey and OpenStreetMap case study, with more than 90% of roads matched in experiments (Du et al. 2011). However, though it is based on ontology, no formal ontologies were generated. More research on automated ontology merging is needed in the geospatial field to fill the current gap.

3 Design

The methodology is based on ontology, which refers to a logical conceptual framework for the representation of information in a particular domain. The OWL 2 Web Ontology Language (OWL 2), a W3C standard web ontology language, is used to represent ontologies to which input road vector data sets are converted. OWL 2 has been developed as one of the standard formats to facilitate information sharing and integration (W3C 2009). By adding more vocabulary for describing properties and class, it facilitates greater machine interpretability of Web content than XML, RDF and RDF-Schema (W3C 2009). An OWL 2 ontology usually has an ontology Internationalized Resource Identifier (IRI). IRI is a new protocol element and a complement to URI (The Internet Society 2005). An IRI is made up of characters from the Universal Character Set (Unicode/ISO10646). Compared to URI, IRI supports not only languages using alphabet but also other languages. Pellet, a theorem prover and OWL reasoner, is used to check whether an ontology is consistent (Clark and Parsia LLC 2011).

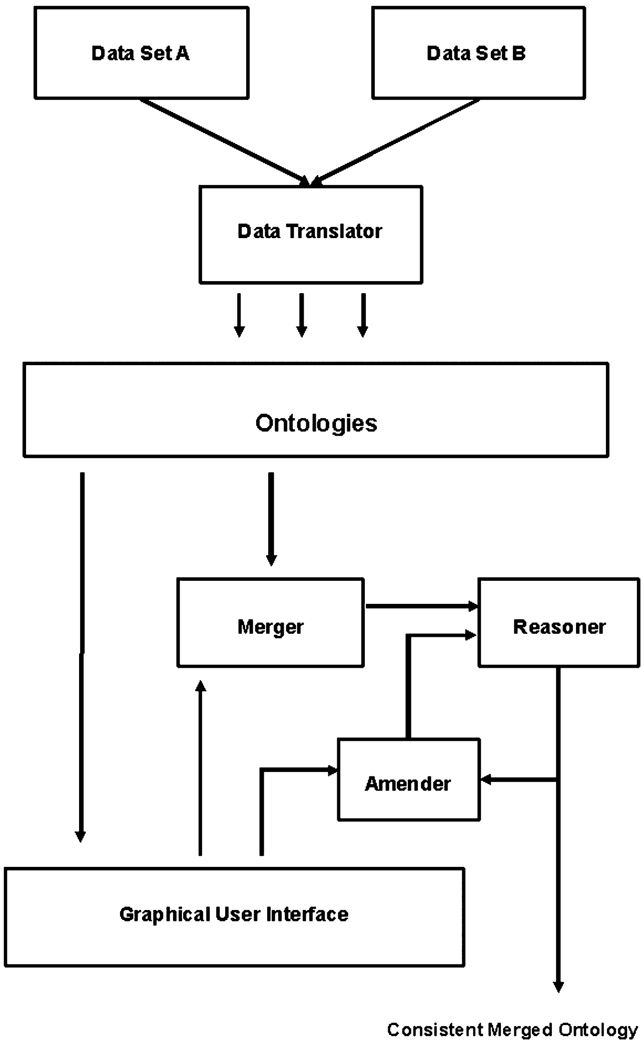


Figure 1 Architecture Design

3.1 System Architecture

The system is designed as shown in Figure 1. Each rectangle represents a component and each arrow shows the direction of data flow. The system firstly translates input road vector data from each set into an ontology, and then merges ontologies into a new one, which is checked and validated to ensure that it is logically consistent. The role of each main system component, such as data translator, merger, reasoner, amender, and graphical user interface, are explained below.

Data translator takes a geospatial data file as input, converts the input data into an ontology. Given data from disparate geospatial **data sets**, e.g. Ordnance Survey Integrated Transport Network (OS ITN) and OpenStreetMap, the translator can transform each based on a defined model (e.g. graph model), resulting in corresponding **ontologies**.

To deal with road vector data, the graph model is employed, so that the road network is represented as a graph made of edges and vertices. **Merger** takes different ontologies as input, and generates a new ontology which combines information from the input ontologies. **Amender** takes an inconsistent ontology as input, and allows users to select a strategy to fix inconsistencies. Currently there are three basic strategies or operators: union, selection, and regeneration. Union will combine two geometries into one. Selection will select the geometry with a higher degree of belief. Regeneration will generate a new geometry based on these two original geometries and their corresponding degrees of belief. **Reasoner** takes the newly generated ontology from Merger or the ontology from Amender as input, and checks whether it is logically consistent or not. Logical consistency means that from the given ontology axioms (statements which say for example that every individual has exactly one topology) and facts about individuals, a contradiction is not logically derivable. If the ontology is not consistent, Amender will be activated. Inconsistency here refers to conflicts or unsolvable differences when referring to the same property of an object. Because of the structural difference between the studied data sets, it is difficult to find a same attribute, except for the road name. Thus, this project focuses on dealing with topological inconsistency and geometry inconsistency, the definitions of which will be given in the following sections. **Graphical user interface** is responsible for representing ontologies in a way that users can read and understand easily. It also passes commands and user-based information to Merger and Amender.

3.2 Ontology Design

To deal with the road vector data, the graph model is applied when building the top level ontology, assuming that that input data sets share a common name attribute and are at the same scale. A graph is made up of a collection of vertices and a collection of edges that connect pairs of vertices. A road network, no matter how complex it is, can be simplified as a graph, with each road as an edge, and the end points of a road as vertices. It is important to note that the edges may not be straight lines. By employing this model, it is possible to capture the basic relationships between different roads. For example, if two roads share the same end point with a pre-determined tolerance level, they are connected. Following the graph model, there are two basic classes in designed OWL 2 ontologies, class “Edge” and class “Vertex”. To express the relationships among edge individuals and vertex individuals, two object properties are specified, “hasVertex” and “IsVertexOf”. So the end points of a known road can be found easily, as can the roads to which a given end point belongs.

Having embedded the graph model in the ontology, the next step is to flow the input data (as facts) from different data sets (e.g. OS ITN and OSM) into this model. Named individuals are created, with information about each of them being stored as corresponding data properties that are determined by the feature schema of input datasets. The name of an individual can be seen as its identifier, and used as the link helping to find correspondence between different input datasets. Currently, road names, though they may not be unique keys in input datasets, are used as individual names for edges, since it is a simple and reasonable way to start and can effectively tackle the problem. So when a road name exists in at least one input data set, the information about the same individual is found. Vertices in this context are actually geometries of extracted end points of roads, so input data does not include any attribute information about them. For vertices, rounded geometries are used as the individual names, in order to reduce the interference from small geometric differences that often exist among different geospatial information sources. For example, the coordinate of a vertex in one data set is (100001, 100005), while in the other it is (100002, 100006). Only after rounding the last digit will the coordinate be (100000, 100010) in both data sets, so the system can recognize that the two different coordinates actually describe the same vertex.

3.3 Consistency Definitions

Based on the graph model, topological and geometry consistency are defined for the reasoner as follows. The reasoner involves two parts: Pellet for checking topological consistency, and a self-defined part for checking geometry consistency.

3.3.1 Topological Consistency

A new functional data property is generated to store all the neighbours of an edge. Two edges are neighbours if they have the same vertex. Neighbour set, or adjacency list, is one of the two standard ways of storing neighbours of a node (the other one is adjacency matrix). In sparse graphs with few neighbours, adjacency list representation is more efficient than adjacency matrix (Goodrich and Tamassia 2006). This data property is functional, namely for each edge, there can be at most one distinct sorted literal representing its neighbour set. In other words, if two input ontologies have different neighbour sets for the same edge, topological inconsistency exists. Pellet is used to discover topological inconsistency. To fix it, the neighbour set which is generated from the more authoritative data set is selected for the final output.

3.3.2 Geometry Consistency

We could have declared the geometry of a line to be a functional property, too, and used Pellet to check that the same lines have exactly the same geometry in both ontologies. However, this would be too restrictive. Instead, we use the following definition of geometry consistency.

If every point in each line is within the fuzzy distance from the other line, then the geometries of these two lines are consistent. When it is consistent, to ensure that the final output has one geometry for each individual, the data from the more authoritative dataset is retained. Otherwise, users can apply one of the strategies (union, selection and regeneration).

Definition 1

```
Definition: equals(Line g1, Line g2, double fuzzy)
1   Line g1, g2;
    // Compute buffer areas around g1, g2, having the given fuzzy as width.
2   Polygon bg1= g1.buffer(fuzzy), bg2=g2.buffer(fuzzy);
    // See whether the buffer area of a line covers the other line and vice versa.
3   return bg1.covers(g2)&&bg2.covers(g1);
```

Table 1 Shapefile as a table

	Field 1	Field 2	Field 3	Field 4	. . . . . .
Individual 1					
Individual 2					
Individual 3					
. . . . . .					

3.4 Algorithm Design

This section will explain the algorithms designed for translator, merger, and amender.

ALGORITHM TRANSLATE specifies how to translate a given input file into an OWL 2 ontology. To generate an ontology in OWL, there is a need firstly to specify an ontology IRI, which can be seen as the identifier of the ontology. Following the graph model, two basic classes, "Edge" and "Vertex", are created. The input file (in this case Shapefile) stores data as a table with each row storing data about an individual on several attributes (see Table 1). Firstly, each field will be translated into a data property and added to the ontology. The table will then be read row by row. In each row, data in the identifier field will be used to generate an individual of class Edge (its end points will be translated into individuals of class Vertex), while data in other fields will be added to this individual as information of corresponding data properties, including geometry, timestamps and so on. Finally, the newly generated ontology will be returned.

ALGORITHM TRANSLATE: translate (Shapefile file, IRI ontologyIRI)

```
1  Data table ← file.getData ();
2  Ontology ontology ← new Ontology (ontologyIRI)
   // There are two basic concepts, Edge and Vertex, in the ontology.
3  ontology.createEdgeClass ();
4  ontology.createVertexClass ();
5  Vector < DataProperty > dataProperty ← new Vector < DataProperty > ()
   //Convert attributes to data properties in the ontology. Find identifier and
   timestamp fields.
6  FOR ( int i=0; i< table.getNumFields (); i++ )
7  Value fieldname ← table.getFieldName (i);
8  DataProperty dp← ontology.addDataProperty (fieldname);
9  dataProperty.add (dp);
10  record_idfield ();
11  record_timefield ();
12  ENDFOR
    // Convert each road to an instance of Edge, and end points to instances of Vertex.
13  FOR ( int x=0; x< table.getNumRows (); x++)
14  Value id ← table.getValueAt (x, get_idfield ())
15  Individual edge ←ontology.addEdge (id);
16  ontology.addVertexfromEdge (edge);
    // Add geometry to each individual.
```

```
17     Geometry geometry ← getGeometry (x);
18     ontology.addGeometryToIndividual (edge, geometry)
     // Add timestamp to each individual.
19     Value time ←table.getValueAt (x, get_timefield ())
20     ontology.addTimeStampToIndividual (edge, time);
     //Add all other data to corresponding data properties of each individual
21     FOR ( int y; y< table.getNumField (); y++ )
22       Value value ← table.getValueAt (x, y);
23       ontology.addDataToIndividual (edge, dataProperty.get (y), value)
24     ENDFOR
25   ENDFOR
26   RETURN ontology;
```

ALGORITHM MERGE describes the process of merging different ontologies into one. It takes two parameters, a collection of OWL ontologies, and user inputs (such as merge information about an individual or all individuals, which attributes a user wants to include into the merged ontology, whether a user wants to merge geometric information). Firstly, a new OWL ontology will be initialized with its own ontology IRI. Given a name of an individual, the algorithm will find information about specified data properties about individuals with the same name in given ontologies, then transform and add this information into the new ontology. Given the key word "ALL", the algorithm will process every individual (e.g. every road) in the input ontologies, and add information to the new ontology. Following this algorithm, a new ontology will be generated using information from the different ontologies.

ALGORITHM MERGE: merge (Vector <Ontology> ontologies, UserInput input)

```
1   Ontology newOntology ← new Ontology (input.getOntologyIRI);
2   FOR (Ontology ontology: ontologies)
3     Value clue ← input.getClue();
4     IF (clue.isALL)
        // Merge data for all individuals.
5       FOR each Individual individual in ontology
6         process (individual, ontology);
7       ENDFOR
8     ELSE
        // Merge data for one particular individual that the user specified.
9       Individual individual ← ontology.getIndividual (clue);
10      process (individual, ontology);
11    ENDIF
12  ENDFOR
13  RETURN newOntology;
14  process (Individual individual, Ontology ontology)
      // Create a new individual with the same name.
15    Individual newIndividual ← newOntology.createIndividual (individual);
      // Add user selected data properties and corresponding data to the new
      individual.
```

```
  FOR each DataProperty property in ontology
    IF (input.select (property))
      DataProperty newProperty←newOntology.createDataProperty (property);
      Value value ← individual.getDataPropertyValue (property, ontology);
      newOntology.addDataToIndividual (newIndividual, property, value);
    ENDIF
  ENDFOR
```

ALGORITHM AMEND specifies how to fix geometry inconsistencies for an individual of a merged ontology. It defines three basic operators – union, selection and regeneration, and takes the user inputs, including which operator the user wants to apply, and the degree of belief, as a parameter. Users are allowed to decide the degree of belief for different cases separately. The details of regeneration process will be explained in the next algorithm. ALGORITHM AMEND can be applied to the whole ontology by iterating over each individual.

ALGORITHM AMEND: amend (Ontology merged, Individual individual, UserInput input)

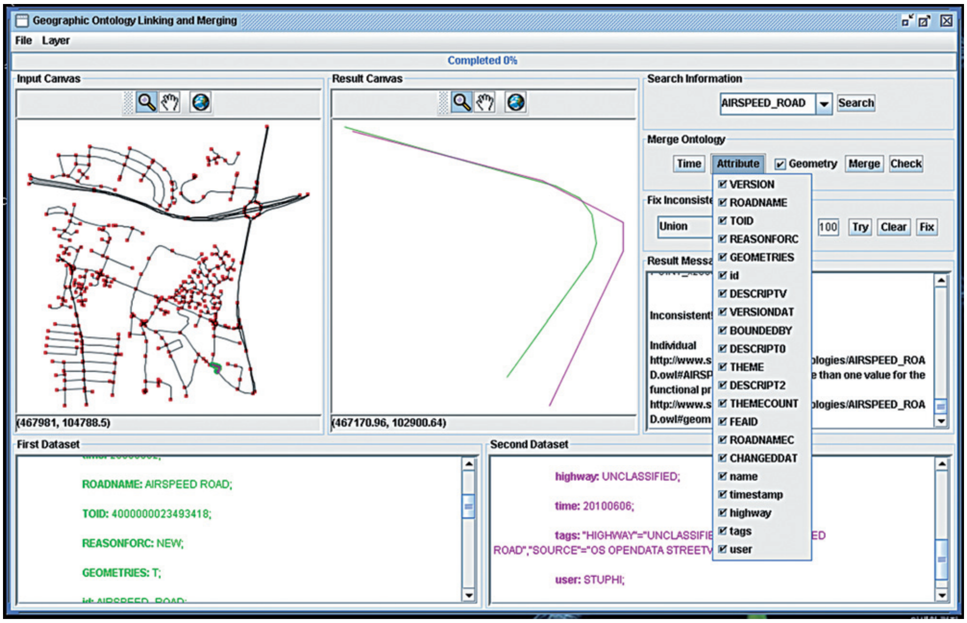
```
// This algorithm is mainly for fixing geometry inconsistency
int operator ← input.getOperator ();
// degree of belief assigned to the first data set
int belief_fst ← input.getBelief();
int belief_snd ← 100 – belief_fst;
//Get individual geometries, which are from different original sources.
Geometry geo_fst ← merged.getFstGeo (individual);
Geometry geo_snd ← merged.getSndGeo (individual);
Geometry geometry;
SWITCH (operator)
  // Union–combine two geometries into one.
  CASE union:
    geometry← geo_fst.union (geo_snd);
    break;
  // Selection–select the geometry with higher degree of belief.
  CASE selection:
    IF (belief_snd > belief.fst);
      geometry←geo_snd;
    ELSE
      geometry ← geo_fst;
    ENDIF
    break;
  // Regeneration–generate a new geometry based on these two original
     geometries and their corresponding degrees of belief.
  CASE regeneration:
    geometry ← regenerate (geo_fst, geo_snd, belief_fst, belief_snd,
    input.getFuzzy());
    break;
```

```
END
// Replace old geometry using the new geometry
merged.reassignGeometry (individual, geometry);
RETURN merged;
```

ALGORITHM REGENERATE specifies how to generate a new geometry from two corresponding input geometries based on corresponding degrees of belief and a level of fuzzy tolerance. Firstly, the geometry is initialized by the input geometry with the higher degree of belief (higher weight). Then the algorithm will try to modify this geometry taking the lower weighted input geometry into account. The modification process is defined as the following. Firstly, we get all the coordinates of the initial geometry (g). For each coordinate c1, test whether there exist coordinates which are equal to it within the given level of fuzzy tolerance in both input geometries. If it exists, the algorithm will find the nearest coordinate c2 in the lower weighted geometry. Then a new coordinate c3 will be generated by computing the weighted average of coordinates c1 and c2, and replace c1 in output geometry (g). If not, c1 does not change. A smooth operator will do some angle checking to ensure the amendments do not generate strange shapes. When a new point is generated, it will form new sub-lines with its adjacent points. The smooth operator checks whether the slopes of these new sub-lines will form a sharp angle. If yes, this new point will not be included. The algorithm returns a newly generated geometry.

ALGORITHM REGENERATE: regeneration (Geometry geo_fst, Geometry geo_snd, int fstwt, int sndwt, int fuzzy)

```
/* select the basic geometry from these two geometries available for the individual,
depending on input weight.*/
Geometry geometry;
boolean b ← true;
IF (sndwt > fstwt);
   geometry←geo_snd;
   b←false
ELSE
   geometry ← geo_fst;
ENDIF
Coordinate[] coordinates← geometry.getCoordinates() ;
FOR each coordinate in coordinates
   // exist in both datasets given a fuzzy tolerance
   IF existInBothGeometry(coordinate, geo_fst, geo_snd, fuzzy)
      Coordinate fstcoord, sndcoord;
      IF (b)
         fstcoord← coordinate;
         sndcoord← getNearest(geo_snd, coordinate);
      ELSE
         fstcoord← getNearest(geo_fst, coordinate)
         sndcoord← coordinate;
      ENDIF
      //Calculate the new coordinate.
```



Inconsistent

Individual
http://www.s
D.owl#AIRSP
functional pr
http://www.s
D.owl#geom

ologies/AIRSPEED_ROA
e than one value for the
ologies/AIRSPEED_ROA

ROADNAME: AIRSPEED ROAD;

TOID: 4000000023493418;

REASONFORC: NEW;

GEOMETRIES: T;

highway: UNCLASSIFIED;

time: 20100606;

tags: "HIGHWAY"="UNCLASSIFIE ED
ROAD","SOURCE"="OS OPENDATA STREETV

user: STUPHI;

Figure 2 Graphical User Interface

```
20          coordinate ← (fstcoord*fstwt + sndcoord*sndwt)/(fstwt + sndwt) ;
            //Replace the old coordinate with the new one.
21          geometry.update(coordinate) ;
22      ENDIF
23    ENDFOR
      // angle checking to ensure the amendments do not generate strange shapes
24    geometry.smooth();
25    RETURN geometry;
```

3.5 User Interface Design

The designed graphical user interface is shown in Figure 2. Most of the space in this interface is used to visualize information. One of the main differences between geospatial information and most other kinds of data is that it has a geometry component. The geometric information extracted from the input file can be output in Well-known text (wkt) format, developed by the Open Geospatial Consortium (OGC). Taking Adstone Lane in the city of Portsmouth, UK, for example, its geometry in wkt format in the Ordnance Survey ITN data set is as follows:

> LINESTRING (467049 103638, 467051 103642, 467053 103650, 467054 103668, 467054 103671, 467056 103673, 467058 103675, 467061 103677, 467070 103680, 467071 103681, 467083 103694);

Information like this is difficult to read and understand, and does not even make much sense, especially to users with limited geospatial knowledge. Thus, there are two canvases, which can capture users' attention easily, to visualize geometry as lines and points using different colours. The JUMP Unified Mapping Platform (Vivid Solutions 2010) provides the layer view panel together with three tools – Zoom In, Pan, and Zoom to Full Extent, which are important components of the canvas. The JUMP Unified Mapping Platform is an open source application for viewing, editing and processing geospatial data (Vivid Solutions 2010). The left canvas (Input Canvas) is designed for input visualization; while the right one (Result Canvas) is for showing generated results (e.g. searched geometry, merged geometry, etc.). Below them are two data boards (First Dataset and Second Dataset), which show information in text from the first source and second source, respectively.

4 Implementation

The prototype is implemented in Java. To create and interact with ontology written in OWL 2, the OWL 2 API, a Java library developed by the University of Manchester, is used (Sourceforge 2011). The system implements the following functionalities: data is converted from a geospatial information format Shp to ontology in OWL 2, and ontologies are merged in accordance with user specific operations.

4.1 Translating Data and Creating Ontology

4.1.1 Shp2OWL

The Shp2OWL class acts not only as a Shp file reader, but also translator. It knows the structure of the Shp file, and how to extract information (e.g. schema) from it. It passes information to its OntologyBuilder, and this knows how to use it to build a new OWL ontology. The Shp2OWL class has a method read() that implemented the ALGORITHM TRANSLATE. When reading a shapefile, the OntologyBuilder will add information to the ontology in OWL 2.

4.1.2 OntologyBuilder

The OntologyBuilder class is responsible for creating a new ontology (OWLOntology) and storing it in OWL 2 ontology. It keeps track of the current ontology it is working on. OntologyBuilder is defined based on several classes of the OWL 2 API. OWLOntology-Manager and OWLDataFactory are the main ones. When adding new information to the ontology, OWLDataFactory is used to generate appropriate Axioms, referring to statements which say what is true in the domain (W3C 2009). OWLOntologyManager can add Axioms and save them into the ontology.

4.2 Generating Merged Consistent Ontology

This component consists of four parts. The *Merger* class extracts user selected information about an individual (e.g. an edge or a vertex) or all the information from both data sources, and passes it to its OntologyBuilder, which will store it in a newly created ontology. The ALGORITHM MERGE is implemented in the Java class called Merger. The *Checker (Reasoner)* class checks the topological and geometry consistency. It creates a Pellet Reasoner for a given OWL ontology. Pellet Reasoner can check whether the ontology is topologically consistent or not, by checking the consistency of its knowledge base which is generated from the ontology, and explain reasons for any inconsistencies. A knowledge base in this case is the set of ontology axioms (general statements such as uniqueness of topology) and facts (statements about individuals). The *Remover* class can delete all the information about an OWL entity (e.g. a class, an individual, a data property, or an object property). Before adding consistent information (e.g. geometry or neighbour) into the merged ontology, inconsistent information needs to be removed from the ontology. The *Amender* class implements the ALGORITHM AMEND, and defines the three operators to deal with geometry inconsistency during merging. Each of them is implemented as methods both for an individual and an ontology.

- Union: Combine two different geometries of an individual into one that contains every point of both.
- Selection: Select one from two different geometries for an individual, based on a user's degrees of belief (DoB) on data sources. For example, if the DoB on the first source is higher, geometry from the first data source will be selected.
- Regeneration: Find corresponding points of two different geometries, then, compute intermediate coordinates for these points based on the DoB of the user. The final geometry is based on the input geometry with higher DoB and goes through all the generated intermediate points. See the ALGORITHM REGENERATE for details.

It is necessary to note that when implementing the ALGORITHM AMEND, there is a need to refer to the original two ontologies. This is because in the merged ontology, information is stored in the same way, thus it is not possible to identify the source of a particular piece of information. In addition, since information in an OWL 2 ontology is stored in Java *Set*, the order of stored information cannot be relied upon to know its source, even if information from the first data set is stored before that from the second. To amend the geometry inconsistencies of the merged ontology, it is necessary to go back to redo the merging for geometry in the way the user specifies.

5 System Evaluation

Black box testing methodology was used to check whether or not the system met the functional requirements. Black box testing includes methods for generating test cases that are independent of the software's internal structure (Ostrand 2002). It is also called specification-based or functional testing, since black box testing is based on the function of the software, rather than its structure and design. Ordnance Survey's Integrated Transport Network (ITN) road data and OpenStreetMap (OSM) road data for Portsmouth, UK were used as imports to test the software. (A "Singleparts_to_Multipart" operation was applied to the data first using QGIS to specify the name fields as identifier.) Protégé, an ontology editor, is used to open exported OWL ontologies (Stanford Center for Biochemical Informatics Research 2011). Functionality testing confirmed that the system is implemented correctly. The process of testing and its results are summarized in Table 2.

In order to evaluate the algorithms, we compared the results of applying the algorithms on the test data with the desired results as determined by expert human users.

We concentrated in particular on the ALGORITHM REGENERATE, since other algorithms (e.g. ALGORITHM TRANSLATE and ALGORITHM MERGE) can be examined by simply looking at the output ontologies using Protégé, or the correctness of

Table 2 Test Sheet

System Test Sheet	
Project Title	Geospatial Ontology Merging from Disparate Sources
Objective	To test whether the system meets the functional requirements.
Method	Black Box testing
Test Environment	CPU: Intel (R) Core (TM) 2 Duo CPU T5750 @ 2.00GHz RAM: 3.00 GB Operating System: Microsoft Windows XP Professional 2002

Test Process

Requirement to test	Action	Expected Result	Y/N
The system should allow users to import new geospatial data in Shapefile format.	File→Import→OS_idn1.shp	Data is displayed in Input Canvas;	Y
		Two items (OS_idn1.owl1 and OS_idn1.owl2) are added to Menu Layer.	Y
		The JComboBox in Search Section contains a list of names of all the individual from the OS.idn1 ontology	Y
Each data set should have its OWL ontology. Export OWL ontology	File → Export→OS.owl Open OS.owl using Protégé	A file named as OS.owl will be generated. It contains the OWL ontology translated from OS_idn1.shp	Y
The system should be able to load information from disparate data sources.	File→Import→OSM_id.shp	Data is displayed in Input Canvas;	Y
		Two items (OSM_id.owl1 and OSM_id.owl2) are added to Menu Layer.	Y
		The JComboBox in Search Section also contains a list of names of all the individual from the OSM.idn ontology	Y
Each data set should have its owl OWL ontology. Export OWL ontology.	File→ Export→OSM.owl Open OSM.owl using Protégé	A file named as OSM.owl will be generated. It contains the OWL ontology translated from OSM_id.shp	Y

Table 2 Continued

Test Process			
Requirement to test	**Action**	**Expected Result**	**Y/N**
The system should allow users to search for information of interest. The system should be able to show required original information from both sources to users.	In Search Information section, select “ADSTONE_LANE”, then press Button Search	The geometries of “ADSTONE_LANE” will be shown in Result Canvas in different colors.	Y
		The geometries of “ADSTONE_LANE” will be highlighted in Input Canvas in different colors.	Y
		First Dataset board will display all the text information of “ADSTONE_LANE” from the first ontology.	Y
		Second Dataset board will display all the text information of “ADSTONE_LANE” from the second ontology.	Y
	In Search Information section, select “POINT_x23300y 5183zNaN”, then press Button Search	The geometries of “POINT_x23300y5183zNaN” will be highlighted in Result Canvas in red.	Y
		The geometries of the edge (MERLIN_DRIVE) it belongs to will be shown in Result Canvas in different colors.	
		The geometries of “MERLIN_DRIVE” will be highlighted in Input Canvas in different colors.	Y
		First Dataset board will display all the text information of “POINT_x23300y5183zNaN” from the first ontology.	Y
		Second Dataset board will display all the text information of “POINT_x23300y5183zNaN” from the second ontology.	Y
	In Search Information section, enter “all”, then press Button Search	The geometries of all the individuals will be shown in Result Canvas in different colors indicating which ontology they are from.	Y
		First Dataset board will display all the text information of every individual from the first ontology.	Y
		Second Dataset board will display all the text information of every individual from the second ontology.	Y
	In Search Information section, enter nothing, then press Button Search	Result Canvas, First Dataset board and Second Dataset board are all blank.	Y
The system should allow users to check information from which source is more recent.	After searching “AIRSPEED_ROAD” for example, press Button Time in Merge Ontology Section,	A message pops up reporting timestamp information about “AIRSPEED_ROAD”.	Y
The system should allow users to select the information attributes they want, and merge selected information. The system should be able to check whether merged information is consistent or not and give justifications if inconsistent.	In Merge Ontology Section, click Menu Attribute, deselect “name”. Deselect “Geometry”. Press Button Merge. Export a new ontology and open it using Protégé.	A merged ontology about “AIRSPEED_ROAD” containing selected information will be generated.	Y
		A message pops up reporting whether it is consistent. (It is consistent.)	Y
		Closing the message box, the Result Message board will display the information about “AIRSPEED_ROAD” in the merged ontology.	Y
The system should be able to show merged information to user, no matter whether it is consistent or not.	Select “Geometry”. Press Button Merge. Export a new ontology and open it using Protégé.	Similar expected results as above except that this ontology is not consistent. Result Message board will also show the reasons for inconsistency.	Y

Table 2 Continued

Test Process

Requirement to test	**Action**	**Expected Result**	**Y/N**
If there is any inconsistency, the system should be able to offer ways to help users fix it, and provide consistent information to users.	In Fix Inconsistency Section, select Union Operator, press Button Try.	Both geometries will be colored in blue in Result Canvas.	Y
	Press Button Clear. Select Selection Operator. Press Button Try.	Geometry from the first ontology will be colored blue in Result Canvas. (Because DoB=100>=50)	Y
	Press Button Clear. Enter a number less than 50, for example 40, in the DoB field. Press Button Try.	Geometry from the second ontology will be colored blue in Result Canvas. (Because DoB=40<50)	Y
	Press Button Clear. Select Amendment Operator. Press Button Try.	A new geometry colored blue will be displayed in Result Canvas. It is nearer and more similar to geometry from the second ontology, since the DoB is 40.	Y
	Enter a number greater than 50, for example 90, in the DoB field. Press Button Try.	A new geometry colored blue will be displayed in Result Canvas. It is nearer and more similar to geometry from the first ontology, since the DoB is 90.	Y
	Press Button Fix	The Result Canvas will only display the resulting geometry in red.	Y
		A message pops up reporting the ontology is consistent.	Y
		Closing the message box, the Result Message board will display the information about "AIRSPEED_ROAD" in fixed ontology.	Y
The system should allow users to export generated ontology in OWL format.	File→Export→ AIRSPEED_ROAD.owl Open AIRSPEED_ROAD.owl using Protégé	A file named as AIRSPEED_ROAD.owl will be generated. It contains the merged OWL ontology about "AIRSPEED_ROAD".	Y
Clear out after test	File → Clear All data are removed from the system.		

Conclusion

Tester	Heshan Du	Date	2011/3/1	Test Result	Pass

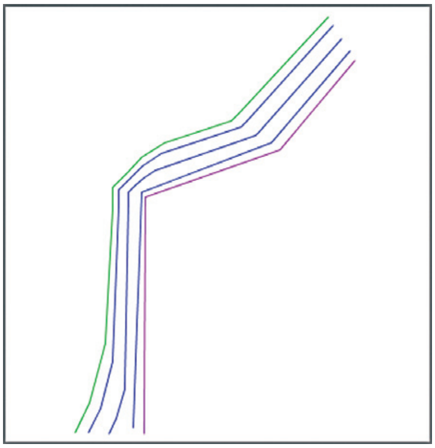


Figure 3 Adstone Lane

some parts is obvious (e.g. the union operator and selection operator in the ALGORITHM AMEND). The ALGORITHM REGENERATE specifies the behaviour of the regeneration operator in the ALGORITHM AMEND. The ALGORITHM REGENERATE is evaluated against the expected results. Given two Lines g1, g2, and a degree of belief d%, the expected new Line g3 should be between g1 and g2, and the distance (g3, g2) should be about d% of the distance (g1, g2). Below, we describe the results of evaluating the ALGORITHM REGENERATE.

As a first example, take "ADSTONE LANE"; it is representative of a large number of cases where the algorithm performed well. In Figure 3, the green line (first left) shows the geometry of "ADSTONE LANE" from the first dataset, while the purple line (first right) shows the geometry from the second dataset. The three blue lines in the middle, from left to right, show the geometries generated by the algorithm given different degrees of belief, 80, 50, and 20. This output was judged correct by the human expert users based on the distances between lines.

However, the ALGORITHM REGENERATE does not work so well for the cases where there are very few points in the original geometries that can find corresponding points. In those cases, the generated geometry will be very close to an original geometry except for little movements of one or two points. For example, when applying the algorithm to "ACKWORTH ROAD" with degree of belief being 50, the result is shown in Figure 4. The circled point is the only one that is moved into the middle when generating the new geometry. This output is almost the same as the geometry (like a "T") from the first data set, so does not constitute the desired result.

Results show that among 105 roads that have different geometries in two input data sets, the recreation algorithm can generate desirable new geometries for 92 of them, just as for "ADSTONE LANE". In other words, the ALGORITHM REGENERATE is about 85% effective for the testing data inputs.

In the remainder of this section, we discuss out design choices and limitations of the current system.

In our ontology design, the road network is simplified as a graph, with each road as an edge and end points of each road as vertices. The graph model works well for road networks, and can also be easily applied to other line features, such as railways and

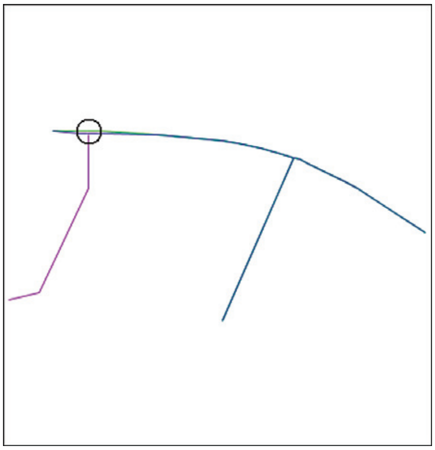


Figure 4 Ackworth Road

rivers. However, it is not suitable for polygon layers where an object (e.g. a building) is drawn as a polygon. To expand the scope of the application, we need to be able to represent polygons in our ontology.

Another design choice which is reasonable for the current scope of the project, but would need to be reconsidered for larger and more varied data sets, is the choice of road names as individual identifiers. It is a simple and reasonable way to start and can effectively tackle the problem. Of the 108 roads that exist in both input data sets, 105 have exactly the same names. However, the requirement that every input geospatial data set should have a name field to ensure every individual edge has a name is too strong. Though this requirement can be met by authoritative data sets (e.g. OS), there are several unnamed features stored in informal data sets (e.g. OSM), information from which can be incomplete. So, data about these unnamed features cannot flow into the model successfully. Even if information in name fields is complete, the system cannot recognize slightly different names as the same names. For example, a road is recorded as "GREEN FARM GARDENS" in the OS ITN data, while appearing as "GREEN FARMS GARDENS" in the OSM data. Other examples include "St" and "Saint", "road" and "close". Hence we need a more sophisticated algorithm for identifying the same feature in different data sets.

The authors also acknowledge that the feature matching approach used currently is limited in that it relies on a comparison being made between the feature name attribute of a pair of potentially matching features (e.g. FEATURE A name is "NEW ROAD", FEATURE B name is "NEW ROAD" – so there is a match). This presents a problem in situations where the feature name attribute is missing or where feature names are recorded differently (e.g. "NEW ROAD" and "A465"). In such situations an alternative approach might be to attempt to match features on the basis of their geometry. Geometric matching of points, lines and polygons has received considerable attention previously. Pioneering work in this domain was carried out by Lupien and Moreland (1987), who present a technique for both identifying and merging features from a pair of maps that represent the same real-world object. Another early work of note is that presented by Saalfeld (1988), which describes a conflation technique that achieves feature matching using both spatial (e.g. location and shape) and attribute information. There are also useful ideas to be drawn from the related problem of polygon overlay (e.g. Zhang and Tulip 1990, Chrisman et al. 1992, Harvey and Vauglin 1996) where the goal is to identify and remove sliver polygons, which can be thought of as equivalent to matching and merging. The method presented by Ware and Jones (1998) is particularly interesting since it deals with the problem of identifying the best pair of matching features from multiple pairs of possible matches (a situation that is likely to occur in areas where feature density is high). It is also noted that geometric matching which relies solely on simple distance measures has been identified as inadequate by several authors, and as such alternative approaches that use additional and alternative similarity measures have been proposed (e.g. Saalfeld 1988, Jones et al. 1996, Kundu 2006, Fu and Wu 2008, Xiaohua et al. 2009). In future work these existing approaches will be evaluated and, where appropriate, used to enhance the feature matching and merging processes of the existing system.

Our notion of inconsistency is necessarily limited in the current version of the application, where it refers to conflicts or unsolvable differences when talking about a same property of an object. Topological and geometry consistency are defined previously. However, it can be argued that inconsistency is currently defined quite narrowly since it focuses on geometry and topological relations and tells little about attribute information, which is equally important. It is possible that attribute information from different data sources do not totally agree with each other with regard to the same object, thus generating different types of inconsistency. For example, inconsistency will arise when a road is classified into Type A in one data set, while it is classified into Type B in the other, given that Type A and Type B have no intersection. Hence we need to extend the ontology with more information about the relationships between attributes expressed as ontology axioms, for example stating that Type A and Type B are disjoint. Furthermore, it can be argued that inconsistency is defined too strictly for geometry. Following this definition, inconsistency will arise even if two geometries are the same (given a fuzzy tolerance) except for the coordinates of one point. This approach needs to be relaxed to allow 'degrees of inconsistency'.

6 Conclusions and Future Work

This article describes research undertaken by the authors on the design, implementation, and evaluation of algorithms and procedures for producing a coherent ontology from disparate geospatial data sources. This article discusses the development of techniques for geographical information fusion from disparate sources for road vector data. An ontology based methodology was developed and implemented for geospatial information linking and merging. The source code developed is available at http://sourceforge.net/projects/geoontomerging/files/ for the benefit of anyone interested. The developed methodology can deal with topological and geometry inconsistency and provide more flexibility for geospatial information merging. The results are promising but more work needs to be done in refining the process of linking information and in inconsistency resolution. National Mapping Agencies and other geospatial users may benefit immensely from such developments but research is needed to understand how to tap into this huge potential opportunity and to obtain a consistent, quality and verifiable product from the data so acquired within the terms of use of the crowd-sourced data. Future work will concentrate on developing more robust and sound strategies for inconsistency resolution to solve different real world problems in other domains.

Acknowledgements

We thank the Ordnance Survey for funding Suchith Anand's research through the Future Data project.

References

Anand S, Morley J, Jiang W, Du H, Hart G, and Jackson M J 2010 When worlds collide: Combining Ordnance Survey and OSM data. *In Proceedings of the AGI GeoCommunity '10 Conference*, Stratford-upon-Avon, United Kingdom (available at http://www.agi.org.uk/storage/geocommunity/papers/SucithAnand.pdf)

Bundy A and McNeill F 2006 Representation as a fluent: An AI challenge for the next half century. *IEEE Intelligent Systems* 2006: 85–87

Chalupsky H 2000 Ontomorph: A translation system for symbolic knowledge. In *Proceedings of the Seventh International Conference on the Principles of Knowledge Representation and Reasoning (*KR-*2000*), Breckenridge, Colorado: 471–82 (available at http://ai.isi.edu/pubs/papers/chalupsky2000ontomorph.pdf).

Choi N, Il-Yeol S, and Han H 2006 A survey on ontology mapping. *ACM SIGMOD Record* 35(3): 34–41

Chrisman N R, Dougenik J A, and White D 1992 Lessons for the design of polygon overlay processing from the Odyssey Whirlpool algorithm. In *Proceedings of the Fifth International Symposium on Spatial Data Handling*, Charleston, South Carolina: 401–10

Clark and Parsia LLC 2011 *Pellet: OWL 2 Reasoner for Java* [online]. WWW document, http://clarkparsia.com/pellet

Du H, Jiang W, Anand S, Morley J, Hart G, and Jackson M J 2011 Ontology-based approach for geospatial data integration. In *Proceedings of the International Cartography Conference*, Paris, France

European Commission INSPIRE 2011 *European Commission INSPIRE*. WWW document, http://inspire.jrc.ec.europa.eu/index.cfm

Flouris G, Manakanatas D, Kondylakis H, Plexousakis D, and Antoniou G 2008 Ontology change: Classification and survey. *Knowledge Engineering Review* 23(2): 1–29

Fu Z and Wu J 2008 Entity matching in vector spatial data. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 37(B4): 1467–72

Goodrich M and Tamassia R 2006 *Data Structures and Algorithms in Java* (Fourth Edition). New York, John Wiley and Sons

Harvey F and Vauglin F 1996 Geometric match processing: Applying multiple tolerances. In *Proceedings of the Seventh International Symposium on Spatial Data Handling*, Delft, The Netherlands: 13–29

Jackson M J, Rahemtulla H A, and Morley J 2010 The synergistic use of authenticated and crowd-sourced data for emergency response. In *Proceedings of the Second International Workshop on Validation of Geo-Information Products for Crisis Management* (VALgEO), Ispra, Italy: 91–99

Jones C B, Kidner D, Luo L, Bundy G, and Ware J M 1996 Database design for a multi-scale spatial information system. *International Journal of Geographical Information Systems* 10: 901–20

Kundu S 2006 Conflating two polygonal lines. *Pattern Recognition* 39: 363–72

Lupien A E and Moreland W H 1987 A general approach to map conflation. In *Proceedings of AutoCarto 8*, Baltimore, Maryland: 630–39

McGuinness DL, Fikes R, Rice J, and Wilder S 2000 An environment for merging and testing large ontologies. *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*. Breckenridge, Colorado. April 12–15, 2000

Noy N and Musen M 1999 SMART: Automated support for ontology merging and alignment. In *Proceedings of the Twelfth Banff Workshop on Knowledge Acquisition, Modeling, and Management*, Banff, Alberta

Noy N and Musen M 2000 PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of AAAI-2000*, Austin, Texas: 450–55

Noy N and Musen M 2001 Anchor-PROMPT: Using non-local context for semantic matching. In *Proceedings of the Workshop on Ontologies and Information Sharing at the International Joint Conference on Artificial Intelligence (IJCAI '01)*, Seattle, Washington

OpenStreetMap 2011 OpenStreetMap. WWW document, http://www.openstreetmap.org/

Ordnance Survey 2011. Ordnance Survey Map. WWW document, http://leisure.ordnancesurvey.co.uk/

Ostrand T 2002 Black-box testing. Marciniak J J (ed) *Encyclopedia of Software Engineering*. New York, John Wiley and Sons

Quddus M, Ochieng W, and Noland R 2007 Current map-matching algorithms for transport applications: State-of-the-art and future research directions. *Transportation Research Part C* 15: 312–28

Robin C and Uma G 2010 A novel algorithm for fully automated ontology merging using hybrid strategy. *European Journal of Scientific Research* 47: 74–81

Saalfeld A 1988 Conflation: Automated map compilation. *International Journal of Geographical Information Systems* 2: 217–28

Sourceforge 2011 *The OWL API*. WWW document, http://owlapi.sourceforge.net/index.html

Sowa J 2000 *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Pacific Grove, CA, Brooks Cole

Stanford Center for Biomedical Informatics Research 2011 *Protégé*. WWW document, http://protege.stanford.edu/

Stumme G and Maedche A 2001 FCA-Merge: Bottom-up merging of ontologies. In *Proceedings of the Seventh International Conference on Artificial Intelligence (IJCAI '01)*, Seattle, Washington: 225–30

The Internet Society 2005 Internationalized Resource Identifiers (IRIs). WWW document, http://www.ietf.org/rfc/rfc3987.txt

Vivid Solutions 2010 JUMP Unified Mapping Platform. WWW document, http://www.vividsolutions.com/jump

Ware J M and Jones C B 1998 Matching and aligning features in overlaid coverages. In *Proceedings of the Sixth ACM International Symposium on Advances in GIS*, Washington, D.C.: 28–33

W3C 2009 OWL 2 Web Ontology Language Overview. WWW document, http://www.w3.org/TR/owl2-syntax

Xiaohua T, Shib X W, and Denga S 2009 A probability-based multi-measure feature matching method in map conflation. *International Journal of Remote Sensing* 30: 5453–72

Zhang G and Tulip J 1990 An algorithm for the avoidance of sliver polygons and clusters of points in spatial overlay. In *Proceedings of Fourth International Symposium on Spatial Data Handling*, Zurich, Switzerland: 141–50