

A Knowledge-based Approach for Creating Detailed  
Landscape Representations by Fusing GIS Data Collections  
with Associated Uncertainty

Pedro Maroun Eid

A Thesis  
in  
The Department  
of  
Computer Science & Software Engineering

Presented in Partial Fulfillment of the Requirements  
For the Degree of Doctor of Philosophy  
Concordia University  
Montréal, Québec, Canada

January 2014  
©Pedro Maroun Eid, 2014

**CONCORDIA UNIVERSITY**  
**SCHOOL OF GRADUATE STUDIES**

This is to certify that the thesis prepared

By: Pedro Maroun Eid

Entitled: A Knowledge-based Approach for Creating Detailed Landscape Representations by Fusing GIS Data Collections with Associated Uncertainty

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Chair

---

Dr. M. Mehmet Ali

External Examiner

---

Dr. V. Bhavsar

External to Program

---

Dr. R. Ganesan

Examiner

---

Dr. V. Haarslev

Examiner

---

Dr. T. Fevens

Thesis Supervisor

---

Dr. S. Mudur

Approved by:

---

Dr. V. Haarslev , Graduate Program Director

---

June 11, 2014

---

Dr. C. Trueman, Interim Dean

---

Faculty of Engineering and Computer Science

# ***ABSTRACT***

**A Knowledge-based Approach for Creating Detailed Landscape Representations by Fusing GIS Data Collections with Associated Uncertainty**

**Pedro Maroun Eid, Ph.D.**

**Concordia University, 2014**

Geographic Information Systems (GIS) data for a region is of different types and collected from different sources, such as aerial digitized color imagery, elevation data consisting of terrain height at different points in that region, and feature data consisting of geometric information and properties about entities above/below the ground in that region. Merging GIS data and understanding the real world information present explicitly or implicitly in that data is a challenging task. This is often done manually by domain experts because of their superior capability to efficiently recognize patterns, combine, reason, and relate information. When a detailed digital representation of the region is to be created, domain experts are required to make best-guess decisions about each object. For example, a human would create representations of entities by collectively looking at the data layers, noting even elements that are not visible, like a covered overpass or underwater tunnel of a certain width and length. Such detailed representations are needed for use by processes like visualization or 3D modeling in applications used by military, simulation, earth sciences and gaming communities. Many of these applications are increasingly using digitally synthesized visuals and require detailed digital 3D representations to be generated quickly after acquiring the necessary initial data.

Our main thesis, and a significant research contribution of this work, is that this task of creating detailed representations can be automated to a very large extent using a methodology which first fuses all Geographic Information System (GIS) data sources available into knowledge base (KB) assertions (instances) representing real world objects using a subprocess called GIS2KB. Then using reasoning, implicit information is inferred to define detailed 3D entity representations using a geometry definition engine called

KB2Scene. Semantic Web is used as the semantic inferencing system and is extended with a data extraction framework. This framework enables the extraction of implicit property information using data and image analysis techniques. The data extraction framework supports extraction of spatial relationship values and attribution of uncertainties to inferred details. Uncertainty is recorded per property and used under Zadeh fuzzy semantics to compute a resulting uncertainty for inferred assertional axioms. This is achieved by another major contribution of our research, a unique extension of the KB ABox Realization service using KB explanation services.

Previous semantics based research in this domain has concentrated more on improving represented details through the addition of artifacts like lights, signage, crosswalks, etc. Previous attempts regarding uncertainty in assertions use a modified reasoner expressivity and calculus. Our work differs in that separating formal knowledge from data processing allows fusion of different heterogeneous data sources which share the same context. Imprecision is modeled through uncertainty on assertions without defining a new expressivity as long as KB explanation services are available for the used expressivity. We also believe that in our use case, this simplifies uncertainty calculations. The uncertainties are then available for user-decision at output. We show that the process of creating 3D visuals from GIS data sources can be more automated, modular, verifiable, and the knowledge base instances available for other applications to use as part of a common knowledge base. We define our method's components, discuss advantages and limitations, and show sample results for the transportation domain.

## ***Acknowledgements***

First and foremost, I sincerely thank my supervisor, Prof. Sudhir P. Mudur, for his patience and support throughout the completion of this thesis. I would not have asked for a more experienced and thorough supervisor. His help and supervision allowed me to learn tremendously throughout the course of my Ph.D. I would also like to thank my supervisory committee, Prof. Thomas Fevens, Prof. Rajamohan Ganesan, Prof. Mustafa Mehmet Ali, and especially Prof. Volker Haarslev and Prof. Virendrakumar Bhavsar for insightful comments and support.

I also thank members of Concordia University and the Department of Computer Science for the continuous financial and moral support during the course of this research.

This research was supported by GRAND NCE and NSERC as well as Engineering and Computer Science Faculty Research Grants and the Canada Foundation of Innovation which supported equipment acquisition. This work was conducted using the Protégé resource, which is supported by grant GM10331601 from the National Institute of General Medical Sciences of the US National Institutes of Health. We thank OSGeo, the Open Source Geospatial Foundation, and its primary supporter, Ordnance Survey®, for providing open source software to allow the implementation of the prototype used in this research. We also would like to thank Google Inc. for providing software support, as well as Presagis and DVC for their software which allowed a broad investigation of available capabilities and needs. We include ESRI, the OGC, USGS and other organizations for providing information and toolkits on standards used.

*"If I have seen further than others, it is by standing  
upon the shoulders of giants."*  
**Isaac Newton**

*"Do not follow where the path may lead,  
go instead where there is no path  
and leave a trail."*  
**Ralph Waldo Emerson**

*"If you can dream it, you can do it."*  
**Walt Disney**

To my father, Chawki, with all my thankfulness and appreciation.

To my mother, Josette, my brother Georges, and my sister, Honorée  
Claris. To dearest friends and family...

...this would not have been possible without your support.

# ***Table of Contents***

List of Figures .....	xi
Chapter 1 Introduction .....	1
1.1 Motivation.....	5
1.2 Problem Statement .....	6
1.3 Objectives .....	7
1.4 Methodology and Proposed Solution.....	8
1.5 Implementation .....	10
1.6 Simple Example from our System's Results .....	11
1.7 Contributions.....	14
1.7.1 Conceptual Contributions .....	14
1.7.2 Practical Contributions.....	15
1.8 Thesis Layout.....	16
Chapter 2 Background .....	18
2.1 Relevant GIS Sources .....	22
2.1.1 ESRI Shapefiles .....	25
2.2 Other Commonly used GIS Standards .....	27
2.2.1 Legacy Sources .....	28
2.2.2 OpenFlight Standard .....	29

2.2.3 Newer Sensor Technologies .....	31
2.3 Topological Relations and DE-9IM.....	32
2.4 Semantic Web .....	33
2.4.1 Reasoner Services and Justification.....	37
2.4.2 SPARQL and GeoSPARQL .....	38
2.5 Summary .....	40
Chapter 3 Related Work.....	41
3.1 On Standards.....	41
3.2 Terrain Modeling Tools .....	43
3.3 Methods in Scene Understanding .....	50
3.4 Applications of Formal Knowledge to GIS data.....	55
3.5 Uncertainty and the Semantic Web.....	60
Chapter 4 Knowledge Base and 3D Landscape Creation .....	64
4.1 Theoretical Formulation.....	67
4.1.1 Definitions.....	67
4.1.2 Assumptions.....	69
4.1.3 Basic Mapping of GIS Sources.....	71
4.1.4 Adding Relationships.....	74
4.1.5 Filling of Property Values.....	76
4.1.6 Representation Creation.....	80
4.1.7 Thoughts .....	81
4.2 Evolution of our Method.....	82
4.2.1 Legacy Processes (for comparison) .....	82
4.2.2 Attempt 1: Extracting all Semantics from Raster Data.....	84
4.2.3 Attempt 2: Linking KB Objects to Relevant Raster Data.....	85

4.2.4 Attempt 3: Running All Extractors for Every Missing Property .....	87
4.2.5 Final Solution: Running only Relevant Extractor Functions.....	88
Chapter 5 System Implementation.....	90
5.1 The Ontology Hierarchy .....	91
5.1.1 Ontology of Source Concepts Mapping (SD).....	94
5.1.2 Ontology of Parameterized Models (RC) .....	95
5.2 Main Process and Geometry Definition Engine .....	97
5.3 GIS2KB and the Data Extractors Framework .....	100
5.3.1 Initial Mapping - Shapefiles.....	103
5.3.2 Data Extractors.....	105
5.3.2.1 Transportation Object Extractors .....	106
5.3.2.2 Object Property Extractors.....	108
5.3.2.3 Data Property Extractors.....	109
5.3.3 Adding Object-Object Relationships .....	115
5.3.4 Knowledge Base Realization .....	118
5.4 OWL Link Adaptation .....	119
5.5 Simple Deck Bridge 3D Model – An Example .....	121
Chapter 6 Handling Data Uncertainty.....	125
6.1 Zadeh Semantics with KB Justifications .....	128
6.2 Algorithm KBConsistent .....	131
6.3 Examples.....	132
6.3.1 Example 1 .....	132
6.3.2 Example 2 .....	133
6.3.3 Example 3 .....	134
6.4 Observations .....	136

Chapter 7 Results and Examples.....	138
7.1 Generating Thoroughfares and Their Details .....	138
7.2 Uncertainty of Inferences.....	140
7.3 A Bridge Example in Honolulu .....	144
7.4 An Overpass.....	152
7.5 Overwater Bridge.....	154
7.6 The Champlain Bridge.....	155
7.7 School Zone Example: Ste. Marguerite .....	162
Chapter 8 Conclusion and Future Work .....	167
8.1 On Achieving Objectives.....	167
8.2 Advantages and Limitations .....	169
8.2.1 Inherited Properties .....	170
8.2.2 Ambiguity .....	175
8.2.3 Incompleteness in Data Set.....	177
8.2.4 Limitations .....	177
8.3 Future Work .....	179
8.4 Final Remarks .....	182
Bibliography .....	184
List of Abbreviations .....	200
List of Publications Resulting from this Work .....	202
Appendix A Property Listing and Extractor Associations.....	203
Appendix B OWL Link Porting.....	221

## ***List of Figures***

Figure 1: Google Earth® Image of a Region of Interest (ROI) .....	12
Figure 2: Publicly Available Elevation Raster Data, Georeferenced Imagery, and Linear Feature Information for the Same ROI of Figure 1 .....	12
Figure 3: Detailed 3D Bridge Created by Using our Knowledge-based Approach for the same ROI of Figure 1 (South View) .....	13
Figure 4: GIS Content Definition and Transformation in Legacy Processes .....	19
Figure 5: GIS and 3D Generation Systems [Brockway, 2002].....	28
Figure 6: Raw LIDAR Data [Stelle, 2003] .....	31
Figure 7: Knowledge Base System Components.....	33
Figure 8: The Ontology Spectrum [Bitters, 2005] .....	34
Figure 9: Description Logic Syntax and Interpretation .....	36
Figure 10: NGATE’s Detection of Elevations [SocetSet, 2014] .....	48
Figure 11: Popular Fuzzy Logics (After [Bobillo & Straccia, 2011]) .....	62
Figure 12: GIS2Geometry System and Process.....	65
Figure 13: Data Transformations and Usage in our Process.....	65
Figure 14: Generic Transport, Bridge and Covered Bridge Example Taxonomy .....	96
Figure 15: Retrieving The Most Specific Type of InstanceX using SPARQL.....	98
Figure 16: Retrieving a Property Value using SPARQL .....	98
Figure 17: Shape Definition SD Ontology Mapping .....	103

Figure 18: Road Properties .....	107
Figure 19: OL Vectors on a Road Linear Record .....	110
Figure 20: Normal From Three Points Defining Plane.....	110
Figure 21: Width Using OL .....	110
Figure 22: Average Normal .....	112
Figure 23: Direction Vector D from N and OL (Z-axis is up).....	112
Figure 24: Separators with Markings Showing Traffic Direction .....	114
Figure 25: Relations on Geometries .....	116
Figure 26: Deck Only Example .....	124
Figure 27: Shp0 Polyline Record.....	138
Figure 28: Generated <i>Connected_Segments</i> .....	138
Figure 29: Knowledge Graph Defining Shp0 .....	139
Figure 30: Example Dialog Querying User for Fact Verification .....	144
Figure 31: 10m Resolution Elevation Data.....	144
Figure 32: Shapefile Data Sets in Points, Linears and Polygons .....	145
Figure 33: 1-foot Resolution Partial Imagery Data.....	145
Figure 34: Honolulu Bridge Satellite View .....	146
Figure 35: GIS Data of Honolulu Bridge.....	146
Figure 36: Ground View using GenesisRT without imagery .....	147
Figure 37: Ground View using GenesisRT.....	147
Figure 38: Ground View Visualization using Presagis Terra Vista.....	147
Figure 39: Elevation Profile of ShapeID 4 .....	149
Figure 40: Retrieved Values for the Honolulu Bridge.....	151
Figure 41: Generated Cantilever Beam (North View).....	151
Figure 42: Overpass Areal View.....	152

Figure 43: Overpass GIS Data Sources .....	152
Figure 44: Retrieved Values for Overpass.....	152
Figure 45: Generated Overpass.....	153
Figure 46: Overwater Bridge Areal View.....	154
Figure 47: Overwater Elevation Data Source .....	154
Figure 48: Overwater Road, Building, and Tree Features Combined .....	154
Figure 49: Side View of the Champlain Bridge.....	155
Figure 50: Overlay with Shape Data (Google Earth®).....	156
Figure 51: Tied Arch Bridge using Edges 2 and 3.....	158
Figure 52: Cantilever Open Spandrel using Edges 1 and 4 .....	159
Figure 53: Cantilever Through Arc using Edges 1 and 4 .....	160
Figure 54: Champlain Bridge Results.....	161
Figure 55: Retrieving Entities <500m Distance from a School Using SPARQL .....	163
Figure 56: View around Ste Marguerite, Laval-Des-Rapides (Google Earth®) .....	165
Figure 57: Instances (133) of ROI layed over Google Maps®.....	165
Figure 58: Results from Query in Figure 55 on Ste Marguerite Primary School .....	166

# ***Chapter 1***

## ***Introduction***

Digital synthesized visuals of real world landscapes are essential in military, simulation, earth sciences and serious gaming applications. Such applications require detailed representations including 3D models of land entities to be generated in short turnaround times. In this thesis, the term “detailed representation” refers to detailed property values of all entities needed by the end user in such a landscape representation. Turnaround time is defined as the time required, after acquiring the necessary initial data, to create a corresponding landscape representation (digital) with detail as required. The military, for example, is already looking at ways to reduce this turnaround time between receiving new data about a specific mission and having the mission training simulator ready for mission rehearsal right before real world execution [Fillmore, 2006].

The process of creating detailed digital representations for scenes is often a very manual process. 3D geometric models, terrain, and other components are usually created from Geospatial databases that define a real world region of interest (ROI). Three main data sources are invariably contained, also collectively called Geographic Information

Systems (GIS) source data, namely, elevation data, imagery and feature data. While, elevation data and imagery are easily available, feature data (also known as vector data) and the associated 3D models are often not. The most common feature files format is the *Shapefile* standard, which is described in detail in Chapter 2. Creation of shapefiles is labor intensive. As of today, in most cases, elevation and imagery are used to semi-automatically generate a triangulated and textured mesh of the terrain (ground surface). 3D digital models of other entities in the landscape are hand crafted or altered using modeling tools based on descriptions of features or their respective real world images.

The current methodology for shapefile creation involves domain experts. GIS data is presented to these experts and they extract and make certain information explicit and well-attributed for other systems or computer programs to consume. Thus, one of their main tasks is to resolve the ambiguities in the data by analyzing the different sources of data and to create the most precise and unambiguous definition for each entity in the ROI as required by the application. This may require testing the results of their definitions and iterating back until an acceptable representation is achieved. We will discuss these techniques further in Chapter 3.

A detailed digital representation of a real world landscape would need, along with terrain data, adequate representations of all additional entities—both above and below the ground—such as tunnels, water bodies, bridges, buildings, roads, trees, etc. GIS datasets contain the initial data needed to model these real world objects, although their details (property values) may only be implicitly available. By implicit we mean that the values are not explicitly present in the shapefiles or cannot be derived using algebraic methods from the data sources, but rather one has to infer the values from the collective data set.

Our research addresses the problem of using these multiple GIS source datasets to automatically generate details of the entities in the ROI including 3D details. We differ from other processes discussed in Chapter 3 by making certain implicit facts in the initial data explicit for defining property values and aggregating uncertainties for decision making on the produced output. These property values are used to procedurally generate entities in the ROI as required by the end application.

Entities in the ROI which are present in the shapefiles are organized as instances in a knowledge base with values for their properties. An expert cartographer's knowledge is formalized by means of an ontology. Description Logic reasoners are used to infer information about the instances so as to reveal their specific identities (say from a generic line element to bridge to covered bridge and so on), and to provide associated property values needed for detailed representation of the entities. "Data Extractors" is the name of the framework we use to fuse data from the different GIS sources (shapefile, elevation, and imagery datasets) and to automatically derive values from explicit and implicit knowledge present in the GIS source data.

Computer vision based techniques such as pattern recognition and image analysis methods are often used to computationally estimate the entities and their properties. Due to the imprecision and ambiguity of captured sensor data and in the results of these vision-based algorithms, values cannot be obtained with complete (100%) certainty. Unlike uncertainty in knowledge that deals with defining the meanings of ambiguous concepts such as "often" in "a bridge often has supports", assertions from GIS sources only require a certainty value on their validity (the certainty of existence with regards to the real world) e.g. how certain are we that "wood is the type of cover of a bridge

instance i”. The Data Extractors framework handles object-object relationships (includes spatial relationships), object-value relationships, as well as their data uncertainties. Following work by [Poole et al., 2009], uncertainties are recorded as attributions on their respective assertions under the principle of the separation of knowledge assertions from the uncertainties associated with this knowledge into different contexts. Inferred axioms are associated with a resulting value under Zadeh fuzzy semantics [Zadeh, 1965]. Semantic Web ABox Realization service is extended, and explanation services (KB Justifications as per [Horridge et al., 2009]) are used to associate uncertainty values to inferred assertional axioms. We address complex and multiple explanations for a certain inference by reducing every explanation into a minimum set of axioms, which we call the variable set. This set allows us to calculate the inferred axiom's certainty value.

Our methodology makes the availability and analysis of GIS data more automated, modular, verifiable, and the results more streamlined for other applications to use. The fusion of different data sources and collections allows a common knowledge base defining entity instances logically. This KB is then further used to generate 3D representations of the entity instances. We extended OWL Link [Liebig et al., 2008] to C++ and C# in order to enable module intercommunications using the Semantic Web framework. We show a way to address uncertainty in this data with minimum impact on system complexity or change of semantics expressivity. We discuss the advantages and limitations of this process with respect to legacy methods. We present our implementation details using the transportation domain as an example which includes road networks and transportation-related features and models, especially bridge structures. However, this process can be applied similarly to other domains.

## **1.1 Motivation**

Digitally synthesized versions of familiar 3D environments enable users to be immersed into the virtual worlds created by the computer and thus to fully utilize their everyday capabilities and senses to solve a given problem. They also provide a good training mechanism in complex and hazardous environments, lowering the cost of hardware training and at the same time making it more effective. Detail is important in creating a realistic 3D environment, an important requirement for providing the immersive experience in many domains such as entertainment, education & training, simulations, engineering and science. Constructing a 3D model of the ROI with the required level of detail usually consumes a lot of manual work and design. Methods which can help create detail without significant domain expert involvement are essential.

Several organizations including the National Geospatial-Intelligence Agency (NGA) cooperate under the Multinational Geospatial Co-production Program (MGCP) to collect, produce and share digital geographic information. Despite the availability of information, and established quality standards and requirements such as those shown by [Fillmore, 2006], there is a need to reduce the turnaround time between obtaining new GIS data of an area and having a 3D model suitable for simulation systems and training applications. Visualization is only one application targeted by the MGCP.

Urban operations, for example, have lately been the focus of several research efforts due to the high requirement in detail and fidelity of the geographical surface features. As compared to an Out-The-Window (OTW) view of a flight simulation where terrain surface and features are only represented as a texture map applied to generic 3D

shapes, urban simulation requires detailed 3D modeling of close-by objects. We shall refer to a detailed 3D model as a geospecific model as it is more accurate in reflecting the real world object's dimensions and shape features as compared to a generic model which is often simply defined as a properly sized box with an appropriately assigned color/textured. To produce geospecific 3D models representing entities in a certain area, an expert cartographer would need to study the different layers of source data available and estimate property values for the entities. This may need many iterations before finalizing the result.

A few interactive tools exist which can assist the expert in this task. But it is very difficult for a tool to consolidate knowledge independent of the actual data being studied, when this data is in different forms and standards such as rasters (for elevation and imagery) and records (for features and culture). Computer vision based techniques work primarily with raster images for identifying features and their details. Restricting oneself to pixel-based techniques is inadequate for our problem.

## **1.2 Problem Statement**

Our main problem can be stated as follows:

Given GIS source data (feature and feature values in the form of shapefiles, raster images of the landscape in the form of pixelized data, and elevation data in the form of a height raster or triangulated height values), create a 3D digital landscape representation

for that ROI having all the detail with correctness and precision that exists explicitly or implicitly in the given GIS source data, and with reduced human expert involvement.

A secondary problem to the above can be stated as follows:

Given GIS source data for a ROI, in which certainty in the form of percentage values are associated with the feature and feature values in the sources, handle the certainty values for all the entities and their property values in the KB.

### ***1.3 Objectives***

Our main objectives can be summarized as follows:

- To define a methodology to extract the knowledge (facts about entities) that exists in the GIS source data in a format independent manner. In order to be able to do this, create a mapping which maps facts in the source data to assertions in a knowledge base. For this mapping, we need to define:
  - Domain knowledge ontologies under ROI constraints, considering knowledge that can be collected based on the ROI and that is relevant to our needs (domain-based capabilities, driven by implementation needs).
  - Data to domain mapping ontologies separating source format mapping specifics from domain knowledge descriptions.
- To develop a framework which will infer implicit information in the knowledge (use the domain knowledge ontologies) w.r.t. entities in the landscape of the ROI and

further extract missing values for required properties of these entities from the input data sources. Develop procedures by which the inferred entities and their property values can be used to procedurally create detailed digital representations of the landscape.

- To define a mechanism to handle and compute uncertainty in the added data.
- For all the above, to comply as much as possible with public standards and technologies.

## ***1.4 Methodology and Proposed Solution***

Our methodology [Eid & Mudur, 2009] is driven by the fact that we see the task of creating geospecific 3D models from GIS source data as composed of two distinct steps: facts extraction and spatial knowledge extraction. We plan to devise mechanisms for facts extraction, or feature extraction (GIS to knowledge), which fuse data by adding knowledge in the form of assertions to a formal knowledge base. This formal knowledge allows for inference of new information about the instances, i.e., new information becomes explicit. Spatial knowledge extraction (knowledge to representation) uses the available information in the knowledge base to construct a corresponding landscape representation for the ROI.

Further, as part of our implementation methodology, we propose the use of Semantic Web technology for formally modeling the knowledge base. Properties inferred by the semantic engine will be processed further to extract corresponding values from the

input sources. These are also inserted as assertions into the knowledge base which in turn contributes to further formal knowledge about the instances, and further possible inferences. Uncertainties based on data imprecision or as a result of methods used will be associated with the assertions. Using Semantic Web technology therefore allows data independence; the same process can be used on different sensor data sources (various feature, elevation, and imagery sources) to extract needed data.

Next, a geometry definition engine will be developed to demonstrate the automation in creating the required representation. Specifically, we create 3D features using the property values extracted from the knowledge base. The geometry definition engine will have needed parameters for every 3D model. We use the SPARQL querying language, described in the following chapter, to retrieve the landscape entities and associated properties using a parametric models ontology. For example, for a bridge model, the system can extract quantitative parameters such as bridge width, span and cover texture from the knowledge base. The extracted information is then used in conjunction with a parametric 3D model repository to procedurally construct a higher detail geospecific equivalent of the real world entity as defined in the collection of GIS information.

The three components described above, together, effectively mimic the process of data extraction and estimation done by experts and thus reduce the effort involved. The initial setup for a GIS features domain (like transportation) would require expert time (one time effort) in defining the ontologies and the scripting of modular data extractors for properties and relationships.

## ***1.5 Implementation***

The system proposed in this research is a generic system for assisted generation of 3D digital models of the landscape of a given region of interest from raster, vector and symbolic source data. We have developed a system whose architecture will be described in detail in later chapters. While this system architecture is domain independent, we have populated it with knowledge about the transportation domain, specifically entities such as bridges, roads, tunnels, etc.

Given below is a list of the tools used in developing this system:

- The Open Geospatial Consortium and Ordnance Survey for public standards and ontologies.
- Protégé Editor 4.3 to create our ontologies and testing.
- Pellet 2.3.1 and Hermit 1.3.8 as Semantic Web reasoners.
- OSGeo's GeoTools API 9.3 for GIS data processing, querying and presentation.
- OWL API 3.4.5 for binding into a semantic web engine and complying with OWL 2 specifications in reading, writing and querying.
- Eclipse IDE Juno to develop our prototype and knowledge base using the above Java APIs.
- MapWindow GIS 4.7.5 for investigation and testing of sources with plugins to use our knowledge base system.
- OWL Link API 1.2.0 for the porting of the DIG protocol into C# and C++
- Microsoft® Visual Studio 2010 to develop OWL Link for C#/C++

- Microsoft® .Net Framework for building OWL Link for C#/C++ and MapWindow GIS plugins.
- Google Earth Pro®, DVC GenesisRT®, Presagis Terra Vista® and Creator® for investigation of current state-of-the-art capabilities and needs for a solution that can be integrated with current processes.
- U.S. Geological Survey's National Elevation Dataset, Ordnance Survey resources, and Natural Resources Canada GeoGratis datasets for validation.

## ***1.6 Simple Example from our System's Results***

The image shown in Figure 1 is taken from Google Earth® at 21°21'02.72" N 157°53'41.13" W on January 23, 2014 with a view towards the South. Although a comprehensive visual scene is present, the lack of a 3D model for the bridge is noticeable. The data for such a scene consists of elevations, rasters and annotations to insert 3D models in the scene.

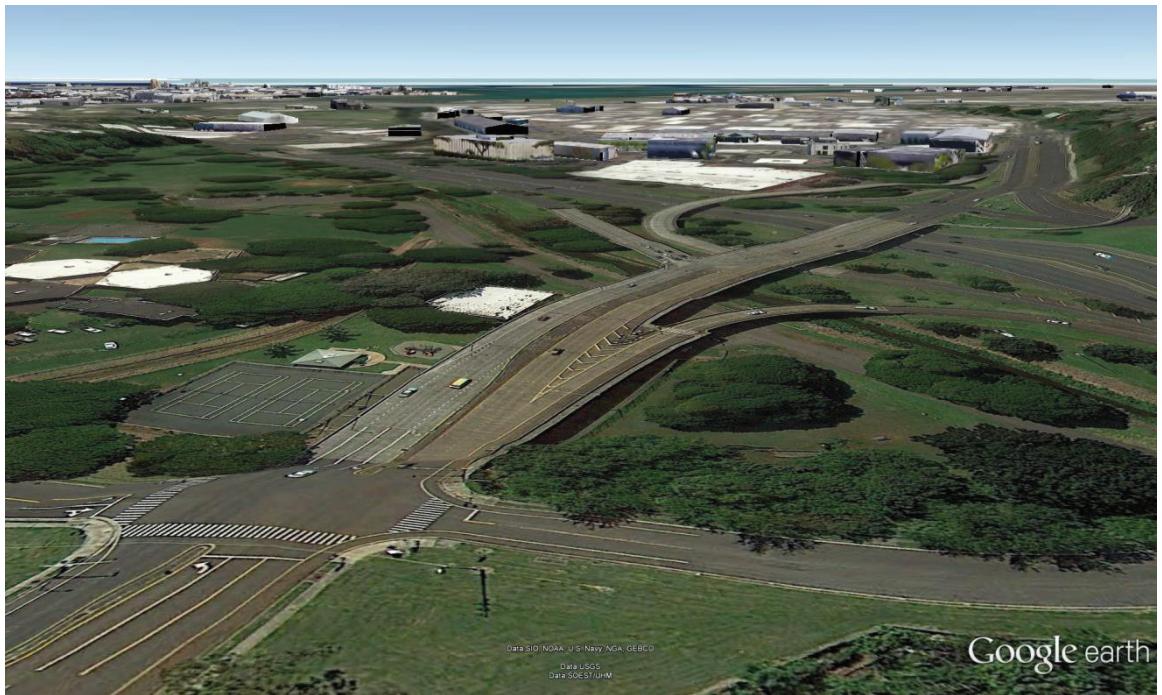


Figure 1: Google Earth® Image of a Region of Interest (ROI)

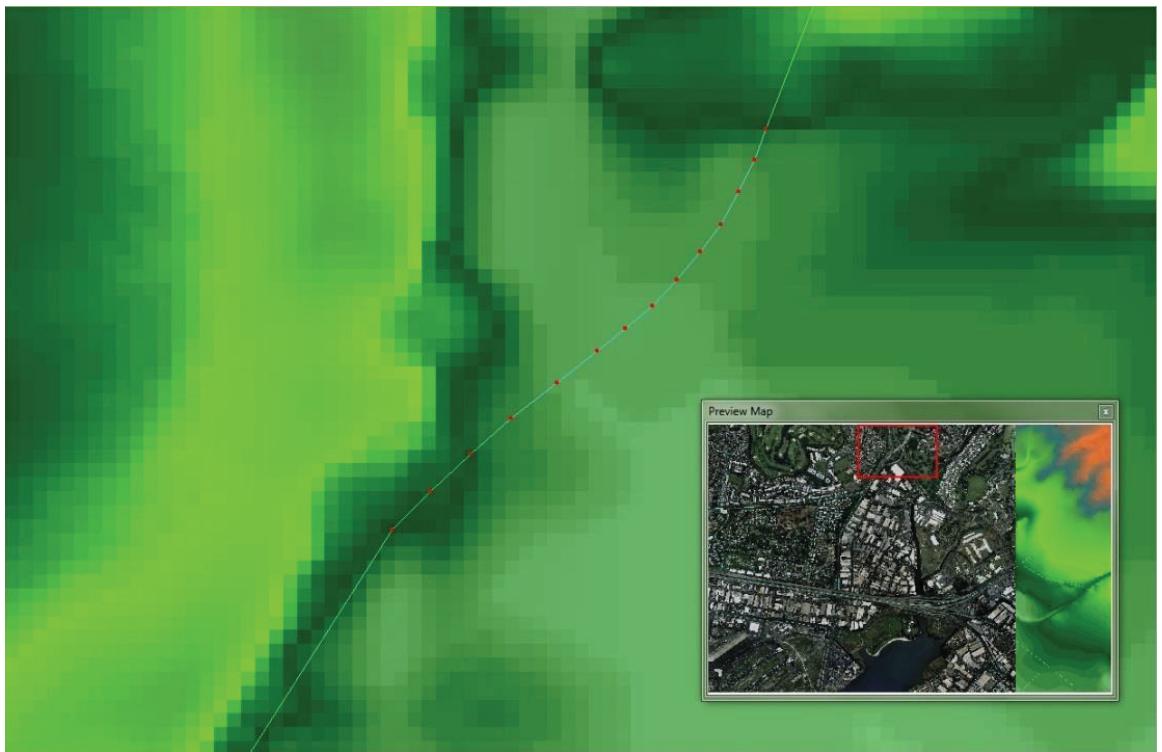


Figure 2: Publicly Available Elevation Raster Data, Georeferenced Imagery, and Linear Feature Information for the Same ROI of Figure 1

For this same area, publicly available elevation raster data, georeferenced imagery, and linear feature information are shown in Figure 2 (North up). These were retrieved (through the USGS website) using remote sensing equipment in the ROI. Another linear feature can be defined for the orthogonal road crossing under the bridge as well. However, in either case, a human can easily distinguish the existence of a bridge whether due to the elevation profile or due to the existence of the orthogonal roadway.



Figure 3: Detailed 3D Bridge Created by Using our Knowledge-based Approach for the same ROI of Figure 1 (South View)

Figure 3 shows the results of the application of our knowledge-based approach to create a 3D representation of the landscape for the same ROI. The 3D details of the bridge entity are clearly visible. Our process first uses the data and domain ontologies to map segments of the road linear as connected segment entities in the knowledge base representing the road. Extractors associated with entity class properties in the domain ontology are used to evaluate spatial relationships between knowledge base entities and

to retrieve missing values for each entity's properties. Extractors also associate the uncertainties found as part of the facts. Deductive reasoning based on a domain ontology is used to reclassify entities to their most specific classes based on their relationships and property values. The result is a knowledge representation of each entity in the ROI. The process then uses the knowledge representation of each entity to procedurally generate a 3D model representation, as shown in Figure 3, while uncertainties can be used for decision-support for example; a user is presented with options and the data that resulted in those options for an informed decision to be made.

## **1.7 Contributions**

### **1.7.1 Conceptual Contributions**

Spatial reasoning and geospatial semantic web are research topics currently in focus. Our first and most significant contribution is the introduction of an overall methodology for creating detailed 3D digital landscape representations using semantic web technology that makes use of GIS source data collectively. We believe that this has not been attempted in earlier research in the field of 3D model synthesis in the GIS field.

A second important contribution is our formulation to derive uncertainties of inferred assertional axioms in knowledge bases based on KB explanation services. The idea of processing explanation services to derive uncertainty values for the entities and

property values in the output landscape representation is innovative. To the best of our knowledge, this has not been explored previously.

### ***1.7.2 Practical Contributions***

As part of our work, we created several ontologies extending existing work from the OGC [OGC, 2014], Ordnance Survey [OSUK, 2014], and the VOTT [Bitters, 2005]. Specifically, this includes the following:

- a 3D landscape data ontology defining the organization based on visual objects layout, cartography and civil engineering principles for reasoning in this domain,
- a shapefile-data-to-3D-landscape-data bridging ontology for the fusion of GIS source data, and
- a listing of 3D models and parameters ontology for 3D entity models. The ontologies are formal and general enough to support testing of our principal ideas.

Ambiguous or inconsistent representations are a major issue in delivering a completed terrain database in short turnaround time. By using Semantic Web technology, the created ontologies and knowledge base can be automatically and easily verified for consistency and explained. The knowledge base can also be used by other external systems such as provide further information as input to the visualization system or consume available information in the knowledge base. We have not found any similar work in published literature in this area. Ordnance Survey seems to have some data fusion methodologies, but they have commercial value and are not published.

We ported OWL Link from Java to C++ and C# in order to test our methodology using available GIS toolkits and APIs. OWL Link authors expressed great interest in our port and mentioned that many users would like to use it.

The ontologies, the OWL Link port and the extended KB Realization code using KB explanation services are provided publicly on the internet.

Earth Sciences can use our approach to visualize implicit information within a 3D scene visualizer. Currently, most systems are based on the querying of information databases to retrieve needed data. Presenting information in 3D with the needed details offers a more visual solution which even untrained personnel can interact with.

The greatest beneficiaries are likely to be the modeling and visual simulation industries which affect commercial, earth sciences, immediate response, homeland security, and military organizations. Our solution provides a way to use stable, available and performance-based legacy standards to create detailed 3D environments for visuals and simulations with reduced user interaction.

## ***1.8 Thesis Layout***

So far, the reader has had a brief overview of the important features of the proposed methodology. The rest of the thesis is divided into the following chapters:

2. Background
3. Related Work

4. Knowledge Base and 3D Landscape Creation
5. System Implementation
6. Handling Data Uncertainty
7. Results and Examples
8. Conclusion and Future Work

Given below is a brief outline of the rest of the chapters discussed in this thesis:

We first present the background related to our domain of research, which includes GIS standards, topological relations and semantic web technology. Then, we review related work that addresses the problem of creating 3D visuals from GIS source data and methods to synthesize missing information. We then present our final process and its theory outlining the evolution through earlier attempts and their limitations. We present the implementations of our facts extraction (GIS2KB) and spatial knowledge extraction (KB2Scene) processes along with the KB Realization extension to address the calculation of uncertainties for inferred axioms. We present sample results from our implementation and, finally, discuss advantages and limitations of the process based on the technologies and tools used.

Appendix A lists our Data Extractors associated with the property listings for the bridge classes we addressed. Appendix B contains our notes on porting OWL Link to C# and C++. Moreover, the full system implementation in predicate notation form, some of our implemented algorithms as well as developed ontologies are available publicly at [http://users.enics.concordia.ca/~pa\\_eid/PhD/](http://users.enics.concordia.ca/~pa_eid/PhD/).

## ***Chapter 2***

### ***Background***

The National Geospatial-Intelligence Agency [NGA, 2014] and the Environmental Systems and Research Institute (ESRI) are the two most active and recognized organizations controlling GIS standards and developing new ones based on scientific, military and commercial needs. GIS data sources are usually available from sources such as the US Geological Survey or Natural Resources Canada. There are various types and formats for this data but semantically, all formats try to represent the region of interest (ROI) with the best possible data structures to provide low complexity and high flexibility for the specific application needs.

Figure 4 shows the different GIS source data types (nodes) and current methods (arcs) used to transform GIS data types in order to generate a 3D representation of various real world objects in a certain ROI. For example, the  $T \rightarrow D$  ranging process which calculates digital locations using time of travel analysis outputs a Digital Elevation Model (DEM) that is then leveled and cleaned by user involvement to produce a Digital Terrain Model (DTM). This GIS data flow diagram was produced after a thorough review

of current systems and methodologies in use for acquiring/creating each of the GIS data types [Eid & Mudur, 2009]. It provides us with a comprehensive view of the data sources involved and related methods. It also provides us with a context in which to compare our approach with other methods which try to address the same problem of creating detailed landscape representations from GIS source data.

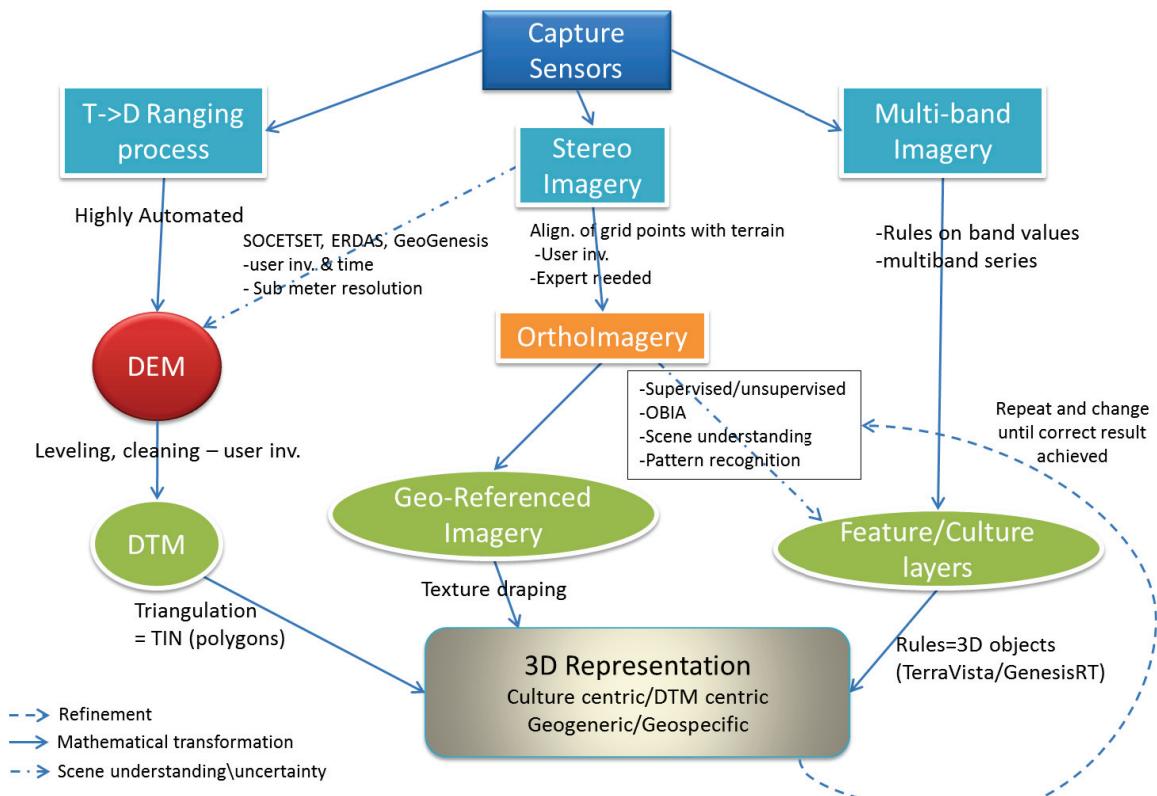


Figure 4: GIS Content Definition and Transformation in Legacy Processes

For example, computer vision based scene understanding techniques such as Object-based Image Analysis (OBIA) are primarily concerned with data transformations of raster data to feature data, as denoted (dotted arrow) in the graph. These will be covered further in Chapter 3. A lot of manual work may be further required in verifying the 3D representation and changing the classification procedures used to output features from

imagery. Another time consuming process is the manual modifications of the outputted feature details to represent the associated 3D object accurately. This iterative process that experts have to go through is marked by the dashed line in the figure (going back to the refinement of Feature layers). In our work, we are mostly interested in the 3 input type classes shown as coming into the 3D representation generation node: elevations, imagery, and features.

We focus simultaneously on all the layers of information available and use collective knowledge about the scene in order to derive detailed 3D representations of objects in the real world landscape. The three arcs coming into the 3D representation node will be the input to our semantic engine which plays a significant role in our approach to create detailed landscape representations. Thus, it should be clear from this discussion that while we can benefit from some of the techniques developed in the domain of scene understanding, there is no significant overlap. A more detailed overview of existing work for the creation of 3D representations using scene understanding techniques is presented in the following chapter.

In Figure 4, as part of elevation sources, DTM is a filtered and leveled version of DEM. DTM is usually used for the triangulation process. Similarly, OrthoImagery is geo-referenced and transformed to correlate with the DTM. The result is referred to as Geo-Referenced Imagery which can be used for texturing the triangulated DTM. OrthoImagery is also used to define feature (culture) layers which are then used to define 3D objects that can be added to the scene. There seems to be a consensus among GIS systems and users for using Shapefiles [ESRI, 1998] to represent all types of surface feature information due to the standard's comprehensive extensibility to fit all needs.

Multi-band imagery as well as other types of sources are also used to define feature layers e.g. materials for roads and areas.

A 3D model created with generic properties results in a geo-generic 3D representation. A 3D model created with respect to specific properties trying to best represent the real world object results in a geo-specific representation. We are focusing on creating detailed geo-specific digital representations of visible real world objects. For simplicity, we will be focusing on using DTM, OrthoImagery, and Shapefiles respectively for our prototype implementation. We consider these sources representative enough of GIS data source collections for an ROI. Other sources could be similarly handled either as derivations of the sources chosen or as part of the Data Extractors Framework (introduced in Chapter 1).

We also cover in this chapter other common standards used in this field. It may be noted that while standards like OpenFlight are available to represent any 3D object to the required level of detail, present methods in use for creating such 3D data are essentially manual and human effort intensive. Light Induced Detection and Ranging (LIDAR) [Opitz et al., 2006] data is another data type concerned with 3D which involves the use of a large format 3D scanner to digitize 3D objects in the real world. Given the difficulty of using large format scanners, in general, LIDAR data is not available for most landscapes. Our approach is to create detailed landscape representations by making use of the collective information available in different source data types. To that extent in our investigation, LIDAR data will not be directly used as GIS data source input, but, if available for specific objects, it could be used to add further detail to the 3D representation of such objects.

In the following sections, we describe the types of data sources in further detail. We also discuss a common topological relations model, Semantic Web technology, and reasoning services relevant to our work, all of which are important for our approach.

## **2.1 Relevant GIS Sources**

Geospatial datasets constitute a digital representation of geospatial features at a point on the earth's surface. Terrestrial surveys, photogrammetry, satellite data, digitized analog maps (charts), GPS data, statistical files, other data such as physical, environmental, boundary, etc. are all types of geospatial datasets. While the earth surface has innumerable features, we only consider the following data sources, which we refer to collectively as “GIS source data” and are related together with geospatial coordinate system descriptions and models:

- Elevation data, describes altitude at locations above some reference level,
- Imagery data, which can be applied on the generated landscape surface for a more realistic view,
- Feature data, describes the ROI's features above, on and below the elevation points.

Terrain, or elevation data, defines the third or vertical dimension of land surface [Wikipedia, 2014]. Features, commonly represented through shape layers, define the terrain culture information. A digital terrain model (DTM) is a digital representation of ground surface topography or terrain including related data objects. A digital elevation model (DEM) depicts only the elevations or altimetry of ground [Kraak, 2010]. High

level-of-detail DTMs for example are used in real-time military systems such as weapon guidance, sensor models, and aircraft navigation systems. DTED, a format that represents DTM/DEMs, is considered as a high performance accurate depiction of the actual terrain. It was developed by the NGA and was lately revised in 2000 for performance specification under military standard MIL-PRF-89020B [DTED, 2000]. We use this format. All terrain elevation standards, and specifically DTM, are currently described as a value, the altitude or elevation, at specific latitude/longitude coordinates. They also usually contain descriptions of the coordinate system and earth model they use. The collection of locations describes a 2D area (grid) representing the terrain surface. Using the elevation data as the third dimension, the DTM points can be graphed in 3D. The DTM defines a terrain based on grid spacing with every grid point as a location on the face of the earth. The location is saved as and converted using a specific coordinate system and projection such as the WGS84 or UTM NGA standards. Both elevation and imagery sources are considered of raster data type and we interpret them as arrays of values where each cell represents a point location as a pixel color or as a relative altitude value.

Imagery data has multiple source formats such as ECW or JPEG2000. ECW or Enhanced Compression Wavelet is an ERDAS proprietary wavelet compression image format optimized for aerial and satellite imagery [Wikipedia, 2014]. JPEG2000 is a comparable format that allows better compression performance as well as scalability. It was created by the same group that created the JPEG format. We use ECW as part of our sources. Imagery data is in itself very varied: Visible light spectrum, Material rasters, multi-band and infrared (IR) are some types of imagery. Visible light spectrum imagery

has been analyzed for visual photographic patterns for value and texture extraction, while other types of imagery have been analyzed using pixel-based techniques under defined criteria in the respective spectrum.

Feature records contain information about the surface of the terrain such as the environment, buildings, roads, lakes, land use, etc. For example, a building on the map could be an area feature object which specifies the feature type as *residential\_building*. It also stores the height, width and length as well as the orientation of the building in the record. Feature records include different kinds of detail about the represented area, and along with elevation data and imagery, they have the potential to be effectively used for creating detailed 3D representations of landscapes. There are two basic abstractions for representing real world objects: discrete objects (a house) and continuous fields (land use). In fact, elevation above sea level is an example of a continuous field, although for historical reasons, it is represented separately through separate elevations data formats. Correspondingly, there are two formats used to store data in a GIS: vector and raster. Vector data represents 3D geometric entities in the form of points, lines, or polygonal regions, whereas raster data is in the form of values in a discrete rectangular grid structure. Even though, collectively, source data may be having the information with certain detail implicitly embedded in a fashion that is often rather easy for a human to infer, many 3D visualization systems are not equipped to render the 3D landscape with the same detail. This is mainly due to the fact that these systems require the 3D landscape geometry in a specific form that cannot be easily derived as a simple transformation of this information in GIS source data. As a result, it is primarily a human intensive manual process that is in vogue for creating detailed 3D landscape representations from such GIS

source data. Typically, users would manually attribute features in the individual layers by studying the satellite images and elevations of the area of concern. Automation support in this process is a current topic of intense research and forms the main problem area for our research work as well.

### ***2.1.1 ESRI Shapefiles***

Shapefiles, an ESRI standard [ESRI, 1998], were initially created in order to address the growing need of 3D features and flexibility on the number and types of properties required by each feature. It is the most common format used by GIS users and GIS software. Shapefiles allow the definition of overlays and annotation of GIS source data such as images and multi-band sources. Shapefile data is flexible and the records can be mapped into object data. For example, some semi-automated systems can analyze continuous patterns in the imagery based on texture and generate matching Shapefiles with linars defining polyline information such as for roads and waterways.

GIS experts can add necessary attributions on the Shapefile record to define explicitly some properties in the image like type of road or number of lanes etc. Shapefiles can have detailed representations for a certain area depending on how much effort the expert has put in creating it. It is a very hard and lengthy process to generate this information and maintain it up-to-date. However, a detailed set of Shapefiles has very high fidelity results and modeling advantages [Dewberry et al., 2002]. Washington DC, for example, has a Shapefile set that models every small detail in the city, even park benches [DCCatalog, 2014].

Shapefiles spatially describe vector features of which we address the most commonly used geometry types: point, polyline or polygon. These are defined by the following data structures:

```

PointZ
{
    Double X; //X coordinate
    Double Y; //Y coordinate
    Double Z; //Z coordinate
    Double M; //Measure
}

PolyLineZ
{
    Double[4] Box; //Bounding Box in X, Y
    Integer NumParts; //Number of Parts
    Integer NumPoints; //Total Number of Points
    Integer[NumParts] Parts; //Index to First Point in Part
    Point[NumPoints] Points; //Points for All Parts
    Double[2] Z_Range; //Bounding Z Range
    Double[NumPoints] Z_Array; // Z Values for All Points
    Double[2] M_Range; //Bounding Measure Range
    Double[NumPoints] M_Array; //Measures
}

PolygonZ
{
    Double[4] Box; //Bounding Box in X, Y
    Integer NumParts; //Number of Parts
    Integer NumPoints; //Total Number of Points
    Integer[NumParts] Parts; //Index to First Point in Part
    Point[NumPoints] Points; //Points for All Parts
    Double[2] Z_Range; //Bounding Z Range
    Double[NumPoints] Z_Array; //Z Values for All Points
    Double[2] M_Range; //Bounding Measure Range
    Double[NumPoints] M_Array; //Measures
}

```

A shapefile follows a schema where it declares the geometry type (referred to as *ShapeType*) that the file will be using and a bounding box for all the elements in the file under a certain projection and coordinate reference system. It then defines an internal file schema applicable for every record with the geometry type, and a list of properties (each property with a name and a type) that the records contain values for. They support both 2D and 3D definitions. Each record then defines a real world feature using values for the geometry (a list of points defining the shape), a bounding box for the feature defined, and a value for each of the properties defined in the schema. Properties and values are stored in open dBase format. Properties can be of type String, Integer, Double or flagged as a

reference to another shape record index. Examples of properties include 3D object file pointer, width, height, elevation, orientation, material type, object type, meta information, etc. Shapefiles do not have the structure to store topological relationships; however, this can be done externally using topological relations computed on the records' geometries and properties.

All the information in shapefiles for a ROI constitutes part of the knowledge necessary to create a detailed digital model of that ROI. This would require a mapping of the data in shapefiles into knowledge base assertions. We will show how this is done in our source to domain mapping implementation.

## ***2.2 Other Commonly used GIS Standards***

After [Brockway, 2002], systems like ESRI ArcView allow the manipulation and creation of GIS databases, such as a geodatabase, to act as a common repository and standardized way that promotes easy and fast access to the needed information. Then, at a higher level in 3D, systems like ESRI's 3D Analyst and Sitebuilder 3D use geodatabases to allow the manipulation of the terrain in 3D and their visualization. Figure 5 shows different known systems and their positions relative to each other. 3D presentation systems like Creator Terrain Studio (Multigen-Paradigm Inc.), Model Librarian (ERDAS IMAGINE) or TerraVista (TERREX) typically involve a lot of user involvement to create the 3D representation of an ROI. Individual objects are created usually with extensive user input and explicit attributions. These create 3D models in a format such as

OpenFlight. This standard is used to represent any 3D object to the required level of detail in a GIS context.

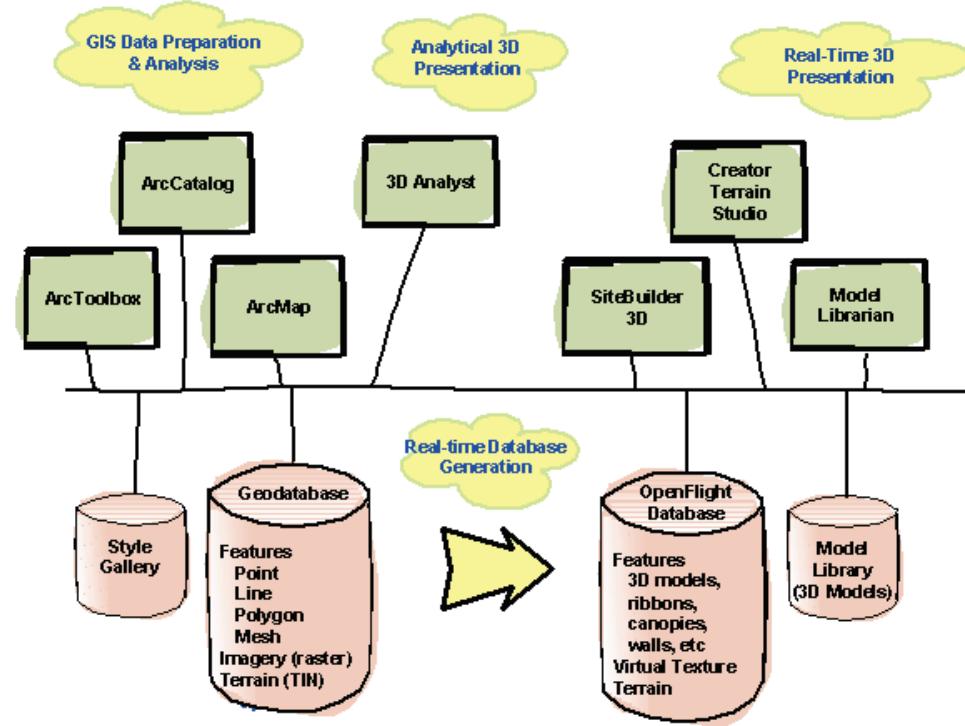


Figure 5: GIS and 3D Generation Systems [Brockway, 2002]

### **2.2.1 Legacy Sources**

The grid describing the DEM is based on a specific sampling of an area, typically using an appropriate scanner. These same scanners can also collect data about the terrain like type of ground at a specific location and save it in another format such as DFAD. DFAD or Digital Feature Attribute Data is a type of feature map in a grid format similar to a DEM but with the data representing type of earth surface rather than elevations.

DFAD [DFAD, 1994] was also developed by the NGA, previously known as NIMA, and is currently being phased out in favor of vector-based standards. Legacy

systems still use it however for performance reasons. DFAD is not suitable to be used in 3D visualizations according to [VectorData, 2014], but along with DTED, it provides an efficient database for weapon and sensor systems and simulations.

Newer standards were developed, like NGA's VMAP [VMAP, 2014] and more recently ESRI's Shapefile formats, which are based on vector representation of the features in an ROI. Shapefiles, described earlier, have similarities to VMAP, but add the capability for the user to create their own attribution including linking 3D models to each of the feature objects defined. VMAP can be considered and processed the same as polyline records in Shapefiles.

Many other types of GIS data exist but they are mostly in the form of raster or vector multi-resolution data. These are considered obsolete as they are only used by legacy systems. Our process, if necessary, can be made to work with such other data types by implementing suitable data extractors.

### ***2.2.2 OpenFlight Standard***

The OpenFlight standard [OpenFlight, 2007] is particularly popular commercially for its simplicity and broad compatibility. It was introduced by Multigen-Paradigm and continues to be improved with a current version of 16.4. Being an open standard, importers and exporters for this format can easily be found publicly. OpenFlight is a format that can also be stored on disk as a compiled structure for fast access and loading at runtime. It can describe any 3D object using simple shapes and their positions and orientations in space organized in a database. OpenFlight stores the coordinates of each

of the polygons in  $(x, y, z)$  based on a model origin defined in the database header. It supports GIS standards by matching the model origin to a GIS database origin in a certain earth model and projection. This way, any point in the OpenFlight model can be projected to its GIS equivalent. In a 3D viewer, the polygons can be used to draw the model in 3D. The compiled structure represents an executable procedure to display the model in 3D.

Initially, this standard was created to describe a terrain and visualize it in a 3D scene for aircraft cockpit displays, and was later extended to handle any kind of 3D model. An airport where an aircraft would land or takeoff, for example, would be modeled with high detail whereas the landmass around would be represented using a few, large textured polygons. OpenFlight supports transformation nodes in the database tree that allow the manipulation of a subtree. Subtrees usually represent a piece or object in the model. It describes every polygon with 3D points that can contain extra information like feature type and texture information. The standard even supports levels of details that can be constructed as subtrees. Dynamic level of detail can be achieved as a function of the viewer's distance and the size of the closest object on screen by gradually switching between the subtrees.

OpenFlight also has the ability to describe a single large model using multiple .flt files on the condition that a master file organizes each of the .flt member files. This allows the reuse of existing parts and the description of complex OpenFlight databases with small patches of terrain, each in an OpenFlight file, combined together using a master file to represent the large area of interest.

### **2.2.3 Newer Sensor Technologies**

New satellite technology is now able to acquire information about the earth's surface with great detail. Also, on ground presence, intelligence and low-level flight scanners using Unmanned Airborne Vehicles (UAV) can acquire even more information from different viewpoints and in high detail. This information is in raw format and would need to be integrated with available terrain data to form a detailed representation and contribute to the appearance of the terrain in a 3D scene.

Extracting features that enhance terrain has always been under research. Common methods include photogrammetry. Photogrammetry is the capturing of images in a stereo view and performing calculations (like shape from shading) to find heights and distances [Wikipedia, 2014] revealing the terrain structure. 3D object reconstruction can then be done for the captured view, even if partial.

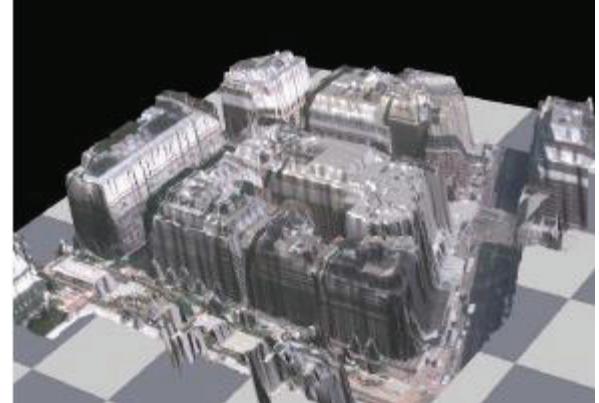


Figure 6: Raw LIDAR Data [Stelle, 2003]

This same technology is now being applied with range accurate sensors to acquire buildings and objects from distance. LIDAR or Light Induced Detection and Ranging is a system which projects a LASER (Light Amplification by Stimulated Emission of Radiation) beam and finds the depth where the laser hits a surface using the reflected ray. Laser technology is very precise and is as fast as light to measure a distance. Therefore, with a simple pass over an area of interest, a 3D point cloud of that area can be

constructed. This point cloud, however, will need editing in order to generate the final 3D model. Editing will include the separation of the acquired objects and the removal of sensor acquisition error due to the mechanics involved and environment conditions. Some of these tasks can be done in a semi-automated manner, but a lot of work still remains and is currently being done manually [Stelle, 2003].

Photogrammetry and LIDAR are not sufficient to describe the internal structures of buildings or cultural aspects but they are very efficient in reconstructing 3D features as they appear from air. Building edges and extremities can be identified and 3D models can be constructed to be later added to the scene over the terrain. This will still need manual input to decide on the type of object scanned and to separate and remove the error in the continuous point cloud to generate the different objects for querying purposes.

## ***2.3 Topological Relations and DE-9IM***

Geometric topological information between features can be represented using well-defined relationships between the geometries representing these features. The Dimensionally Extended nine-Intersection Model (DE-9IM) is an Open Geospatial Consortium standard developed by [Clementini et al., 1994]. It allows the computation of a matrix between two spatial entities that can be used to identify a set of basic geometric topological relationships as well as the dimension of each relationship between them. We use these topological relationships and also extend some relationships to express 3D relations such as over, under and 3D intersection. Our framework uses GeoTools v.9.3 as

an implementation of DE-9IM. Moreover, other topological information about a feature (those not covered by only geometric properties) might also exist e.g. the belonging of an instance to a certain class or group.

All topological information about a feature can be translated into knowledge representation in the form of assertions. We fuse all such representations into the knowledge base. We show how this is done as part of our relationships mapping process.

## 2.4 Semantic Web

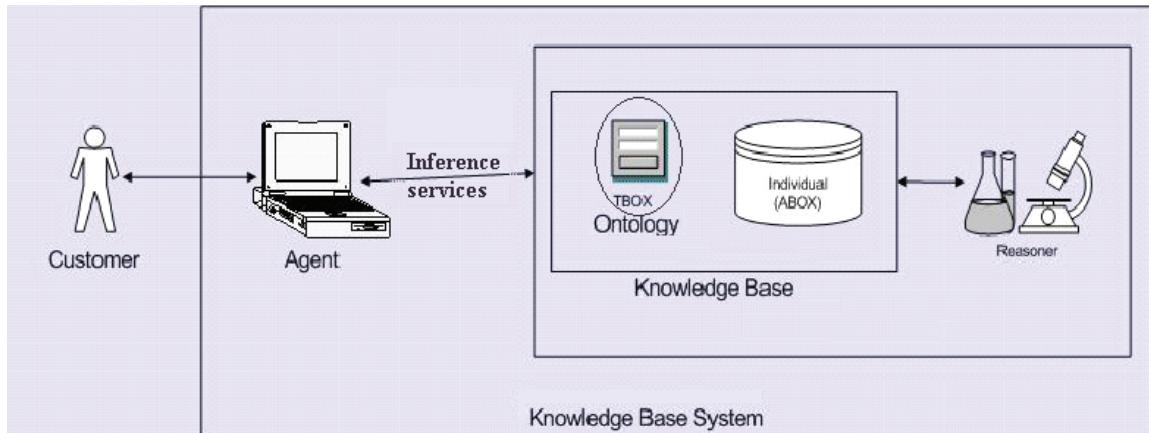


Figure 7: Knowledge Base System Components

The Semantic Web [Berners-Lee et al., 2001], or the web of data with meaning [Daconta et al., 2003], is an active research area which addresses knowledge representation and interpretation. It allows the definition of formal knowledge in the form of ontologies that cooperate with other components as presented in Figure 7. Figure 8 shows the ontology spectrum and the different levels of semantics they can achieve [Baader et al., 2003].

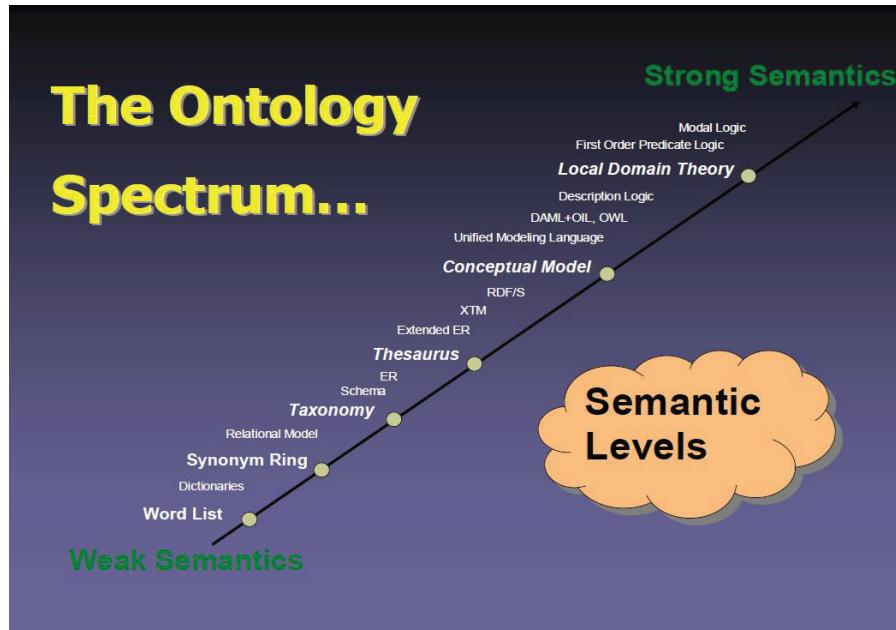


Figure 8: The Ontology Spectrum [Bitters, 2005]

[Hitzler et al., 2011] and [Haarslev, 2014] provide good references covering developments in this technology. The knowledge can be decentralized, but still forms a satisfiable and consistent knowledge base (KB). A terminology box or TBox defines formal knowledge while an assertions box or ABox defines instances of concepts in the TBox (also referred to as Individuals) with memberships and relationships (also referred to as roles). Both the TBox and the ABox define the KB. This KB can then be manipulated and queried using a Semantic Web reasoner, based on description logic, to return results that systems can understand and interpret due to the formal semantics in the TBox. Description logic is a subset of first-order predicate logic which allows formal logic-based semantics, through the definition of axioms, while being more expressive than propositional logic. It is a subclass of first-order predicate logic since it defines some restrictions on relationships (binary relations), to allow further decidability.

The TBox is static for the domain of discourse, while the ABox can be changed dynamically based on the available data. The structure of the Semantic Web allows the separation of formal knowledge and instances. While formal knowledge can be reused, instances can be added, removed or modified. The Semantic Web reasoner provides services to query, analyze, and modify the TBox or the ABox. When the KB is realized through a reasoner service, all the instances are processed and reclassified, by entailment, to their actual specialized subclasses according to the semantics they represent. A description logic reasoner can deduce information based on formal semantics defined in the TBox such as axioms representing equivalence classes, concept or role inclusion, domains and ranges for relationship properties, as well as rule definitions depending on its supported expressivity for reasoning. Ontologies are usually defined in a specific language such as OWL2 [OWL2, 2012], a W3C specification subset of description logic, with a certain chosen expressivity. OWL2 is based on  $SROIQ^{(D)}$  expressivity. Ontologies are used with a compatible reasoner whose capabilities are defined based on the support and reasoning services it provides. Assertions are axioms that define instances and their properties (such as object-object or object-value relationships) under the OWL2 semantics. Figure 9 below modified after [Hudelot et al., 2008] describes  $SROIQ$  semantics under an interpretation  $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ :

Description logics syntax and interpretation

Constructor	Syntax	Example	Semantics
Atomic concept	$A$	Bridge	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Individual	$a$	S253(Champlain)	$a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
Top	$\top$	Thing	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
Bottom	$\perp$	Nothing	$\perp^{\mathcal{I}} = \emptyset^{\mathcal{I}}$
Atomic role	$r$	intersects	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Conjunction	$C \sqcap D$	Thoroughfare $\sqcap$ Bridge	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$C \sqcup D$	Bridge $\sqcup$ Tunnel	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Negation	$\neg C$	$\neg$ Bridge	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Existential restriction	$\exists r.C$	$\exists$ Next_To.Monument	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
Universal restriction	$\forall r.C$	$\forall$ over.WaterBody	$\{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}} : (x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$
Value restriction	$\exists r.\{a\}$	$\exists$ under:{S253}	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x, y) \in R^{\mathcal{I}} \Rightarrow y = a^{\mathcal{I}}\}$
Number restriction	$(\geq nR.C)$ $(\leq nR.C)$	$\geq 2$ Next_To.Monument $\leq 1$ hasBoundingBox.Area	$\{x \in \Delta^{\mathcal{I}} \mid  \{y \mid (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}  \geq n\}$ $\{x \in \Delta^{\mathcal{I}} \mid  \{y \mid (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}  \leq n\}$
Subsumption	$C \sqsubseteq D$	Bridge $\sqsubseteq$ Thoroughfare	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Concept definition	$C \equiv D$	Avenue $\equiv$ Thoroughfare $\sqcap \exists$ Next_to.Monument	$C^{\mathcal{I}} = D^{\mathcal{I}}$
Concept assertion	$a : C$	S252 : WaterBody	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
Role assertion	$(a, b) : R$	(S253, S252) : over	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$

Figure 9: Description Logic Syntax and Interpretation

Under  $SROIQ^{(D)}$ , in addition to object-object relationships, an extension allows the definition of object-value relationships (datatype roles) as well, with its range in the domain of a certain data type based on [OWL Datatypes, 2012] e.g. Literal or Short.

There exists a plethora of tools for defining ontologies, managing knowledge and integrating into frameworks. We use Protégé v.4.3 as an OWL ontology editor, Pellet v.2.3.1 as our reasoner with full support for  $SROIQ^{(D)}$  expressivity along with explanation services support, and OWL API v.3.4.5 for integrating our process into semantic web services. We comply with ontologies from [GeoSPARQL, 2012], an [OGC, 2014] standard since 2012, to allow querying of geospatial datasets under DE-9IM relationships. We also retargeted OWL Link v.1.2.0 to be used with C# and C++. This allows communications into OWL API from other GIS frameworks such as MapWindow GIS v.4.7.5 which only supports extensions in C# plugins.

### **2.4.1 Reasoner Services and Justification**

Semantic web reasoners generally provide multiple services for software agents to operate on ontologies and the knowledge base. Common services under the  $SROIQ^{(D)}$  expressivity include context separation between Boxes (allows scalability, data fusion and easier maintenance), satisfiability (subsumption, consistency, inference, coherence, instance checking, and realization) even on large internet datasets, querying (such as with SPARQL [McCarthy, 2005], defined in the following section) and applying rules, defining annotations, and explanation. In addition, they perform reasoning under the Open World Assumption (OWA) and without Unique Name Assumption (UNA). These, respectively, allow the reasoning to remain neutral in the case of lack of knowledge and always assume that two instances might be the same unless otherwise specified.

Of special interest for this work are the explanation services for  $SROIQ^{(D)}$  entailments as defined by Horridge, Parsia et al. in [Horridge et al., 2008 & 2009]. A certain explanation for any entailment is a minimum set of ontology axioms that, when combined, directly result in that entailment. Moreover, for every entailed axiom in the knowledge base, it is possible to obtain all explanations deriving this axiom.

For example, consider a TBox definition such as:

$$\begin{aligned} \text{Tunnel} \equiv & \exists \text{Through}. (\text{NaturalObject} \sqcup \text{Structure}) \\ & \sqcup \exists \text{Under}. (\text{NaturalObject} \sqcup \text{Structure} \sqcup \text{Thoroughfare}) \end{aligned}$$

where the elements are simple classes and properties defined in the knowledge base. An ABox defines  $\{i2:NaturalObject, i3:Structure, i4:Throughfare, (i1,i2):Under, (i1,i3):Through, (i1,i4):Under\}$

After knowledge base realization,  $(i1:Tunnel)$  is entailed and three explanations given:

- 1-  $\{(i1,i4):Under, i4:Throughfare,$   
 $(Tunnel \equiv$   
 $\exists Through.(NaturalObject \sqcup Structure) \sqcup \exists Under.(NaturalObject \sqcup$   
 $Structure \sqcup Thoroughfare)\}\}$
- 2-  $\{(i1,i2):Under, i2:NaturalObject,$   
 $(Tunnel \equiv$   
 $\exists Through.(NaturalObject \sqcup Structure) \sqcup \exists Under.(NaturalObject \sqcup$   
 $Structure \sqcup Thoroughfare)\}\}$
- 3-  $\{(i1,i3):Through, i3:Structure, (Tunnel \equiv \exists Through.(NaturalObject \sqcup$   
 $Structure) \sqcup \exists Under.(NaturalObject \sqcup Structure \sqcup Thoroughfare))\}\}$

The explanations are a list of ontology axioms that can be queried and analyzed.

Later in this thesis, we will show how we use these in order to calculate the resulting uncertainty value of an entailment.

#### 2.4.2 SPARQL and GeoSPARQL

SPARQL Protocol And RDF Query Language, referred to as SPARQL [McCarthy, 2005], allows the querying of a connected Resource Description Framework

(RDF), a W3C specification, graph representing the KB. Version 1.1 became a W3C recommendation on 21 March 2013. The SPARQL query language is similar to the SQL query language in its syntax and allows formatting and retrieving KB data by pattern matching. It also supports quantitative value testing as part of the query. The result of a SPARQL query is returned in the form of a result set.

GeoSPARQL [GeoSPARQL, 2012] is an OGC standard since 2012. Its name refers to SPARQL as it allows the querying of spatial and topological relations between knowledge base elements using SPARQL. It defines an ontology that formalizes all the DE-9IM relationships as well as equivalent concepts in some of the other topological relations models such as the Egenhofer model [Egenhofer & Franzosa, 1991]. Inheriting from this ontology and using the relationships it defines, the concepts are organized in such a way that they can be queried using topological relations and SPARQL. It basically defines a SpatialObject concept that can be a Feature or a Geometry (or both) meant to inherit from by extending definitions of the General Feature Model, Simple Feature, and Geometry ontologies developed and standardized by the OGC as well. The defined relationships are binary properties relating two SpatialObject instances. It currently does not implement characteristics on properties where relevant and the definition of every property is mostly textual. It serves our purposes well since our features also inherit from SpatialObject and by defining a certain spatial property between two instances in the knowledge base, the properties can be queried and can result in inferences when applicable. Ordnance Survey extends the GeoSPARQL ontology by adding some role properties such as transitivity and symmetry.

## ***2.5 Summary***

In this chapter we have briefly provided all the background concepts essential for the understanding of the work reported in the rest of this thesis. Specifically, we have first described the various GIS sources, their standards, possible transformations of one data type to another, the current state of the art in carrying out these transformations and lastly the positioning of our work in the context of these transformations. Next we have discussed the geospatial relationships among GIS geometric entities and their extensions which are very important in our work on inferring features and their property values from GIS data. Lastly, we have described semantic web technology, which is the primary knowledge based technology we use to create our knowledge base and extract information from it.

# ***Chapter 3***

## ***Related Work***

The following sections summarize an extended bibliography study undertaken in domains of usage of semantics in GIS and the state-of-the-art in scene and image understanding. We also bring out specific readings on producing geospecific information, using different GIS data sets collectively for a specific task, and extracting information from GIS data sets based on semantics.

### ***3.1 On Standards***

Landscapes and GIS data are a primary need for simulation, geological and geographical studies, serious gaming, military and aerospace training, etc. Many studies suggest that researchers and scientists are trying to represent underground cavities, volumes and constitutions of different materials. While many available systems still visualize the area of interest in 2D, 3D allows scientists to better visualize geological data

and understand the information in a spatial and collective manner. This requires standards capable of supporting 3D representations.

A heavy emphasis is placed on standards that represent information collected in the feature extraction process for the purposes of data sharing and reuse. Thus, professionals and organizations are heavily involved in the definition and development of standards for terrain and other similar data sources. However, these standards are not necessarily based on how to represent the detailed 3D landscape. Rather, the information layers are intended for querying purposes. Terrain reasoning has traditionally been limited to spatial terrain queries such as Line of Sight (LOS), raycasts in the path of a player, height above ground, etc... A little more work was done with Terrain reasoning for AI purposes such as [Van der Sterren, 2001] and using A\* path-finding [Hart et al., 1968] algorithms, which can also be extended to 3D. Then major studies to push flexible information representation into standards took place with the emergence of the ESRI Shapefile standard and the ArcGIS database where users could represent more knowledge in the geospatial database such as the work by [Stanzione, 2006] which uses Shapefile information to identify mobility, cover and concealment points.

ESRI Shapefiles have thrived in the definition of surface feature information. However, they lack formal semantics and users can interpret a Shapefile record differently. In most cases, there is no formal semantics behind the classes and properties used in the Shapefile record. Very few have attempted to convert GIS information automatically to equivalent formal knowledge that can be used as a common knowledge base.

### ***3.2 Terrain Modeling Tools***

Our main competition in terms of existing methods and tools is with real-time 3D presentation support provided by existing state-of-the-art tools as depicted in Figure 5. These include systems like Creator Terrain Studio (Multigen-Paradigm Inc.), Model Librarian (ERDAS IMAGINE), and TerraVista (TerrEx). These allow the user to define a ruleset to model the 3D digital environment. These tools define the environment offline because they require considerable amount of user input in the process of building and defining the final 3D environment. They make use of 3D model repositories, manual explicit attributions and rules in order to generate the final landscape model to be rendered by a corresponding 3D Renderer system that accepts their output format. We will discuss each of these tools further below. These systems can create a 3D view of the terrain by applying TIN techniques after points in space are extracted from the elevation information. For other real world objects in the 3D environment, these systems typically require a lot of user involvement to create the 3D representation. They construct the 3D environment by layering the GIS data input layers available and then addressing detail of individual objects primarily with pre-defined rule sets and considerable user input including explicit attribution [McKeown et al., 2007].

Some systems are feature centric while others are elevation centric. This basically determines which layer has more priority in the visual system. For example, should the terrain be matched to the base of a building or should the building be extruded to touch the terrain when these have a small gap? This happens often when the object (say, a building) is not positioned on a flat surface. We consider our methodology to consist

more of a detail-oriented approach, in the sense that the above kinds of decisions are dependent on the actual GIS source data available for the area. Depending on the resolution of each data source, sometimes the elevation information would be altered to match a detailed feature and sometimes a feature would be altered to fit the higher resolution elevation data set available. This information would need to be encoded as part of uncertainty of assertions in the knowledge.

GenesisRT [DVC, 2014] is a state-of-the-art tool that takes GIS source data and dynamically constructs and generates the 3D view of the landscape in real-time. It uses a set of fixed built-in rules that do not require a lot of user-involvement to define, but at the expense of reduced customization and fidelity. The user has a working 3D landscape as soon as the application is started, but not necessarily with details. No pre-processing is needed though. This is a good example of what we mean by high turnaround time. To the best of our knowledge, methods within GenesisRT do not make use of semantics on the input GIS source data. The user has to manually create the input data and explicitly attribute it in order to output the required detail. The user needs to match the properties in the input to a set standardized by GenesisRT. If the needed properties are found, GenesisRT will try to make use of them. For example, if the user specifies in a feature record a property HDG with value 90 (degrees), the system will draw the feature with a 90 degree rotation from its model origin. Aircraft simulators typically use a highly detailed model of the source and destination airports and all the intermediate terrain can be of low level resolution. GenesisRT supports this need by providing the lowest level resolution elevation data, DTED Level 0 (at 1km resolution), for the whole world by default. GenesisRT also provides general airport, navigation aids, and tower listings as

well as coastline data by default. Low level of resolution elevation data would not be sufficient to generate accurate coastlines. Therefore, GenesisRT formulated a way to deform and re-level the terrain in order to match the coastline data available either by default or in higher detail as added by the user. It applies this re-leveling algorithm [Pendris, 2006] to shores, roads and buildings. This augments the scene with more information than what is available in the input source data. The user can install high level of detail areas using high resolution elevation, feature and texture sources.

Shapefiles can point to OpenFlight models to load high resolution buildings or can describe the building using a defined set of parameters and a procedural model can generate it. Typical information like elevation, position, orientation, footprint and type (normally using an aerial shape with some attribution) are needed to generate the building. The type could be used to generate a building with certain characteristics like a texture map using a texture palette. For example, a large industrial building can be considered in some area as either a hangar or a no-window structure. The texture is automatically chosen to meet these conditions from the palette based on time of day and concentration of sets of objects in order to allow for variety instead of uniformity in the generated 3D scene. Another example is that the system automatically generates lights along roads when the road is close to an airport (considered as a populated area) and it illuminates those lights based on the time of day. In and around airfields, it also automatically generates navigation lights and signs, route connections, and leveled areas at a certain distance around airfields and landing pads. Leveling and terrain deformation such as creating craters can be done at runtime using technology adopted from medical simulation (surgery and tissue) [Woodward et al., 2001]. Initial parameters given to

GenesisRT would allow the generation of environmental conditions such as clouds, rain, and storms. It has an embedded astronomical database to populate the sky based on the viewer's position and orientation in the case of a clear sky at night. All of these are dynamic elements that are added at runtime to modify the scene based on hard coded rules in the system and the position of the viewer in the environment. In spite of all the above described capabilities, we consider hard-coded rules as not highly suitable for creating accurate geospecific details as they cannot be easily checked for completeness or consistency.

Model Librarian [ERDAS, 2014] allows the user to define geo-referenced 3D models and maintain them in a repository with proper indexing and attribution. Here again, these models are normally extracted by feature extraction techniques executed on stereo imagery. This is also performed using rule-based mechanisms and manual editing and attribution of the models and their headers in order to add detail when requesting the needed model and building the 3D landscape.

Creator Terrain Studio [Presagis, 2014], CTS, is a 3D model editor based tool that allows users to create, add and modify feature models from a model library in order to construct the 3D model of the environment. The output is a large OpenFlight database with referenced models and textures to create a large area environment. The user cannot define rules in CTS to automate the process of building the 3D environment; rather she/he uses common 3D model editing techniques to build the final 3D environment. CTS is also used to define individual 3D models. It contains a helper wizard concept which questions the user with choice arrays and parameters. This uses procedural generation to create a 3D model close to what the user requires which can then be further

edited and refined. The inconvenience in using an OpenFlight database to construct the 3D environment is due to the need of building/cleaning all the objects manually and adding them to the 3D environment. CTS makes the task of creating a detailed 3D landscape much easier, but the 3D models still need to be constructed by a human-in-the-loop sometimes with several human-years to create the needed environment.

Presagis Terra Vista [Presagis, 2014] allows the user to predefine a ruleset and edit feature properties to model exactly how the final 3D digital environment will be generated. There is a certain overhead for the user to define additional rules, but the resulting output can be highly customized with the ruleset determining the final visualization details. After building the environment, the output is a model in a selected format. The rules are meant to describe what to do with a certain feature with specific properties. For example, if a linear feature record has a property type with value “Secondary Road”, the rule defines the texture and width to use along the linear object and generates the components of the road’s 3D representation. The user has to define all rules based on feature types and properties, and rules are parsed procedurally as listed. The first rule matched will be executed in the build process and will generate the 3D representation. For each feature object in the source data, the rule list is parsed and if a matching rule exists, it is executed. While defining the rules, the user may edit feature record properties to make the task easier. Users normally do so in order to reduce the number of rules to define, usually at the cost of reduced detail. Also, it is common for most feature source data to be delivered without properties, unless numerous hours are spent by experts in attributing the records properly. This feature source data is the output of systems such as BAE SocetSet [SocetSet, 2014] or ERDAS IMAGINE [ERDAS,

2014] which in turn are based on pattern recognition and feature extraction methodologies.

In 2012, Presagis released a new technology named SEGGen Server [Presagis, 2014] developed to create higher definition imagery based on Shapefile information and rules such as, North Africa is mostly desert with change of composition and terrain based on height. The imagery is created based on Shapefile data such as road information and some synthetic additions such as roads, waterways, buildings, etc... just to make the imagery look better but is not precise compared to the real world. There is clearly a challenge in having detail, accuracy and high-automation in a single process and most processes are relying on static rule bases, but with no automatic inferencing.

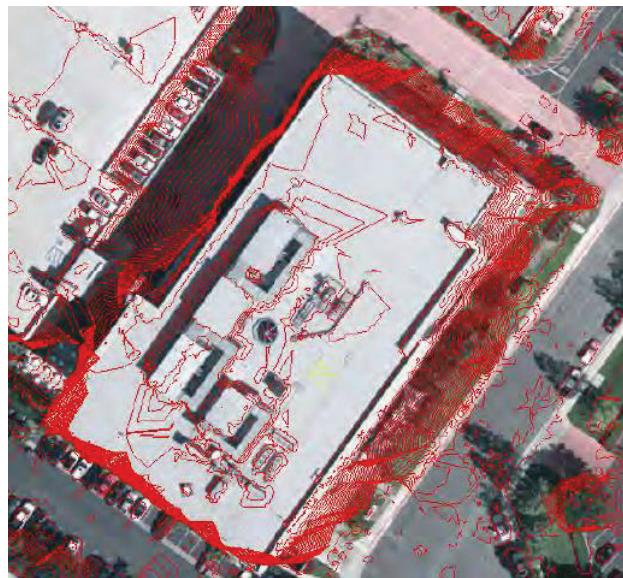


Figure 10: NGATE's Detection of Elevations [SocetSet, 2014]

SocetSet from BAE Systems [SocetSet, 2014] is a toolset using the Next-Generation Automatic Terrain Extraction (NGATE) module to allow the generation of

elevations and features from aerial photographs. NGATE is a tool that processes stereo imagery with very high detail and outputs a 3m detailed elevation model [Zhang et al., 2007]. According to the authors, 3m is enough detail in order to generate any large object like buildings. SocetSet uses the DTM acquired from the NGATE module to automatically identify objects using constant elevation and flat areas. It can then generate the 3D representation using the determined object footprint, elevation values and a generation template (see Figure 10). It then applies a texture to the model based on the image information and some predefined texture palette. The user is required to enter manual information to help in the identification of the footprint. The user also has to define the generation templates and the texture palettes. The final output can be exported using the OpenFlight standard which allows it to be added to a 3D scene. SocetSet is also able to extract general features like roads, parking lots and trees. It can create the corresponding feature maps based on manual entry of missing information about the features of the acquired objects. There is no mention and no references were found that indicate that semantics is or will be part of this technology's roadmap. It requires a lot of user interaction and per-object workflow to define the objects set. It also does not address extraction of information using collective data fusion from multiple source data sets.

Google Inc. has been a major player in the area of mapping and 3D landscape visualization for the past few years. With the creation of Google Maps® and Google Earth® and their open ended integrations available for any user having access to the internet, many personal and commercial centers were able to create useful applications utilizing Google's technology. Google Maps, for example, has an API that would allow the control of how the map is shown and can be used to augment the presented map or

show information in a different style. It provides an API that would communicate with the Google Maps server to query and display existing information. New information can then be added to the local view of the map and the database. Google Earth, on the other hand, can be used to display custom 3D models in its 3D visual environment. The KML standard used by this software and developed by Google Inc. allows the description of 3D objects, their visual appearance, as well as the feature information by meta-tagging. In both products, the custom information added to the objects would be used by the server resulting in an increased accuracy in the query results. We are not aware of any formal semantics used in this process, but some form of semantics is used in order to give better results without the need of detailed meta-tags in the added objects or extensive manual user input. Google uses satellite images and geo-referenced data stored in databases to retrieve information for user's queries. It does not make use of proper GIS data and raw data as in the simulation and earth sciences industries. The emphasis is on speed and on creating a good visual impression on their users, and very little or none on automatically generating high detail 3D representations.

### ***3.3 Methods in Scene Understanding***

At a task that humans do extremely well, computers need the necessary domain knowledge and specialized feature recognition methods in order to recognize a certain set of objects in a certain photograph or more generally, in the presented digital media. In computer vision, visual object recognition and automated feature extraction are always addressed topics. Common techniques used to identify features include filtering and

image segmentation [Sirakov, 2006]. These have proven very effective in identifying roads, railways, power lines over terrain and in some cases even rivers. They are used mostly to select those pictures or media that fit the search query (although specific set training is required), a topic of research that commonly goes by the name content based image retrieval (CBIR). An application of CBIR and automated image feature extraction in the domain of GIS and military satellite reconnaissance is described in [ISAR, 2007] where the US navy is looking for ways to identify threats and targets using real-time satellite imagery of the ocean. They are seeking not only to detect a ship but also to identify its type, model and classification based on a model library and a trained system. The proposal for this research effort was closed in January 2007 and the project is therefore under development. Unfortunately, we cannot learn about the technology being implemented as the details are not available to the public with ITAR security classification. Similarly, Scene Understanding, also known as Image Understanding, is primarily concerned with feature detection and categorization of image representations of scenes. Feature extraction techniques are used in order to extract objects that match a certain criteria based on certainty factor. It is currently an intensely researched discipline but the technology is far from generic and work is still on specific cases and it often breaks when generalized or a new context or image is given. The current push in image based scene analysis in the GIS domain is towards object-based image analysis (OBIA), more specifically, GEOBIA.

Traditional pattern analysis techniques focus on extracting an element and comparing it to a specific ground truth pattern and if certainty is considerable, the image is attributed or categorized accordingly. There is relatively lower focus on extracting the

actual object with its details. In most cases, when an object is suitably identified/categorized, there is no more effort to further process it, such as required for 3D representation. Examples of such work are [Delenne et al., 2006] and [Zhang et al., 2004]. [Jain et al., 1990] describes the process of analyzing a range image as detecting similar adjacent pixels that would constitute parts. Multiple adjacent parts would constitute a feature and multiple features would constitute a specific high-level object. Methodologies focus on detecting objects by making use of Bayesian models on the detected parts or by filtering and making pixels for a set of parts more obvious for detecting a specific object. More recent research addresses the classification and categorization of high-resolution satellite imagery and objects using novel segmentation methods and using other data sources such as stereo photography or LIDAR. Most of these studies focus on segmenting the input image using new methodologies such as landscape metrics that is very different from traditional patch-matrix models. A critique of the patch-matrix model can be found in [Blaschke et al., 2003]. The purpose of segmentation is categorization of the image and is not for creating a detailed landscape representation for use in other applications. New computer vision techniques such as Active Vision try to identify specific objects using a machine learning system in order to categorize the image properly. Categorizing is often at a high-level of what the image is about and is not about detail. The shift from pixel-based methodologies to object-based methodologies was introduced through a sub-discipline of scene understanding of satellite imagery, currently called object-based image analysis or OBIA. This area of research addresses the recognition of objects in images based on detected patterns at the object level instead [Lang & Blaschke, 2006]. It focuses on detecting simple objects

using image rules that will allow the identification of the complex objects [Lang, Albrecht, & Blaschke, 2006]. Using object-based methodologies the segmentation and detection of an object in an image is much more successful when compared to pixel-based approaches. In summary, it segments the image based on a best probability that a certain segment has the same object. It combines segments if the 2 segments are determined to contain the same object. As an example, [Blundell et al., 2006] defines a neural network to fuse and identify objects using parts detected from pixel values. Segmentation would use spatial constraints such as white lines surrounded by grass or asphalt. GEOBIA is a very recent specific of OBIA and addresses scalable satellite imagery as its input instead of general photographic images. The current research focus in GEOBIA is on man-made objects like buildings and roads, and natural tree distribution and forests that can be viewed in satellite photographs.

The use of multiple sources for scene understanding is recommended by many. In particular, [Deng et al., 2006] states that redundant data in different sources would provide less need for user interaction and authentication. Using multiple data sources to analyze images and extract information, some researchers have targeted identifying composition of an image and others have addressed 3D visualization of extracted objects. For example, [De Kok et al., 2006] uses multispectral imagery along with images over years of a certain area to detect tree crowns in an expanding or shrinking forest along with textures to derive the composition of the forest. They retrieve properties that were extracted from the segmentation process as values which are then used as recognized object parameters. [Deng et al., 2006] uses LIDAR in addition to image sequences to provide enough information for a system to automatically extract 3D buildings and roads

from the LIDAR information with accurate segmentation information from the imagery. [Opitz et al., 2006] discusses the extraction of 3D objects from remotely sensed range images from both terrestrial and areal sources.

Semantics in the process of visual object recognition is an active topic of research. [Marszalek et al., 2007] and [Town, 2004] show that, using semantics, higher probabilities for positive matches can be achieved. Also, [Vogel et al., 2007] has attempted scene categorization by sub-region classification to semantic classes. The layout of the semantic classes suggested the category of the scene in the photograph. Classification of the sub-regions is based on processing features like high spatial frequency associated with a semantic feature.

In summary, reasoning on collective information from different sources is not the current emphasis in scene understanding/pattern recognition as their focus is on segmentation, categorization and selection from a photographic image. They do not address the issue of merging the information in the different data sets acquired from different sources such as sensors or surveys nor do they address different data input other than pixel and color information. These systems, including those in specific areas such as computer vision, scene understanding and feature extraction, focus on classification methodologies which restrict the extraction of information to a class from a predefined set. The spatial reasoning mechanisms used in approaches such as GEOBIA are local to a specific dataset. The knowledge is not transferred with the output that will be consumed by other systems. Therefore, some knowledge is lost because of the change in context. We find that collective knowledge would add otherwise unavailable information and therefore help create more detailed representations. Furthermore, the segmentation and

filtering techniques being published only apply to imagery and pixel values; they do not apply to feature and elevation data sets and these datasets do not necessarily come only from satellite imagery feature and elevation extraction as presented in Figure 4. The use of such technologies in our work would find its place in the implementation of data extractors part of the framework defined later in Chapters 4 and 5.

### ***3.4 Applications of Formal Knowledge to GIS data***

Within the GIS domain, over the last few years, semantics has become one of the most prominent research themes, partly aimed at addressing this problem of deriving information that is collectively but not explicitly present in source data. Such concepts as ontology-driven geographic information systems and the geospatial Semantic Web have fuelled a plethora of research in semantic similarity. These topics complement the traditional focus in GIS research, which has dealt primarily with geometric entities, their spatial relations, and efficient data structures. Most recent uses adopt semantics mainly for the reason of integration of the different data sets with regard to GIS related queries. Concepts from different datasets in different systems are being mapped based on meaning. Prior to this, data sets were independent and merging them was all being done manually. Our research focuses on the investigation of the role of semantics in the synthesis of 3D landscape representations from GIS source data. Below are some examples of previous applications of formal knowledge in GIS.

[Brodaric et al., 2002] worked on the reclassification of earth materials for a certain area. This can be considered a form of feature information about a certain area. They used specific domain knowledge of materials in order to classify certain volumes of materials more accurately. However, this was not done using prevailing ontology and knowledge representation standards. Information sharing between different systems is difficult. Creating detailed landscapes was not their focus.

Earth science studies have recently gained interest in formalizing the concepts and definitions and in knowledge sharing between various systems [Goodwin, 2005]. [Wiegand et al., 2007] introduced semantic web to automate geospatial data retrieval using a task-based ontology for immediate response personnel. This technology helps the personnel in accurately finding needed data sources, from the internet, based on the task description. Earth sciences are among the first domains to attempt the use of semantics with GIS source data. Taxonomies are being used for formal definition mapping between different systems. This allows consistent information sharing and proper identification of concepts. Ordnance Survey, Great Britain's National Mapping Agency, for example, is developing an integrated system and ontology to share information consistently between all their systems for querying purposes [Mizen et al., 2005]. They are also using inference services to correctly classify information in response to a query. Due to the formal definitions of concepts, these results can also be used by computer programs. To achieve this, two types of ontologies were created and linked. Bridge ontologies bridging concepts together and Data ontologies linking concepts to specific data formats. These ontologies work together with the actual data in order to provide search queries with needed results; for example, where is the closest mall next to Montreal's Town Hall?

However, this does not address extraction of precise parameter values needed to produce geospecific 3D models in the region.

[Hummel et al., 2008] uses Semantic Web in scene understanding of urban road intersections in image sequences. The use of description logic in this context allowed a more generic approach in detecting types of complex road intersections and to infer information about the involved lanes in the intersection. This information was then used to define and predict movements and restrictions of cars. The paper approaches the problem by TBox definition and ABox dynamic construction, as data becomes available from the image sensor sequences and the land surveying office map. We adopt a similar approach in generating the ABox dynamically, but for defining geospecific 3D models.

[Vanegas et al., 2009] approaches the problem of visualizing simulated urban spaces in the future by inferring on gathered data such as the original street network and aerial imagery. The results augment the original urban layout by subdividing parcels based on rules and inferring urban layouts at any time step based on some user input parameters such as population growth. There is no mention in using Semantic Web technology and although they are inferring on properties for the purposes of visualization, they are not concerned with detailed representations of real world objects.

Barry Bitters, at Florida State University, has been working since 2005 on an ontology classification of all objects in the visual domain [Bitters, 2005]. Bitters mentions that his work has contributed to a 14,000 unique concept taxonomy in the visual objects domain and 1,100 3D models that represent some of these concepts. The result of his work is publicly available on the internet. It is in the form of a visual 3D model

library organized in a taxonomy of Visual Objects, named the Visual Objects Taxonomy and Thesaurus (VOTT). This library defines a comprehensive list of visual objects, their classification and common properties but is not appropriate for inferencing as it lacks formal definitions of concepts. His primary objective was to have a common ground for a concept and definition library for all systems to reference along with their corresponding 3D models to reuse. This work can be useful in a 3D scene generator where feature source data defines concepts and correct models mapped from the concept identifiers can be automatically retrieved. The user defines the feature data, and includes the concept identifier it represents. Bitters also investigated the automatic generation of details in the environment such as selecting textures for road signage based on road layouts and the generation of vegetation based on statistical input [Bitters, 2007]. He also worked on enhancing scene generation based on the probability of missing elements such as mailboxes next to houses or stop signs at intersections [Bitters, 2008]. He did not formulate a process for determining the existence of such objects and admits of a high failure rate in most realistic cases.

[Fonseca et al., 2002] describes the benefits of using ontologies (not necessarily Semantic Web) in order to integrate information of different sources and to determine embedded knowledge in the collective data sources to be used by client applications. The authors define the benefits and provide a methodology for defining and layering semantic systems to achieve needed results. This work is oriented towards finding results to user queries and providing more accurate information that cannot be provided by current GIS database systems. This, however, does not address the specific problems we have with facts extraction, spatial reasoning, and details (specializations) of the 3D representation

of entities in the landscape. The authors also argue that ontologies are needed on different levels such as application, domain and specific source data type. This works well in our case too, as we consider that some semantic rules are general while others are specific to a certain region. The user in this case will only need to alter the specific ontology needed for their area of concern. While their work does show the importance of using ontologies for uncovering embedded knowledge from collective data sources, it does not provide a solution for our problem.

[Arpinar et al., 2006] suggests a methodology to develop GIS ontologies as an extension to the Semantic Web mainly for the purpose of geo-referencing documents such as tasks and data. Their methodology can be used in our case for the same purpose of formalizing definitions of data objects. We plan to use this formal information to derive details about the data objects for use in the 3D representation of the landscape. Again, in this case, while their work introduces the use of semantic web in the GIS domain, they also do not provide a solution to our problem, or anything similar.

In [Kalogerakis et al., 2006] and [Yin et al., 2009], use of ontologies for generating 3D content was mentioned. While the first addresses the generation of building models from architectural drawings, the second uses ontologies to generate graphics content through knowledge-driven visualization. However, neither of the described techniques addresses inferencing based on available data in the knowledge base nor do they address uncertainty in the data as part of knowledge assertions.

We see that, in all earlier work reviewed above, there is no mention about using knowledge in order to fuse different datasets together and allow inference on assertions

for the purposes of creating geospecific 3D models. The above works do show that Semantic Web technology is promising for defining ontologies and semantics in GIS through its inference services. Inferencing allows a certain system to obtain the most specific information of a certain object and its class and properties provided the information is available in the knowledge base. It also offers rule order-independence using description logic, unlike rule-based systems which are hard to maintain and where the first rule matched will have precedence. Also, Semantic Web offers services to check the semantics defined in the ontology and verify its coherence. Importantly, it serves our purpose in the way we would like the system to assert information about instances representing 3D models and their details.

### ***3.5 Uncertainty and the Semantic Web***

As previously mentioned, there are uncertainties associated with GIS data. For our system to be able to transfer such uncertainties to a higher level of processing for resolution by end-application decision or by user, it needs to address uncertainties that are associated with such data assertions and their entailments. This section provides a review of uncertainty models and methods as part of Semantic Web.

In [Poole et al., 2009], the separation of probabilities and meanings is advocated. Our methodology uses similar principles to define knowledge base terminology axioms in a way that allows this separation. Essentially, instead of defining concepts using the terminology hierarchy, classes and property domains and ranges, each definition is

associated a discrete set of conditions which is called the differentia that can uniquely define it from other classes in its scope (there is no need for probability representation under this context). Those conditions are in the form of classes, object-object and object-value relationships. The relationships can also be defined with restrictions on their domain or range under  $SROIQ^{(D)}$ . In the case of classes, the definition would be recursive with the basic constructs being relationships and restrictions. The relationships are asserted with a true or false value in the knowledge base as described previously. But these relationships could also represent a random variable that can depend on probabilities. We consider representing the uncertainty in this context more appropriate.

Uncertainty can be modeled using probability theory, Zadeh semantics or similar logics. Probability theory is not suitable for our needs as we do not deal with events having outcomes from a closed sample space. Others like [Lukasiewicz, 2008] have attempted to use probability theory in semantic web by defining a concept-concept probability interval association such as  $(Flies, Bird)[0.90, 0.95]$  and  $(Wings, Bird)[0.99, 1.0]$  and answering queries such as “if I is a Bird that doesn’t fly what are the values of  $I:(Flies)$  and  $I:(Wings)$ ?” These methods model probabilities in the ontology and attempt to represent imprecise information. In our case, we address uncertainties in the acquired knowledge about instances and their assertions. Consider the following example:

$$Tunnel \equiv \exists Through. (NaturalObject \sqcup Structure)$$

$$\sqcup \exists Under. (NaturalObject \sqcup Structure \sqcup Thoroughfare)$$

The two relationships of *Through* and *Under* are mutually exclusive. An object with a relationship of *Through* might or might not have a relationship of *Under*. Let's assume for an instance  $I$  the probability of occurrence of *Through* with some *NaturalObject* to be 0.5 and that of *Under* with a different object 0.5 as well. Using regular probability theory ( $P_{A \cup B} = P_A + P_B$ ) we have  $P(I:Tunnel) = 0.5 + 0.5 = 1.0$  suggesting that this instance is definitely of type *Tunnel*. The answer here should be 0.5 since either *Through* or *Under* are required to classify  $I$  as a *Tunnel*. For this reason, we use the Zadeh logic based on [Zadeh, 1965] and [Gerla, 1994]. [Bobillo & Straccia, 2011] recently summarized the work of probability and possibility in semantic web as shown in Figure 11. They also discussed the advantages/disadvantages of each family.

Some popular fuzzy logics.

Family	t-Norm $\alpha \otimes \beta$	t-Conorm $\alpha \oplus \beta$	Negation $\ominus \alpha$	Implication $\alpha \Rightarrow \beta$
Zadeh	$\min\{\alpha, \beta\}$	$\max\{\alpha, \beta\}$	$1 - \alpha$	$\max\{1 - \alpha, \beta\}$
Gödel	$\min\{\alpha, \beta\}$	$\max\{\alpha, \beta\}$	$\begin{cases} 1, & \alpha = 0 \\ 0, & \alpha > 0 \end{cases}$	$\begin{cases} 1, & \alpha \leq \beta \\ \beta, & \alpha > \beta \end{cases}$
Łukasiewicz	$\max\{\alpha + \beta - 1, 0\}$	$\min\{\alpha + \beta, 1\}$	$1 - \alpha$	$\min\{1 - \alpha + \beta, 1\}$
Product	$\alpha \cdot \beta$	$\alpha + \beta - \alpha \cdot \beta$	$\begin{cases} 1, & \alpha = 0 \\ 0, & \alpha > 0 \end{cases}$	$\begin{cases} 1, & \alpha \leq \beta \\ \beta/\alpha, & \alpha > \beta \end{cases}$

Figure 11: Popular Fuzzy Logics (After [Bobillo & Straccia, 2011])

We are particularly interested in conjunction (t-Norm) and disjunction (t-Conorm) of two axioms  $\alpha$  and  $\beta$  represented by their certainty values under the Zadeh logic as we think these are sufficient and necessary for our instance class entailments. This logic also extends to multiple variables.

Much of the related work in the domain of possibilistic logic in semantic web is described by Straccia. Furthermore, we couldn't find any actual reasoner that could serve our purpose and address our problem. Some reasoners such as FuzzyDL [Bobillo &

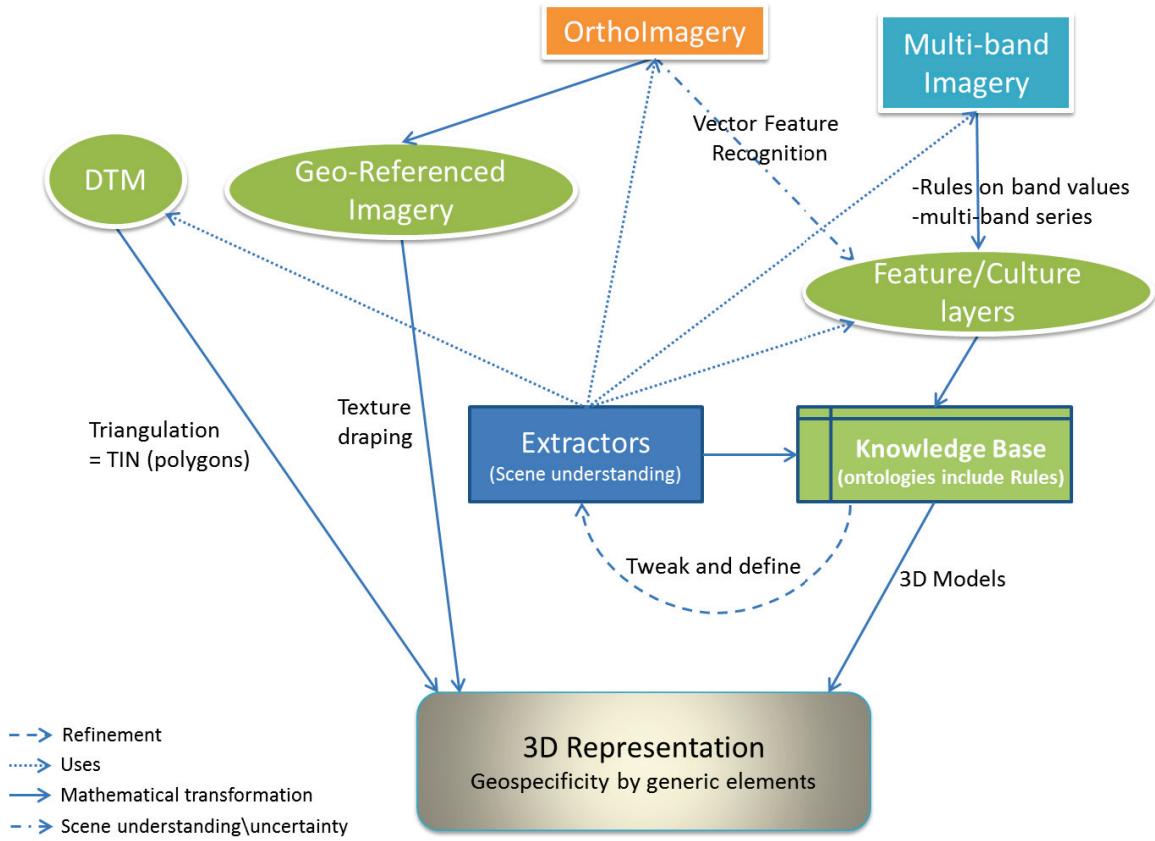
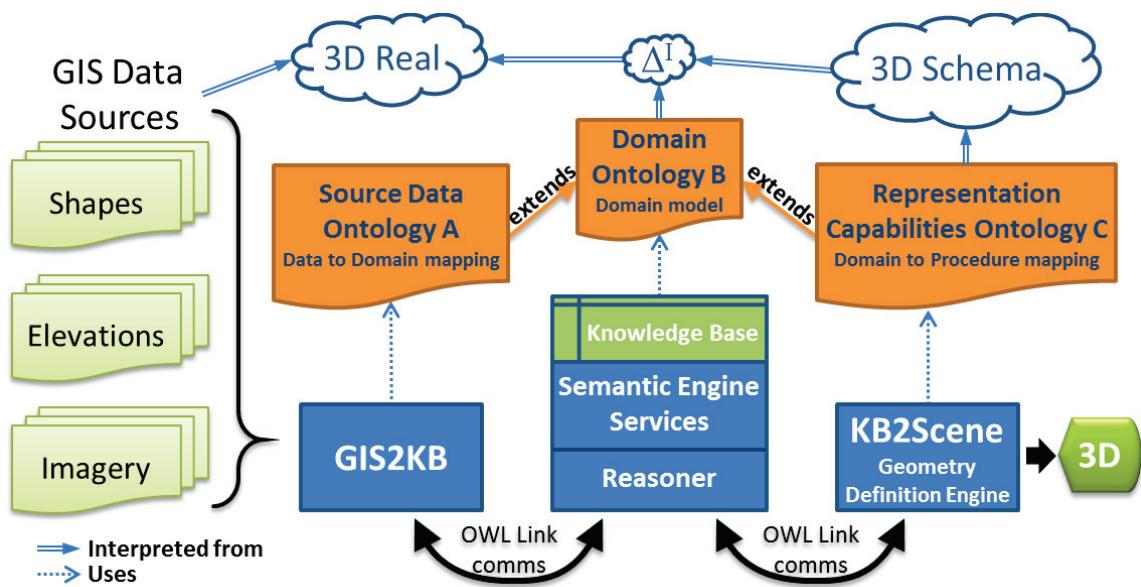
Straccia, 2011], DeLorean [Bobillo et al., 2012] and Pronto [Klinov & Parsia, 2013] are overly complex to what is needed by our process as they try to address the generic problem of uncertainty in knowledge and others were described but not available in the public domain. There also exist some query languages that handle uncertainty by conditional querying. We have opted however to use direct reasoner access rather than querying in our process in order to gain direct access to axioms within the knowledge base. Moreover, we did not find much work in uncertainty under  $SROIQ^{(D)}$  beyond the work of Lukasiewicz and Straccia which mainly addresses  $SHOIN^{(D)}$  but probably can be extended to handle complex role inclusion and qualified restrictions axioms. Even with the availability of such reasoners based on an extended expressivity such as  $P - SHOIN^{(D)}$  (probabilistic- $SHOIN^{(D)}$ ), the complexity class of reasoning is increased to  $FP^{NEXPTIME}$  from  $SHOIN^{(D)}$ 's  $NEXPTIME$ .  $FP^{NEXPTIME}$  is the functional analog of  $P^{NEXPTIME}$  which contains all problems that are decidable in polynomial time on a deterministic Turing machine with the help of an oracle for  $NEXPTIME$  [Lukasiewicz, 2008] where  $EXPTIME \subseteq NEXPTIME \subseteq P^{NEXPTIME}$ .

We, therefore, define our own method of dealing with uncertainty which we will describe later.

## ***Chapter 4***

### ***Knowledge Base and 3D Landscape Creation***

In this chapter, we first provide the formulation of the process for transforming GIS source data (captured through sensing on the real world 3D, refer to Figure 4) for a given ROI into a knowledge base and then the process of creating a detailed 3D landscape representation by querying the knowledge base for 3D entities in the given ROI. We show that, under some assumptions, this process creates a detailed 3D digital geometry definition for every entity present explicitly or implicitly in the input GIS source data sets. We show how an entity is mapped as assertions, how it gets specialized to its specific class and how its property values are extracted to create the detailed representation. Entity specialization is best explained with an example. A linear element entity in the shapefile may get specialized first to a thoroughfare and then to a bridge entity, even if the bridge membership is not explicitly present as a property in the shapefile. We may also refer to entity generalization. For example, a bridge entity may be generalized to a linear element entity.



Our overall process consists of the following stages. The entities defined in Shapefiles are added to the knowledge base as initial instances along with their class, their properties and relationships. This is done using a Source Data ontology, which is defined using the Shapefile's schema and domain elements. The entities in this initial knowledge base are specialized by extracting entity-entity relationships and required property values. This is done using the Data Extractors framework which also addresses spatial and topological relationships and their uncertainties. The knowledge base is modified and by computationally resolving implicit information with the help of rules defined by an ontology (the Domain ontology), explicit information will surface in the knowledge base. This is used to generate geospecific 3D models for that ROI. The above process is summarized diagrammatically and shown in Figure 12. There are three distinct sections in the figure:

- (1) The source data collections (input) on the left side in the figure and labeled GIS Data Sources.
- (2) The ontologies A, B and C shown at the top that service the sub processes, where, ontology B is a common domain ontology that defines the transportation and visible entities domain ( $\Delta^I$ ), and A and C are bridge ontologies that define a mapping between domain and data concepts.
- (3) The semantic web based subprocesses GIS2KB and KB2Scene which interact with and query the knowledge base using the reasoner system. The knowledge base is persistent information as part of the reasoner system.

In order to allow integration using existing mapping tools and a variety of services, we have defined subprocess interactions using the OWL Link protocol [Liebig et al., 2008]. We chose the OWL2 language [OWL2, 2012] with  $SROIQ^{(D)}$  expressivity for our semantic system allowing us to define object-value relationships and restrictions. Figure 13 shows the data transformations in our process compared to Figure 4.

## 4.1 Theoretical Formulation

### 4.1.1 Definitions

- 3D Real: the real world 3D Region of Interest (terrain + entities belonging to ROI)
- $\Delta^I$ : the domain of discourse (3D transportation and visible entities)
- 3D Digital: The set of all 3D digital entities needed to represent the landscape for the given ROI (representing 3D Real).
- $S_{GIS}$ :  $\subseteq$  3D Real. The collection of all sensed GIS source data sets available for the given ROI.  $S_{GIS} = S_S \cup S_I \cup S_E$ , where  $S_S$ ,  $S_I$ ,  $S_E$  denote the collection of all shapefile records, imagery rasters, and elevation rasters respectively.  $S_{GIS}$  is implicitly assumed to be a collection of all the entities needed to represent the landscape digitally. Entities belong to classes and may have properties with values. We will denote the class of an entity  $x$  using the subscript  $c$  ( $x_c$ ) and the properties using subscript  $p$  ( $x_p$ ). The detailed data for defining each entity in the landscape may be available in the shapefile records, but may also be inferred from the collection of data sets.

- 3D Schema: Procedural definitions  $\{G_m\}$  for all possible 3D entities in 3D Real, 3D Schema  $\subseteq \Delta^1$ . Each  $G_m$  is a procedure for generating the detailed 3D representation for a given entity class, with property values, in the domain of discourse. For demonstration of our process, we have chosen the transportation domain.
- An ontology: ontology, say  $O$ , under  $SROIQ^{(D)}$  which includes  $O_C$ ,  $O_R$ ,  $O_P$ , and  $O_A$  denoting set of class, object property (role), data property (datatype role), and annotation axioms respectively.  $A(R(C, R_V), R_e) \in O$  where  $A \in O_A$ ,  $R \in O_R$ ,  $C \in O_C$ ,  $R_V \in O_C$ , and  $R_e$  defines the value of the annotation. Similarly,  $A(P(C, P_V), P_e) \in O$  where  $A \in O_A$ ,  $P \in O_P$ ,  $C \in O_C$ ,  $P_V \in [OWL\ Datatypes, 2012]$ , and  $P_e$  defines the value of the annotation.
- SD: Source Data ontology relies on the Simple Features ontology defined by OGC. The attributes mapping axioms are defined as a subontology. SD provides a mapping from  $S_S$  to instances of TD ( $TD \subseteq SD$ ).
- TD: Transportation Domain ontology under  $\Delta^1$ , relies on the Visual Objects Taxonomy and Thesaurus (VOTT) and spatial relations ontology [GeoSPARQL, 2012]. The properties and extractor annotation axioms are defined as a subontology of the TD hierarchy.
- RC: Representation Capabilities ontology provides a mapping from TD to 3D Schema according to all  $G_m$  definitions.
- TBox: set of ontologies defining terminology axioms.
- ABox: set of assertional (instance) axioms w.r.t. TBox. Each Instance  $I \in ABox$  based on a TBox  $O$  is associated with a set  $I_C$  which defines its class assertions  $\subseteq O_C$ .

$I_R$  defines its object relationships  $\subseteq O_R$ , and  $I_P$  defines its data properties  $\subseteq O_P$ . Each instance corresponds to a unique entity in  $S_{GIS}$ .

- KB: Knowledge Base composed of all axioms in TBox and ABox.

#### 4.1.2 Assumptions

Assumption 1:  $S_{GIS}$  contains a full representation (implicit or explicit) for 3D Real. That is, every entity  $e$  of interest in 3D Real, is present in  $S_{GIS}$ , at least in its generalized form, say  $x$ . Further, there must exist data in  $S_{GIS}$  which supports the derivation of relationships and required property values to specialize  $x$  to the entity  $e$ .

Assumption 2: there exists a set of model definitions ( $G_m$  with parameter values) belonging to 3D Schema such that 3D Real can be represented to the desired detail by setting values for the parameters of every model necessary to represent 3D Real.

Assumption 3: (follows from semantic web technology) an instance  $I \in KB$  always has an equivalent  $I_i \in KB_i$  iff  $KB \vDash KB_i$  ( $KB$  entails  $KB_i$ ) where both are sets of axioms under the same expressivity constraints and every element of  $KB_i$  is entailed by  $KB$ . We also denote that  $KB \vDash I_i$  and  $I_i \subseteq I$ .

Based on the above assumptions, we show that (1) Every entity in  $S_{GIS}$  will be created in the form of an instance in the knowledge base, (2) All required property values for entities in the landscape representation are extracted and available to create the detailed landscape representation, and (3) Using the knowledge base, a 3D digital representation is created for every entity in  $S_{GIS}$  with detail determined by the RC

ontology. It follows that due to the knowledge base containing all information including uncertainties about asserted and entailed assertions, these uncertainties are also available for use in the end-application. We shall discuss our approach for dealing with uncertainty associated with entities, property values and inter-entity relationships in Chapter 6. In the rest of this chapter we will discuss the different mappings needed to create the knowledge base and the mappings needed to create the 3D landscape representation.

An initial mapping using explicit data is performed to create the initial knowledge base. Vector data sources such as Shapefiles define related entities and their properties and, therefore, a knowledge graph could be constructed. More generally, any explicit data source that can be transformed to entities and their properties has a similar associated knowledge graph. Such a graph is converted to an equivalent knowledge base representation based on a compatible ontology which defines the mapping (from the data concepts to the domain concepts). Ontologies for mapping Shapefiles to knowledge base have been attempted by several earlier methods such as [Kim et al., 2013] as well as multiple government mapping agencies like Ordnance Survey, U.S. Geological Survey and Natural Resources Canada. We, however, had to create our own ontology (SD) for the mapping of entities or entity components and their relationships. For example, mapping individual segment parts with connectivity properties to the knowledge base as children of a linear instance results in higher level of detail (and more knowledge) when compared to simply creating a linear instance. This is what allows us to identify that a certain segment is actually part of a different classification such as a bridge.

We then determine entity-entity relationships (object relationships) using geometric computations and add these as object properties to the knowledge base. The

data and object properties added take the form of KB assertions. The knowledge base is then reasoned upon (checked for consistency and realized). ABox Realization ensures that every new assertion that can be inferred from existing knowledge is added to the KB. New assertions may result in a possible entailment of a different specialization class for an instance. For example, a linear element may get classified to a bridge class as new knowledge surfaces in the form of assertions. The knowledge base is then enriched using data extractors for property values and entity relationships (Filling) and realized again. This is repeated until there is no change to the KB. The realized knowledge base (through subsumption compute service) is referred to as  $KB_i$ . We shall describe the formulation of these steps below.

#### **4.1.3 Basic Mapping of GIS Sources**

Given  $S_{GIS}$ , the first mapping step can, in general, be performed on the complete  $S_{GIS}$  collection including all raster data sets. However, we will discuss it specifically for shapefiles. This does not affect the generality of this mapping, since the application of computer vision and scene understanding techniques would result in corresponding additions/modifications to be made to shapefiles.

Consider a function  $s\_mapping$  that converts entities in  $S_S$  to ABox axioms:

$$s\_mapping = \{S_S \rightarrow ABox\}: \forall x \in S_S, \exists I \in ABox$$

where  $I$  denotes the initial instance representing entity  $x$  in  $S_S$ .

**Definition (complete ontology):** An SD (or RC) ontology is said to be complete if it provides a mapping for every entity or entity property in  $S_S$  into the knowledge base (or vice-versa) using a corresponding set of axioms referred to as the instance in the ABox. Similarly, the TD ontology is said to be complete if it defines the required domain knowledge for inferencing as well as all entity relationships and properties, and their extractor associations required for the needed representation.

**Lemma 1 (all entities are mapped):** Given Assumption 1 and a complete SD ontology, all entities in  $S_S$  are retrieved and will be present uniquely in the form of instances in the knowledge base.

**Proof:** Consider an entity  $x$  that is present in  $S_S$  for which an equivalent  $I$  does not belong to KB after applying  $s\_mapping$ . This can happen if, for this entity  $x$ ,  $s\_mapping$  did not create the equivalent axioms in ABox from its representation in  $S_S$ . There are 3 cases to be considered:

1.  $x$  is present explicitly in  $S_S$
2.  $x$  is implicitly present in  $S_S$
3.  $x$  is present in  $S_S$ , but not in an unambiguous manner.

If  $x$  is present explicitly in  $S_S$  and  $I$  is not in KB then it implies that a mapping  $(S_S \rightarrow ABox)$  for  $x$  is not defined; but this is not possible if the SD ontology is complete.

If  $x$  is present implicitly in  $S_S$ , then some mapping, as defined by case 1,  $(S_S \rightarrow ABox)$  would create axioms  $\{A_{S1}, A_{S2}, \dots, A_{Sn}\} \subseteq ABox$  where an inference rule

$(A_{S1} \cap A_{S2} \dots \cap A_{Sn}) \rightarrow I$  in SD ontology should exist. If no  $I$  exists then SD ontology cannot be complete or KB realization is incorrect.

If  $x$  is represented ambiguously where two factual versions exist in implicit or explicit form, then assuming a mapping (by cases 1 or 2) represented as  $S_{S1} \rightarrow ABox$  yields a set of axioms  $A_1 \subseteq ABox$  and another mapping  $S_{S2} \rightarrow ABox$  yields a set of axioms  $A_2 \subseteq ABox$ , then two cases exist where  $a$  and  $b$  are some concepts in the KB:

- a.  $\text{KB} \vDash (a \cap b \subseteq \perp)$  where  $\perp$  is the bottom concept; the axioms contradict (are unsatisfiable) and no interpretation exists based on definitions in SD ontology
- b. Otherwise, the axioms are complementary and are not contradicting based on SD ontology

In case a, ABox Realization will result in a clash which explains the contradiction and allows the user to correct the input. In case b, the result is non-contradicting information about  $x$ . If no  $I$  exists with the appropriate assertions then SD ontology is incomplete or KB realization is incorrect.

If a complete SD ontology exists then a mapping  $\{S_S \rightarrow ABox\}$  exists for every entity  $x$  in  $S_S$ . Under Assumption 1,  $s\_mapping$  creates all instances of entities in  $S_S$ . Depending on its class, each entity will have object properties (spatial relations with other entities) and data properties (values), some of which may be filled by the data already available in  $S_S$ .

**Corollary 1:**  $S_S \subseteq \text{3D Real}$ . No entity representation in 3D Real can be reproduced in 3D Digital based on 3D Schema if it does not exist in  $S_S$ . However, even a generalized definition (e.g. linear thoroughfare) is sufficient.

Our *s\_mapping* function creates knowledge base assertions based on the concepts and relationships. The input of *s\_mapping* is defined as the set of feature records in  $S_S$  and a complete SD ontology. Its range is an ABox with instances and associated properties (filled or unfilled) as part of a KB.

The *s\_mapping* function can be formally expressed as follows (using DL syntax):

$$\begin{aligned} \forall x \in S_S, \exists I \in ABox, ((I, y_v): y_c) \in ABox \setminus I_c = x_c \in SD_C, \forall y \in x_p, \\ ((y_c \in SD_R) \cap (\exists R(x_c, R_V) \in SD) \cap (R = y_c \cap y_v \in R_V)) \\ \cup ((y_c \in SD_P) \cap (\exists P(x_c, P_V) \in SD) \cap (P = y_c \cap y_v \in P_V)) \end{aligned}$$

The SD ontology is then used to map basic spatial concepts to concepts in TD.

#### **4.1.4 Adding Relationships**

In our interpretation domain, relationships between entities are mainly based on topological relations which can be derived from shape geometry, layout, visible precedence in imagery, or profile in elevation. In all these cases, relationship evaluation is independent from inference; the evaluation does not change in the case of a different subclass. When an entity is added as an instance to the KB, it is added on the basis of an initial shape class as point, linear or areal as defined by the Shapefile schema. These are enough to evaluate relationships in our case. We, therefore, only need to process relationships once after initial mapping.

A relationship evaluator  $R_e$  is associated with a relationship  $R$  in the ontology through an annotation on  $R$  such that  $\exists A(R, R_e)$ . It implements a modular method that

extracts the validity of a certain relationship between two entities using  $S_{GIS}$ . The existence of the relationship is extracted by analyzing the topological relations between the two entities using their definitions, geometries and/or images. The relationship between two entities can be defined to exist or can be defined to be nonexistent (different from unknown). Both of these situations are expressed under our expressivity.

The  $r\_mapping$  function is formally expressed as:

$$\forall I_1, I_2 \in ABox, \forall R(I_{1c}, I_{2c}) \in SD,$$

$$\left( ((I_1, I_2): R \cup \neg(I_1, I_2): R) \notin ABox \right) \rightarrow (\exists A(R, R_e) \in SD_A, (I_1, I_2): R_e \in ABox)$$

The domain of  $r\_mapping$  is defined as the set of all KB Instances while the range is a set of KB instances complete with associated relationships. It resolves possible relationships between every two instances in the KB.  $R_e$  returns the relationship or the negative of the relationship as described above. Adding relationships are essential for most inferencing. As described earlier, each definition is associated with a discrete set of object and data properties that can uniquely distinguish it from other classes in its scope.

**Lemma 2 (adding relationships):** All relationships between  $KB_i$  instances are evaluated.

**Proof:** Assume a relationship  $R$  such that  $(I_1, I_2): R \notin ABox$  and  $\neg(I_1, I_2): R \notin ABox$  after  $r\_mapping$ , then either (1)  $R(I_{1c}, I_{2c}) \notin SD$ , (2)  $A(R, R_e) \notin SD_A$ , or (3)  $R_e$  is not properly implemented. Case (1) contradicts the definition of complete SD or TD ontologies. In case (2), the relationship evaluator  $R_e$  violates its definition since no annotation associating it to the semantic relationship it represents exists in SD. In case

(3), the implementation of  $R_e$  is incomplete if the evaluator fails to return the relationship or its negative if they exist in  $S_{GIS}$ .

It should be noted that after  $s\_mapping$  and  $r\_mapping$ , KB will include instances for every entity in  $S_S$  as well as their relationships. Some property values may be missing in some of the instances. These will be filled after data extraction process is completed as described next.

#### **4.1.5 Filling of Property Values**

We define and execute a value filling process for unfilled properties using extractor functions. The task of an extractor is to use the collection of  $S_{GIS}$  sources to retrieve a property value. The data extractors framework uses an incremental discovery approach by solving subproblems (retrieving specific data values) based on procedural techniques applied to other GIS data types, such as imagery and elevation data.

**Definition (extractor<sub>p</sub>):** is defined as an extractor function of the form  $extractor_p(KB, S_{GIS}, I)$  implementation referred to by  $P_e$  associated with a property  $P$  in the ontology through an annotation on  $P$  such that  $\exists A_{xtractor}(P, P_e)$ . It implements a modular method that extracts a specific property value from  $S_{GIS}$ . A value is extracted by analyzing relationships, other KB instances and GIS source data.

**Rule 1:** In order for  $P_e$  to be able to retrieve a property value,  $P$  and  $P_e$  need to be defined and associated in the ontology (ontology is complete) and a corresponding procedural implementation for  $P_e$  must exist.

By this rule, for every unfilled property of an entity, for which a corresponding assertion is not present in ABox, extractors will retrieve the property value. Some example extractors will be presented in Chapter 5.

Let us recall that if  $S_{GIS}$  does not contain the required data to extract a value for the property then Assumption 1 is broken.

**Rule 2:** No property representation in 3D Real can be reproduced in 3D Digital if this property value does not exist explicitly or implicitly in  $S_{GIS} (\subseteq 3D\ Real)$  or if an associated implementation for its extractor does not exist.

**Definition (missing property value):** A missing property value  $x_v$  for a property  $x$  is said to be missing iff  $((I, x_v): x_c \notin ABox)$ .

Missing property values are filled using the following *p\_filling* function:

$$\begin{aligned} & \forall I_o \in ABox, \exists I \setminus KB \vDash I, I \subseteq I_o \\ & \forall P(I_c, P_V) \in TD, \exists x \in I_p \setminus x_c = P, P \in TD_P, \\ & ((I, x_v): x_c \notin ABox) \rightarrow (\exists A(x_c, P_e) \in SD_A, (I, P_e): x_c \in ABox) \end{aligned}$$

The domain of *p\_filling* is defined as the set of all KB Instances while the range is a set of KB<sub>i</sub> instances with no missing property values. *p\_filling* is executed iteratively. In each iteration, some missing property values are filled in and the knowledge base realization procedure is carried out. Realizing the knowledge base results in a possible entailment of a different  $I_c$  class for the instance depending on the restrictions defined in the ontology. Since each class might be defined with a different set of properties, the additional

properties are filled by realizing and filling again until the knowledge base is realized and no missing property values remain.

**Lemma 3 (filling missing properties):** There is no instance in  $\text{KB}_i$  for which property values are missing.

**Proof:** assume a property  $x$  such that  $(I, x_v): x_c \notin ABox$  after  $p\_filling$  where  $x_c$  represents the property class and  $x_v$  represents the value, then either (1)  $x_c(I_c, P_V) \notin TD$  where  $P_V \in [\text{OWL Datatypes, 2012}]$ , (2)  $A_{xtractor}(x_c, P_e) \notin SD$ , (3)  $P_e$  is not properly implemented, or (4) knowledge base realization was incorrect. Case (1) contradicts the definition of a complete TD ontology. In case (2), the definition of extractors is violated since no annotation associating an extractor function to the property exists in SD. In case (3), the implementation of data extractors is incomplete if there is no extractor which returns a valid value even though the value exists in  $S_{\text{GIS}}$ .

More generally, when a value exists, data properties attach the value to the corresponding instance using a property definition with a value type. Under Assumption 3, a data property  $p \in I ((I, p_v): p_c \in ABox)$  has  $p_c$  representing the property class and  $p_v$  representing the value. While  $p_c$  is associated with the class definition of  $I (I_c)$ ,  $p_v$  is available under one of the following situations:

1. Explicit in only one of the source data sets  $S_S$ ,  $S_I$ , or  $S_E$
2. Implicit or derived from one or more data sources
3. Represented (implicit or explicit) in  $S_S$  as well as another source ( $S_S$ ,  $S_I$  and/or  $S_E$ )

We prove for each of the cases above that if a value exists in  $S_{GIS}$  but is not available as part of the ABox after completing the  $s\_mapping$ ,  $r\_mapping$  and  $p\_filling$ , then a contradiction happens in either the assumptions or the definitions of the system components.

1. If  $p_v$  is represented explicitly in input set S ( $S_S$ ,  $S_I$ , or  $S_E$ ) then a mapping of object properties ( $S \rightarrow ABox$ ) for  $p_v$  is not defined. If no  $p_v$  is available after mapping  $((I, p_v) : p_c \notin ABox)$ , then the SD ontology is incomplete or a mapping to TD is not defined as shown in the proof of Lemma 1.
2. If  $p_v$  is represented implicitly by 1 or more data sets in  $S_{GIS}$ , then some function  $P_e$  (data extractor) must be available to retrieve  $p_v$  such that  $\exists p_c (I_c, P_V) \in TD$ ,  $A_{extractor}(p_c, P_e) \in SD$  and  $(I, P_e) : p_c \in ABox$  ( $p_v$  defined by running  $P_e$  using  $I$ ). If no  $p_v$  is available after mapping then the data extractor implementation is incomplete.
3. If  $p_v$  is represented ambiguously where two factual versions exist in implicit or explicit form, then assuming a mapping (by cases 1 or 2) yields a set of property axioms  $A_1 \subseteq ABox$  and another mapping yields a set of property axioms  $A_2 \subseteq ABox$ , then two cases exist where  $a$  and  $b$  are some concepts in the KB:
  - a.  $KB \vDash (a \cap b \subseteq \perp)$ ; the property axioms contradict (are unsatisfiable) and no interpretation exists based on definitions in ontology
  - b. Otherwise, the property axioms are complementary and are not contradicting based on definitions in the ontology.

In case a, ABox Realization will result in a clash which explains the contradiction and allows the user to correct the input. In case b, more information or non-contradicting information about  $I$  is resulting. If no  $p \in I$  exists after mapping then

the ontology is incomplete (data property restrictions or mappings are ill-defined) or does not comply with source data.

#### **4.1.6 Representation Creation**

Consider a function  $v\_mapping$  that creates all  $G_m$  models for 3D Digital using a 1-to-1 mapping from instances in the knowledge base:

$$v\_mapping = \{I_i \rightarrow G_m\}: \forall I_o \in ABox, \exists I_i \subseteq I_o \setminus KB \models I_i, m = I_{ic}$$

By Assumption 2, every instance  $I_i$  entailed from the knowledge base (Assumption 3) has a model definition  $G_m$  from 3D Schema for its class  $I_{ic}$ . Models are retrieved for every instance in ABox as follows:

$$\begin{aligned} & \forall I_i \in KB_i, KB \models KB_i, I_{ic} \in RC_C, \exists V \setminus (\forall (I_i, x_v): x_c \in ABox), x_c \in RC_P, x_v \in V, \\ & \exists G_m \setminus m = I_{ic}, V \in G_m \end{aligned}$$

It should be noted that models are retrieved only for the most specialized definition. That is, if a linear element instance is specialized to a covered bridge instance, then only the model for a covered bridge instance will be retrieved. The domain of  $v\_mapping$  is a realized KB ( $KB_i$ ) using the TD ontology with ABox instances, and the RC ontology representing the 3D Schema. Its range is a set of 3D graphical models definitions  $G_m$  for use in procedural generation of the entities in the landscape representation.

**Lemma 4 (3D model for every instance in  $KB_i$ ):** After  $v\_mapping$ , every instance in  $KB_i$  will have a detailed model in the landscape representation.

**Proof:** If there exists an instance  $I_i$  for which there does not exist a detailed model in 3D Digital, then this can only be for the following reasons: (1) a 3D model definition  $G_m$  does not exist for this instance, but this violates assumptions 2 and 3 (if instance is unsatisfiable), and (2) a property or its value needed for creating  $G_m$  is not available, this is not possible by Lemma 3.

We can now state our main theorem as follows:

**Theorem 1 (Detailed 3D Representation Creation):** Given  $S_{GIS}$  for a given ROI, complete ontologies (SD, TD and RC), data extractor implementations for properties of all entities in  $S_{GIS}$  and a 3D Schema complete with respect to entities in  $S_{GIS}$ ,  $s\_mapping$ ,  $r\_mapping$ ,  $p\_filling$  and  $v\_mapping$  will together create a detailed landscape representation of the given ROI.

**Proof:** Follows from lemmas 1 to 4 above, since every entity in  $S_{GIS}$  is mapped into the knowledge base as a unique instance ( $s\_mapping$ ), with all its properties filled with values ( $r\_mapping$  and  $p\_filling$ ), and detailed 3D geometric models for every instance are provided by 3D Schema ( $v\_mapping$ ).

#### 4.1.7 Thoughts

We assign values to properties by executing data extractors for every unfilled property in every instance in the current state of the knowledge base. These property values in turn get added to the knowledge base in the form of suitable assertions with that instance. It results in the advantage that properties also get assigned by KB inference. For example, consider two property classes defined as equivalent in the TBox with the

appropriate restrictions. If one property is defined with a value, the other property would point to the same value. If two different values are inserted for each of the properties, a contradiction would occur upon realization if a complete SD ontology is defined. In addition, data checking in the form of data property restrictions and type inference based on those is available under  $SROIQ^{(D)}$ . Ambiguity, redundancy and wrong or non-existent source data would be handled in a similar fashion. It is for this reason that we chose to put the data values as part of the KB rather than simply put a link to the values.

## ***4.2 Evolution of our Method***

Throughout our research we have changed and refined our methods to achieve our objectives while adapting to the technology's capabilities and benefits. The core of our research has been the use of the semantic web technology to enhance the feature extraction process, fuse and reason on different data types together, and attempt to further automate the resolution of ambiguities and inconsistencies in the data. In the following sections, we show our earlier versions of algorithms and attempts. We first describe the generic legacy process in order to compare with our methodology.

### ***4.2.1 Legacy Processes (for comparison)***

A knowledge base is not used in the legacy process. If reasoning is at all performed, it is done based on rule-based processing that modifies the shape geometry or the TIN based on the shapes created. Tools described in section 3.2 do not deal with

knowledge extraction and associated uncertainties. Most legacy processes take a triangulation, texture it and add the shape geometries forming the final visualization as in the following definitions:

$$SHP = \bigcup_e^L s(e) \rightarrow shapeGeometry \quad TIN = \bigcup_e^L r_h(e) \rightarrow DG \quad TEX = \bigcup_e^L r_i(e)$$

$$VIS = TIN \cup TEX \cup SHP$$

where  $e$  is a layer in  $L$ , the list of all GIS data source layers available related by positional and coordinate systems. A layer is either in the form of a shapefile or raster (imagery or elevation data).  $s(e)$  denotes a features shape layer  $e$ .  $shapeGeometry$  is a list of visual geometry objects generated based on procedural algorithms and user-defined rules on the properties and definitions in  $s(e)$ .  $SHP$  represents the generated geometric meshes representing all features.  $TIN$  generates a triangulated mesh based on a Delaunay Graph.  $r_h(e)$  denotes a single selected height or elevation layer  $e$  in  $L$  for each area in the ROI.  $DG$  denotes the Delaunay Graph generated from all points in the selected elevation layers (forming the terrain).  $TEX$  generates the textures from all imagery data layers.  $r_i(e)$  denotes an imagery data layer  $e$  in  $L$ . The final visualization  $VIS$  is the triangulated graphics mesh  $TIN$  textured with  $TEX$  with all graphics meshes from  $SHP$  added.

We initially investigated two different methods for extracting the necessary information and adding it to the knowledge base. Vector data is always transformed based on the data ontology definitions and added to the knowledge base. The two methods we explored are the following:

- (1) extract all semantics from raster data and insert all resulting data as part of the knowledge base.
- (2) add only the vector data to the knowledge base and use specific data extractors that use context and relevant raster imagery to extract missing data.

However, after further analysis and experimentation, we found that (1) introduced a lot of redundancy and was non optimal and we have improved on (2) to provide a generalization to define and implement specific extractors which associate the knowledge base data property definitions to the actual extraction procedures.

#### **4.2.2 Attempt 1: Extracting all Semantics from Raster Data**

Our first attempt was to model all data as part of the knowledge base providing all required information about a scene as one single knowledge base [Eid & Mudur, Feb. 2009]. This mostly avoids having to shatter related information in different data structures. We attempted to insert all data as part of the knowledge base including all semantics available in raster data:

$$KB = \bigcup_e^L (s(e) \cup r(e)) \rightarrow shapes$$

*shapes* is a list of instances, generated from data in the shape or raster layers, and are related by semantic information in the KB based on the ontologies defined. *KB* can then be realized after this process. The algorithm is defined as follows:

```

Input:  $S_{GIS}$ , SDOntology
Output: KB
KB = {};
For every element in  $S_{GIS}$ 
    shapesLayer = {};
    If isRasterLayer(element) then
        shapesLayer = new shapesLayer(extractFeatures(element));
    Elseif isShapeLayer(element) then shapesLayer = element;
    Else error("list element not recognized", element);
    Endif
    If shapesLayer != {} then
        mapShapefile(shapesLayer, SDOntology, KB);
    Endif
End
EndProc

```

In this case, the *extractFeatures* function is a bridge to another subsystem that extracts all semantics from a given layer. Extracting feature semantics from imagery and elevation data sets is highly complex and is mostly done iteratively by domain experts in cartography. The results of these activities are usually saved as feature data (most commonly in vector Shapefile format) that is then consumed by other systems. Raster data sets are large and the features to be extracted include all the entities and their properties as required by the application domain. Further, efficient implementations for reliable feature segmentation and recognition are in general difficult to obtain, and image analysis/pattern recognition was not the primary goal of our research.

#### **4.2.3 Attempt 2: Linking KB Objects to Relevant Raster Data**

As part of this next attempt, we focused on using the raster layers after the initial knowledge base realization process. We defined a feature data processors mechanism to be part of our method. In order to have the specific extractors operate on a constrained input data set, we attempted to optimize and link the semantic knowledge base data with the relevant raster information and layers. We created data properties in the ontology to

serve as data pointers to the relevant raster data. Relevant raster data is defined as the data within a subarea surrounding the instance. Users can then use the pointers to operate on the relevant data set. This process is summarized as follows:

$$\begin{aligned}
 KB1 &= \bigcup_e^L s(e) \rightarrow shapes \\
 KB2 &= \bigcup_{r(e), n=0}^{L, N} \bigcup_c^{i(KB1)} (rdp_n(c) \rightarrow subarea(d(c), r(e))) \\
 KB3 &= \bigcup_x^X \bigcup_{p(c) \neq nil}^{KB2} (p(c) \rightarrow x(c)) \\
 KB &= i(KB1) \cup KB2 \cup KB3
 \end{aligned}$$

$KB1$  represents a knowledge base containing all feature data.  $KB2$  adds a list of raster data property pointers corresponding to the number of raster layers available to each instance in  $KB1$ .  $n$  is used as a counter for raster data property pointers.  $c$  is a knowledge base instance in the realized knowledge base  $i(KB1)$ .  $rdp(c)$  represents a raster data property that points to a subarea in the raster data. Each raster layer  $r(e)$  would have an equivalent  $rdp(c)$  denoted as  $rdp_n(c)$  where  $n$  denotes which raster layer it came from. The subarea raster is extracted by the  $subarea$  function using  $r(e)$  and the extents of  $c$  denoted as  $d(c)$ .  $KB3$  extracts the missing data properties by running the data extractors on the raster data property pointers list for every instance in the knowledge base.  $x$  is a data processor in the list of data processors available ( $X$ ).  $p(c)$  is a property of instance  $c$  in  $KB2$ .  $x(c)$  runs the data processor  $x$  on the instance  $c$  and attempts to extract a value for  $p(c)$ . The final knowledge base  $KB$  contains all feature data as well as the extracted data properties from the raster data layers. The algorithm is defined as follows:

```

Input:  $S_{GIS}$ , SDOntology, ExtractorsList
Output: KB
KB = {};
For every element in  $S_{GIS}$ 
  If isShapeLayer(element) then
    mapShapefile(layer, SDOntology, KB);
End
ConsistentKB(KB);
For every element in  $S_{GIS}$ 
  If isRasterLayer(element) then
    n = element.layerID;
    For every instance in KB
      // retrieve geo location extents of instance
      di = instanceExtents(instance);
      // extract a sub raster from the raster layer element
      pRaster = subarea(di, element);
      if pRaster != null then
        // add new raster data property
        instance.insert(new raster_dp(n, pRaster));
      Endif
    End
  Endif
End
For every xtractor in ExtractorsList
  For every instance in KB
    For every dataProperty in instance
      If dataProperty = null then
        // runs extractor with instance as input (containing all pertaining data)
        xtractor(instance);
      End
    End
  End
EndProc

```

We did not pursue this method further, since even with a small subimage, reliable implementation of detail extraction from raster images is a difficult problem.

#### **4.2.4 Attempt 3: Running All Extractors for Every Missing Property**

This method does not segment the layers into subparts. After adding all shapes and properties to the knowledge base, it iterates through the source data layers and runs the missing data property extractors on each. This has a limitation of not allowing the data property extractors to use the complete list of layers available and to combine

different layers in the processing to find the necessary new information. It also introduces a lot of redundancy in the extractors processed since for every missing property, the system runs all possible extractors for this property. The algorithm is defined as follows:

```

Input:  $S_{GIS}$ , SDOntology, ExtractorsList
Output: KB
KB = {};
For every element in  $S_{GIS}$ 
    If isShapeLayer(element) then
        mapShapefile(layer, SDOntology, KB);
    End
    ConsistentKB(KB);
For every element in  $S_{GIS}$ 
    For every instance in KB
        For every dataProperty in instance
            If dataProperty = null then
                For every xtractor in ExtractorsList
                    // runs extractor with instance as input
                    xtractor(element, instance);
                End
                ConsistentKB(KB);
            Endif
        End
    End
End
EndProc

```

#### **4.2.5 Final Solution: Running only Relevant Extractor Functions**

We found that extractors often need several data layers to get the best information available. For this reason, we have generalized Attempt 3 to create our final version. This also mimics a human-like divide-and-conquer property filling process by analyzing the available sources to extract the missing property value. The main difference with other methods is the creation of a framework by generalizing the extractor functions and the required inputs, as well as associating an extractor definition per data property as part of the ontology. With this modification, only extractors for the missing properties are executed on the available raster data layers. In addition, the source data list is filtered and

sorted based on each extractor's requirements (e.g. on imagery, on elevation, on multi-band, or on KB). This process allows extractors to be modular and objective driven using the instance definition as constraint.

$$\begin{aligned}
 KB1 &= \bigcup_e^L s(e) \rightarrow shapes \\
 KB2 &= \bigcup_{p(c) \neq nil}^{i(KB1)} (p(c) \rightarrow extractor(x(p), L, c)) \\
 KB &= i(KB1) \cup KB2
 \end{aligned}$$

$KB1$  represents a knowledge base containing all feature data.  $KB2$  extracts the missing data properties by running the data extractors on the available raster data layers  $L$  using the KB instance data  $c$  in  $i(KB1)$ .  $p(c)$  is a property of instance  $c$  in  $KB2$ . The *extractor* function runs the extractor definition  $x$  for the property  $p$  using  $L$  and  $c$  and attempts to extract a value for  $p(c)$ . The final knowledge base  $KB$  contains all feature data as well as the extracted data properties from the GIS source data.

In the next chapter, we will present algorithmic and implementation details for the different mappings formulated in section 4.1.

# ***Chapter 5***

## ***System Implementation***

In this section, we present our implementations of *s\_mapping*, *r\_mapping*, *p\_filling* and *v\_mapping* presented earlier. While *s\_mapping*, *r\_mapping*, and *p\_filling* are implemented as part of the facts extraction step (GIS2KB), *v\_mapping* is implemented in the spatial knowledge extraction step (KB2Scene).

We assume that the main set of entities in the landscape would be present in the feature layers (Shapefiles) and these layers would have been created previously using available techniques and tools. The user need not spend a lot of time precisely defining and attributing each entity found (as required by most state-of-the-art systems). Rather, high-level entities are processed (such as lines cast along a road path) by automatically extracting some facts using extractors and inferring information based on the collection of all available objects. This process would especially help if the data is not well-defined on input or some information is implicit. The user, therefore, does not have to address all possible cases or study the GIS sources extensively to be able to create detailed 3D models.

GIS2KB includes the Shapefile data mapping, relationship mapping and the Data Extractors framework. While the mapping adds initial data to the KB, the framework finds values for missing KB properties. GIS2KB owns and maintains the knowledge base that KB2Scene uses. For this reason, it also extends knowledge base services. It uses the Source Data (SD) and Transportation Domain (TD) ontologies. We chose the Pellet reasoner [Pellet, 2014] implementing OWL2 language with  $SROIQ^{(D)}$  expressivity. KB2Scene uses the Representation Capabilities (RC) ontology of parameterized 3D models to map domain KB assertions to entity definitions with property values. This is achieved by querying the KB using SPARQL and OWL Link. Procedural model generation is then used to create 3D models, which correspond to the KB definitions.

## 5.1 The Ontology Hierarchy

The ontologies we define are based on semantic web ontologies for knowledge bases implementing the OWL language [Bechhofer et al., 2004] and the latest OWL 2 specification [OWL 2, 2012] including datatype restrictions in  $SROIQ^{(D)}$  defined in section 2.4. It allows describing knowledge as terminology and assertions. **Instance** (Object) assertions can be associated with axioms of type **ObjectProperty** (object-object relationship with range in a certain object class) and **DataProperty** (object-value relationship with range in a certain XSD Datatype class [OWL Datatypes, 2012]). We organize our ontologies in two types: domain ontologies and data ontologies.

We have defined domain knowledge ontologies based on knowledge from the domains of natural and environmental objects, man-made features, as well as the visible objects domain. Environmental knowledge defines concepts such as “a Tunnel is a subsurface Entity”. Relationship properties are based on topological relationships between instances such as spatial relations. Bitters' VOTT [Bitters, 2005] organizes elements in the domain of natural and man-made objects. Our domain ontologies are inspired by Bitters' work as a classification basis and implement a subset of VOTT concepts enriched with rules for the purposes of inference. The rules are based on man-made features knowledge such as the laws and principles used by civil engineering. For example, Bitters defines the following in a plain text taxonomy where the underlined elements are keywords that have their own respective definitions in the same taxonomy:

- Street: Is a Road in a BuiltUpRegion.
- Boulevard: Is a broad divided Street in a city often with a wide median, especially a Street laid out with trees and gardens, etc.
- Bridge: Is the subclass of LandTransitways that are artifacts used for crossing water or air-filled gaps that could not be transited over a natural surface.

Bitter's work is intended as a form of standardization of the concepts which allows humans to properly interpret them. Our implementation extends these definitions with properties and rules that allow formal descriptions and inference. Our versions of the above concepts include axioms in the TBox such as the following:

$$\text{Street} \equiv \text{Thoroughfare} \sqcap \exists \text{within}. \text{BuiltUpArea}$$

*Avenue*  $\equiv$  *Thoroughfare*  $\sqcap$   $\exists$  *Number\_Of\_Lanes*. (*xsd:integer*:  $\geq 2$ )

$\sqcap \exists_{\geq 1} \text{NextTo}. (\text{Monument} \sqcup \text{Park})$

*Boulevard*  $\equiv$  *Avenue*  $\sqcap$   $\exists$  *Midsection\_Type*. {"WideSeparator"}  $\wedge$  *xsd:ID*

*Bridge*  $\equiv$  *Thoroughfare*  $\sqcap$   $\exists$  *Over.ImpassableArea*

The intention is to use these definitions as part of the Semantic Web System where an instance of type *Thoroughfare*, for example, could be re-classified as *Boulevard* or *Street* based on the definitions presented. Moreover, data properties were added to include further information about an object's definition. For example, the *Bridge* concept inherits some properties from *Thoroughfare* such as its width, type, number of lanes and pavement type and adds some new data properties such as DeckThickness related to this concept. Although our definitions are simpler (in expressivity and semantics) than the definitions in Bitters' VOTT (they are restricted to elements that can be detected through source data and that represent discrete objects or property states), they are more formal and allow automatic inferencing with our process. We could have defined *Avenue* with object relationships to *Lane* objects instead of the *Number\_Of\_Lanes* data property. But in this case, we only required a data property restriction and the value is available as a property on the *Avenue*. We would only needed object properties if we require the definition of actual *Lane* objects with corresponding restrictions and properties.

Domain concepts are general to the domain and they seldom change except for very different regions or principles. The input GIS source data however changes often such as for different GIS areas or due to source standard or custom data. This data needs to be mapped to the proper properties in the knowledge domain. For example, users and systems define several properties per set of Shapefile records depending on needs or

capabilities. These properties need to be mapped to the domain knowledge, when applicable. In order to minimize the impact of change, we have put in place a hierarchy of ontologies where only a subset would be affected. With such a design, data mapping changes do not impact domain knowledge and vice versa. This context separation allows for maximum reuse and comprehension and minimum impact of change.

Three ontologies were defined (please see Figure 12): (1) a general domain knowledge ontology B called the Transportation Domain (TD) ontology, (2) a bridge data ontology A, called the Source Data (SD) ontology, for GIS data mapping from data concepts to domain concepts in the TD ontology, and (3) a bridge data ontology C, called the Representation Capabilities (RC) ontology, defining the models and properties in 3D Schema. In our case, the SD ontology defines complex mapping and restrictions between data and domain concepts while the RC ontology performs a one-to-one mapping between entity classes and procedural models. The TD ontology also uses OGC's GeoSPARQL and the OrdnanceSurvey's SpatialRelations ontologies for spatial relations based on the DE-9IM model. Moreover, we have enhanced the definitions of the spatial relations to include role inclusion axioms such as transitivity and inverse relationships.

### ***5.1.1 Ontology of Source Concepts Mapping (SD)***

The SD ontology uses OGC's Simple Feature ontology as a basis since our implementation primarily targets the mapping of Shapefiles as discussed earlier. The mapping algorithm is described under section 5.3.1 below. Moreover, the SD ontology defines data extractors as annotations on TD ontology properties. These allow the system to use data extractors to retrieve property values from sources that don't have a mapping

defined. For example, a property *shapeWidth* in the input data is defined as the sum of *roadWidth* and *sidewalkWidth* (the total road way width for road instances) in our domain ontology; a mapping needs to generate both values. In our SD ontology, we annotate the *roadWidth* and *sidewalkWidth* TD properties with data extractor function identifiers such as, *patternImageryWidth*:

$$\exists A_{xtractor}(roadWidth, patternImageryWidth)$$

$$\exists A_{xtractor}(sidewalkWidth, patternImageryWidth)$$

Given the GIS location from the shape definition, a scan vector (calculated from the shape's normal at that location) and a scan width (limited to the *shapeWidth* property by default) as input, the extractor can assign values for *roadWidth* and *sidewalkWidth* by analyzing and segmenting a raster image if necessary. Similarly, texture properties can be associated with extractors that use the *shapeWidth* and the shape's definition to retrieve the GIS coordinates of the texture for the processed object. The GIS2KB engine, described in more detail below, uses the defined ontologies to map a Shapefile record's definition and uses it to fill the parameters required (using the extractor annotations) based on the knowledge base definition.

### **5.1.2 Ontology of Parameterized Models (RC)**

Figure 14 shows an example of a simple parameter taxonomy defining three basic classes: Generic Transport, Bridge, and Covered Bridge. Presagis Creator uses these classes and the listed properties as part of the Bridge Wizard utility to procedurally create corresponding 3D model representations. The RC ontology defines this taxonomy, mapping each element to an equivalent domain element. This defines formal meanings

for the parameters and allows a system to automatically match the graphical parameter values of the parameterized models to equivalent concepts from the semantic engine's GIS sources knowledge base. Goodwin (2005) states that Britain's National Mapping Agency, Ordnance Survey, uses Semantic Web technology primarily to translate or map concepts between different organizations or domains.

<b>Generic Transport Properties</b>	<b>Bridge Properties</b>	<b>Covered Bridge Properties</b>
Start vertex position	SubClassOf: Generic Transport	SubClassOf: Bridge
Start Angle		
Start width		
End vertex position	Span Dividers	Width dividers
End Angle	Deck Thickness	Cover Height
End width	Starting vertical angle	Wall angle
Number of segments	Ending vertical angle	Entrance angle
Left overhang size	Support width	Covered Bridge Textures
Right overhang size	Support depth	
Overhang height	Bridge Textures	
Transport Textures		

Figure 14: Generic Transport, Bridge and Covered Bridge Example Taxonomy

Figure 14 shows the **Covered Bridge** concept and the required parameters. It is a subclass of **Bridge** and **Generic Transport**, and therefore, inherits all the available properties and also adds some properties specific to **Covered Bridge**. The Geometry Definition Engine identifies an instance's class and uses the list of properties needed to extract values from the knowledge base and create a 3D model definition.

## **5.2 Main Process and Geometry Definition Engine**

The main process, based on the defined ontologies, takes the collection of GIS source data ( $S_{GIS}$ ) as input and executes GIS2KB followed by KB2Scene to extract the 3D geometry definitions making up the digital 3D world corresponding to 3D Real. All models including the environment model are created based on definitions and inserted into the landscape representation. We present the main process algorithm below:

```
Procedure MainProcess  
Input:  $S_{GIS}$ , SD, RC, ExtractorsList  
Output: 3D Digital  
    KBi = GIS2KB( $S_{GIS}$ , SD, ExtractorsList);  
    3DModelList = KB2Scene(RC, KBi);  
    3Denv = new EnvironmentModel();  
    3Denv.setDefaultSettings();  
    3Denv = TIN( $S_{GIS}$ ,  $S_E$ ).applyTexture( $S_{GIS}$ ,  $S_E$ );  
    3DModelList.add(3Denv);  
    return Visualize(3DModelList);  
EndProc
```

We will describe GIS2KB as well as the Data Extraction framework in section 5.3. We describe the Geometry Definition Engine KB2Scene as follows:

### **Procedure Geometry Definition Engine**

1. Load the realized KB containing all instances
2. Load the RC ontology defining the capabilities of the visualization system
3. For each instance in the ABox, execute the Instance Geometry Definition procedure
4. Use the resulting output to generate a detailed representation

The Geometry Definition Engine [Eid & Mudur, 2010] analyses every instance and forms a definition for each using the available extracted parameters as follows:

### **Procedure Instance Geometry Definition**

1. Retrieve the instance's most specific class type from the knowledge base

2. Select the corresponding type from the parameterized models ontology (RC ontology)
3. Retrieve the list of all parameters needed by the selected type from the ontology
4. Construct a query for every parameter to retrieve the corresponding value from KB.
5. Store type, parameters, and values as a specification for the instance 3D model.

First, to retrieve the instance's most specific class type (inferred) from the knowledge base, a generic SPARQL query for all instances is used (Figure 15). Second, the inferred class is used to select a specific concept in the RC ontology.

```
Select ?type
Where { InstanceX rdf:type ?type }
```

Figure 15: Retrieving The Most Specific Type of InstanceX using SPARQL

Third, using this concept name, all the properties listed such as those in Figure 14 are retrieved from the ontology. Fourth, a SPARQL query as shown in Figure 16 is constructed for every property to retrieve the corresponding quantitative value such as location, angles, bridge width, span and cover type from the knowledge base. %data\_property% represents a property being queried for InstanceX:

```
Select ?value
Where { InstanceX %data_property% ?value }
```

Figure 16: Retrieving a Property Value using SPARQL

Finally, an instance definition using the concept name, and properties and their values is created. Relationship properties are not used as part of the created definition; they are only used for enriching the knowledge for inference purposes.

We present the KB2Scene algorithm outputting a list of 3D models corresponding to the definitions created as follows:

```

Procedure KB2Scene
Input: RC, KB
Output: 3DModelsList
3DModelsList = {};
For every instance in KB //based on RC (extends TD)
    3DModelDef = GetVis3DSchema().getDefinition(instance.class); //note that instance.class is inferred
    //only data properties are needed (object properties only used for inference)
    For every dataproperty in RC
        If (dataproperty.domain == instance.class) then
            Value = getPropertyValue(instance, dataproperty); //SPARQL Query
            3DModelDef.setProperty(dataproperty.class, Value);
        Endif
    End
    3DModel = createModel(3DModelDef);
    3DModelsList.add(3DModel);
End
return 3DModelsList;
EndProc

```

For example, for a road instance in the knowledge base, required values such as the segments that form the road, the road width, sidewalk width, number of lanes, and the relevant texture data references are queried from the knowledge base to generate a 3D model specification (*3DModelDef*). The *createModel* function takes this specification and runs the corresponding 3D model generator procedure providing the parameters in the specification to generate the road over the terrain (in this case, altering the 3Denv model). In another instance, a bridge is inferred after knowledge base realization. Parameters of width, length, angles, no. of segments, overhang, dividers, support and texture information are extracted and a model specification for the bridge instance is used to generate the bridge 3D model procedurally. 3D models such as tunnels, overpasses and ramps are generated in a similar fashion.

### 5.3 GIS2KB and the Data Extractors Framework

GIS2KB creates a knowledge base representing  $S_{GIS}$  which is then available for querying by the geometry definition engine (KB2Scene described earlier).

```
Procedure GIS2KB
Input:  $S_{GIS}$ , SDOntology, ExtractorsList
Output: KB
KB = {};
shapeLayersList = {}; elevLayersList = {}; imageryLayersList = {};
Forevery GISourceLayer in  $S_{GIS}$ 
    If isShapeLayer(GISourceLayer) then
        mapShapefile(GISourceLayer, SDOntology, KB);
        shapeLayersList.add(GISourceLayer);
    Endif
    If isElevLayer(GISourceLayer) then elevLayersList.add(GISourceLayer);
    If isImageryLayer(GISourceLayer) then imageryLayersList.add(GISourceLayer);
End //at this point all shape data was added to the KB
//sort layers in descending order of layer resolution.
resolutionSort(elevLayersList); resolutionSort(imageryLayersList);
sortedOrderedLists SRI = {shapeLayersList, elevLayersList, imageryLayersList};
ConsistentKB(KB); // Realize KB, check consistency and generate ambiguity/inconsistency report.
                //To have most specific types for adding Shape relations (extractors other than DE9IM)
// add the relations once all records are added to KB. Done here to remove the complexity of
// having to manage uncreated instances and to take into account all shapeLayers.
addShapeRelationsToKB(KB, SDOntology, ExtractorsList, SRI);
//compute inferences based on added relations and
//calculate resulting uncertainties for inferred types
ConsistentKB(KB);

Bool Runagain = true;
While(Runagain)
    Runagain = false;
    Forevery instance in KB //based on SDOntology(extends TD)
        Forevery dataProperty in instance //data or object property assertions
            If dataProperty = null then
                xtractorfunc = ExtractorsList[extractorKey(dataProperty)]; // retrieve  $P_e$ 
                If (xtractorfunc) then
                    //run  $P_e$  to assign  $P_v$  e.g. instance.altitude = the ground altitude at location of instance from  $S_E$ 
                    xtractorfunc(KB, SRI, instance);
                    Runagain = true; //assumption: new information added
                Endif
            Endif
        End
    End
    //realize the KB for the possibility of new properties to evaluate
    If (Runagain) then ConsistentKB(KB);
Endwhile
return KB;
EndProc
```

*ExtractorsList* represents the list of concrete extractor implementations referred by each  $P_e$  and  $R_e$  in the SD ontology. *extractorKey* is a function that retrieves the extractor procedure reference in *ExtractorsList* for object property  $R$  ( $R_e$ ) or data property  $P$  ( $P_e$ ).

Layers are first categorized based on type. Shapefiles are mapped using the Source Data Ontology using the *mapShapefile* procedure described in the next section (5.3.1). After all shape information is inserted in the KB and the GIS dataset categories are sorted in descending order of resolution (highest first), the KB is realized a first time using the *ConsistentKB* procedure which extends a semantic reasoner service. If the KB is not consistent after this operation, then there is an ambiguity or an inconsistency in the input shape information. Semantic reasoner services allow us to extract the inconsistency report that explains the inconsistency in the KB. The user then corrects the problem in the input data. Shape relationships are then processed between every pair of instances in the KB. The *addShapeRelationsToKB* function is described further in section 5.3.3. The KB is realized one more time here for consistency checking and in order to infer implicit properties. Data extractors for object and data properties are discussed in section 5.3.2.

Data extractors use information about a specific instance, its properties and relationships defined in the KB to extract a specific missing property value using the set of data layers including imagery, elevation, and shapefile datasets. Data extractors are plugin-based in the system and an association requires a matching data extractor procedure (*xtractorfunc*) to be part of the system's list of extractors. Different dataset types from  $S_{GIS}$  may be used in extracting a certain value. Also different resolutions (or scale ratios) of the same dataset type are useful to confirm or find missing information.

For example, 0.5meter resolution imagery and elevation datasets have more detail per square meter when compared to 1meter resolution datasets of the same type. The input GIS datasets lists are filtered as required within the extractor procedure (e.g. on imagery, on elevation, and/or on multi-band). Extractors also provide us the option to incorporate procedural reasoning on numbers, statistics and approximations as current semantic reasoner services are not well suited for arithmetic reasoning. [Faddoul, 2011] describes a method to include algebraic reasoning as part of KBs in description logics; however, this area still has several open problems and adds more complexity to our domain of application than it adds flexibility.

After the associated *xtractorfunc* is executed, *Runagain* is used to flag that new information has been added to KB. Knowledge base realization is done using the *ConsistentKB* procedure only after processing all known missing property values at that point in time. As new information is added to the KB, new inferences are possible and assertions are specialized possibly causing other missing property values. For example, a bridge can be reclassified to a covered bridge after running the data extractor and KB realization. A covered bridge then requires a new property *cover\_definition*. The process terminates when the knowledge base is realized and there are no missing property values remaining. We describe our KB Realization solution and the reasons behind the timing of each call to this service from the GIS2KB algorithm in section 5.3.4.

### **5.3.1 Initial Mapping - Shapefiles**

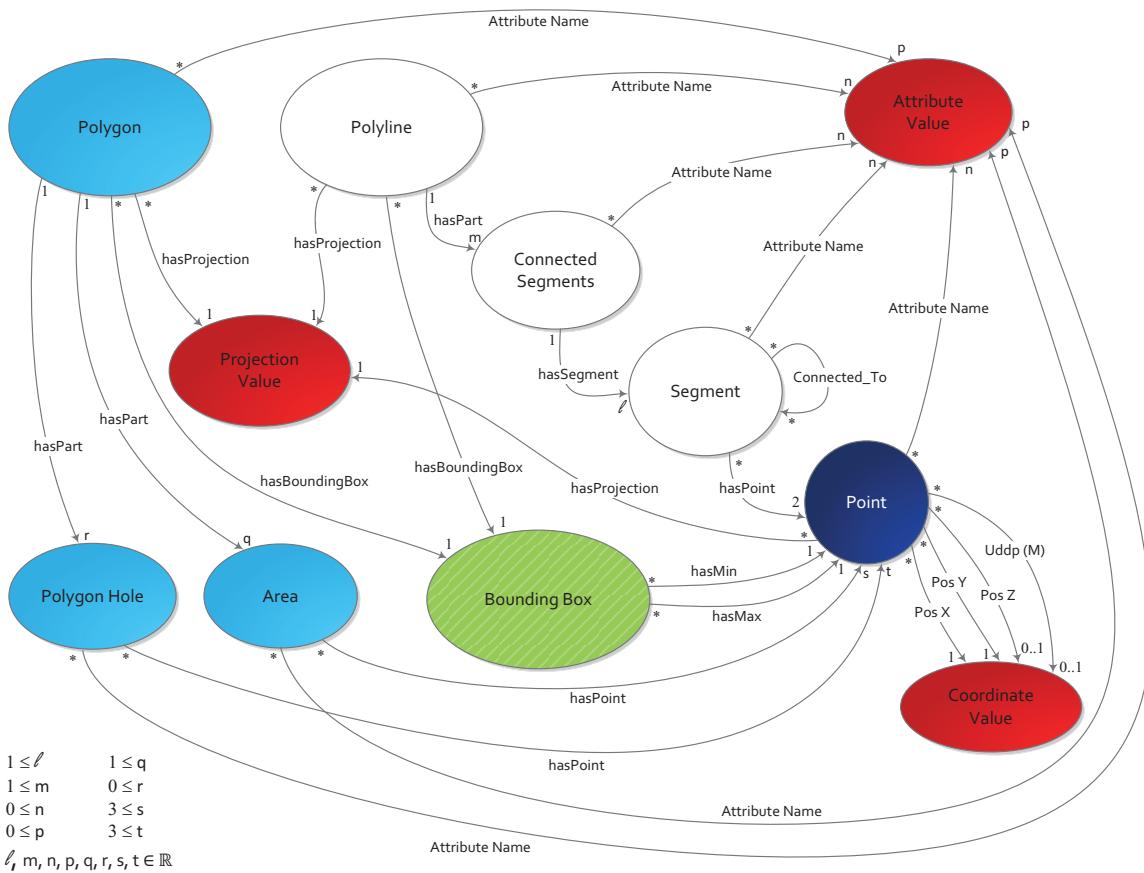


Figure 17: Shape Definition SD Ontology Mapping

Figure 17 shows the implicit knowledge graph that we extract from ESRI Shapefile data format, which we presented in Chapter 2. Each feature record is mapped based on its geometry type (*ShapeType*) to a knowledge base instance. We only consider point, polyline, and polygon types in our work. The following mapping applies unless another *ShapeType* is explicitly indicated (file name, user input, feature property, etc...). *proj\_value* represents a value from an enumeration of projections corresponding to the projection used in the Shapefile. *value* represents the value of the corresponding property in the record. *attribute\_name* and *attribute\_value* represent a property ID and its corresponding value respectively (Dbase format) in the record. *Box[]* contains ordered

coordinate pairs of two point locations representing the bounding box of the record. The Shapefile mapping algorithm maps a certain layer (GIS source layer) of type Shapefile using the SD ontology into assertions which are added to the knowledge base (KB). The *mapShapefile* procedure is defined as follows:

- A. For each **Point** record, an instance *a:Point* is created, a projection is defined based on Shapefile's projection definition, and the X, Y coordinates (and Z, M optional in case of PointZ type) are copied to the KB as values of type Double under object-value relationships. All relevant properties are copied similarly:
  - 1. *(a, value):PosX*, *(a, value):PosY*, and *(a, proj\_value):hasProjection* are added.
  - 2. Optionally, if type is PointZ, *(a, value):PosZ* is added
  - 3. If user-defined data property (uddp) Measure available, *(a, value):uddp* is added
  - 4. Map every attribute for the record using *(a, attribute\_value):attribute\_name*
- B. For each **Polyline** record, *b:PolyLine* is created with *(b, proj\_value):hasProjection*
  - 1. Map every attribute for the record using *(b, attribute\_value):attribute\_name*
  - 2. For each part in the record, a *c:Connected\_Segments* instance is created and linked with *b* using the *(b, c):hasPart* object property
    - a. Map every attribute for the record using *(c, attribute\_value):attribute\_name*
    - b. For each two consecutive points in the part, *d:Segment* is created and *(c, d):hasSegment* and , if applicable, *(d<sub>i-1</sub>, d<sub>i</sub>):Connected\_To* are added
      - i. Map every attribute using *(d, attribute\_value):attribute\_name*
      - ii. For first Point in *d* run point mapping procedure A and add *(d, a):hasRefPoint*
  - 3. For the bounding box, *e:BoundingBox* is created and *(b, e):hasBoundingBox* is added.
    - a. *min:Point* and *max:Point* are created
    - b. *(e, min):hasMinPoint* and *(e, max):hasMaxPoint* and *(min, Box[0]):PosX*, *(min, Box[1]):PosY*, *(max, Box[2]):PosX*, *(max, Box[3]):PosY* are added
- C. For each **Polygon** record assuming well-formed polygon (clean), an instance *f:Polygon* is created, *(f, proj\_value):hasProjection* is added
  - 1. Map every attribute for the record using *(f, attribute\_value):attribute\_name*
  - 2. For each part in the record, a *g:Area* (or *g:PolygonHole* is defined if part's vertices are defined in counterclockwise order) is created and linked with *f* using *(f, g):hasPart*.
    - a. Map every attribute for the record using *(g, attribute\_value):attribute\_name*
    - b. For first point in *g* run point mapping procedure A and add *(g, a):hasRefPoint*
  - 3. For the bounding box, *h:BoundingBox* is created and *(f, h):hasBoundingBox* is added.
    - a. *min:Point* and *max:Point* are created
    - b. *(h, min):hasMinPoint* and *(h, max):hasMaxPoint* and *(min, Box[0]):PosX*, *(min, Box[1]):PosY*, *(max, Box[2]):PosX*, *(max, Box[3]):PosY* are added.

As a result, the knowledge encapsulated as part of a Shapefile is added in the form of assertions to the knowledge base. The concepts used are part of the SD ontology, which maps to concepts in the TD ontology. For example, the *Connected\_Segments* and *Segment* data concepts, part of a linear Shapefile describing roads, are mapped to TD concepts (right-hand side) by default as:

$$\begin{aligned} \textit{Connected\_Segments} &\equiv \textit{Thoroughfare} \\ \textit{Segment} &\equiv \textit{Thoroughfare\_part} \end{aligned}$$

The parts in each record and the segments defining the parts are created as instances in the KB and, after inferencing, they are used for 3D model definitions. For practical purposes, if properties *ObjectClass* and *PartClass* are defined for the record, the instances created are mapped to a specified concept class rather than the default *ShapeType*. Moreover, instances in the KB are related together using object properties that do not transfer the properties easily (vs. inheritance). For these reasons, all properties are mapped for every instance created. The mapping defined in the SD ontology then discards or uses these values by mapping to properties in the TD domain ontology.

### **5.3.2 Data Extractors**

We define three categories of data extractors as described in [Eid & Mudur, 2013]. (A) Transportation Object Extractors are high-level object extractors such as the types of feature extraction procedures available as part of SocetSet. These operate on input GIS datasets in order to retrieve existence of simple objects such as roads, water bodies, or a point representing an object of interest. These can be executed to result in further input data sources or used as part of other extractors. (B) Object Property

Extractors extract object-object relationships from the collection of input data. They operate on objects to validate one or more of the listed properties. Object properties are binary relations that are defined to relate two objects e.g.  $(object1, object2):obj\_property$ . They can be classified as Functional, Inverse-Functional, Transitive, Symmetric, Asymmetric, Reflexive, and/or Irreflexive according to the  $SROIQ^{(D)}$  expressivity. (C) Data Property Extractors use GIS data sources to extract properties (object-value relationships) of a specific object to fill missing values for the 3D model definition. As example, road properties are shown in Figure 18 and the collection of bridge classes have 94 properties, not listed here for brevity (see Appendix A for the complete listing). The following sections show some example data extractors we have defined for these properties.

### **5.3.2.1 Transportation Object Extractors**

- **Point:** represents the location of an object.
- **Vector:** consecutive connected points representing a linear feature such as a road, train track, highway etc...
- **Area:** a set of points representing a polygonal area, or a point representing the location with dimensions defining the area.
- **Obstacle:** any object defined by its area of coverage.
- **Impassable Area:** an area with acute elevations, lowered area, elevated area.
- **Water Body:** an area containing water.
- **Road:** a specific type of Vector that has associated properties, a number of lanes and a roadside definition. (Figure 18 lists the properties used for generating roads.)
- **Lane:** each road has one or many lanes with lane properties.

Section	Property	Input	Definition
Initial Data			Initial Data that is required for all transportation structures: roads and bridges
	Object Class	KB Inference	The Object Classification: a Bridge type (from Types defined) or a Road Type
	Start Edge	W Extractor at linear vertex	The Edge, defined by two coordinate vertices, that defines the cross section where the structure definition starts in 2D or 3D space. 2D is overlayed on ground terrain. This ensures proper connectivity b/w connected segments.
	End Edge	W Extractor at next linear vertex	same as above
Road Properties			Properties required to graph a road transportation element. This generates a textured road based on the type and the Start and End Edges defined.
	Road Type	Object Class Property	The type of road from a classification list.
	Directionality	Road_Directionality Extractor	One way, two way road, or other road. One way roads have 1 road way instance and do not have a midsection definition. For two way and other roads, each way will have a way instance with associated properties.
	Location and Orientation	Start Edge, End Edge properties	The location and orientation of the generated road are defined by the locations of the Start and End Edges.
	Length	Start Edge, End Edge properties	The length of the road is defined by the length of the curve defined by the direction vectors of the Start and End Edges.
	Width	Start Edge, End Edge properties	The width of the road is defined by the average of the widths of the Start and End Edges.
	No. of Roadways	Separators Extractor	The number of Roadways on the road segment between the Start and the End Edges.
	No. of Midsections	Separators Extractor	There is a Midsection between each two roadways. This defines the number of Midsection instances for this road segment.
	Sidewalk (profile, width)	Roadside_Type Extractor	There is a Sidewalk at each side of the road. This defines the type of Sidewalk, its pattern or texture and its width.
Midsection Properties			Each Midsection has a Midsection definition.
	Location	Separators Extractor	Defines the coordinate on the Start Edge where the Midsection starts.
	Midsection (profile, width)	Midsection_Type Extractor	This defines the type of Midsection, its pattern or texture and its width.
Roadway properties			Every road has 1 or more Roadway instances.
	Location	Separators Extractor	Defines the coordinate on the Start Edge where the Roadway starts.
	Directionality	Way_Directionality Extractor	Direction of the roadway instance.
	No. of Lanes per way	Number_of_Lanes Extractor	This defines the number of lanes available within the roadway instance.
Lane Properties			Each Roadway has 1 or more lanes defined by the no. of lanes property.
	Lane Type	Lane_Definition Extractor	The type of lane from a classification.
	Ground Material Type	Lane_Definition Extractor	The ground type of the lane.
	Lane Width	Number_of_Lanes Extractor	The width of the lane.

Figure 18: Road Properties

- **Bridge:** a structure that is part of a Vector record; the vector crosses an obstacle at ground level or over an impassable area at the location of the bridge. All segments of the bridge are extracted. If no values are given as part of the shape record, all segments before and after the inferred bridge segment are analyzed to determine the elevation profile underneath and the imagery profiles along the segments. The start and end points of the bridge are found based on compliance with height and/or feature specifications. All segments between the start and end points of the bridge are made part of the bridge instance in the KB. An example is provided in section 7.3.

### ***5.3.2.2 Object Property Extractors***

- **Under:** belongs to an object under another object defined by the elevations of the two objects or the occultation profile in the imagery at the specific location overlapping the two objects. It is the inverse property of Over and is transitive.
- **Over:** belongs to an object over another object defined by the elevations of the two objects or the occultation profile in the imagery at the specific location overlapping the two objects. It is the inverse property of Under and is transitive.
- **Next\_To:** belongs to an object in direct proximity to another, defined by their locations and extents e.g. when a monument or park is next to a road. This property is symmetric.
- **Connected\_To:** belongs to an object that is physically connected to another. Both are part of the road network e.g. a road segment connected to another road segment or a bridge connected to a road. This property is symmetric.

- **Between:** extends Next\_To. Assumes object having this association with at least 2 other objects where object is within area between the associated objects. E.g. a valley lies between two mountains.
- **Part\_Of:** is defined for every instance as a component of another instance e.g. a road segment is part of a road and a lane is part of a road segment.
- **Through:** belongs to an object passing through another object defined by the elevations of the two objects being equal or the occultation profile in the imagery at the specific location showing intercrossing or penetration. If it is in the form of road intersection, then this property is added to both objects.
- **Has\_Material:** categorizes the texture of the surface coverage of an object given its extents using the imagery into one of (after SMC standardization): asphalt, concrete, liquid, soil, stone, sand, grass, gravel, granite, cobblestones, metal, wood, or limestone coverings. These are analyzed based on texture (color and granularity).
- **Impassable\_Through:** is defined if an object cannot pass through another object under the transportation domain interpretation e.g. for high slope variations of terrain.

### ***5.3.2.3 Data Property Extractors***

- **Position:** extracts latitude, longitude, and altitude (if available) for a given object from shape data or using imagery space coordinates based on an origin and a projection definition.
- **Altitude:** extracts the altitude of ground at a given location using an elevation dataset.
- **OL:** the Orientation-Length extractor calculates a 2D vector between two given points (Figure 19). It is defined as the vector with origin at the current point and spanning until the next point and used for every road segment part of a 3D model.

**Procedure OL**

**Input:** 2 consecutive geolocated points A and B from same linear record

**Output:** OL: 2D Orientation and length (=B-A) vector

$OL = [B(1)-A(1), B(2)-A(2), B(3)-A(3)];$

**EndProc**

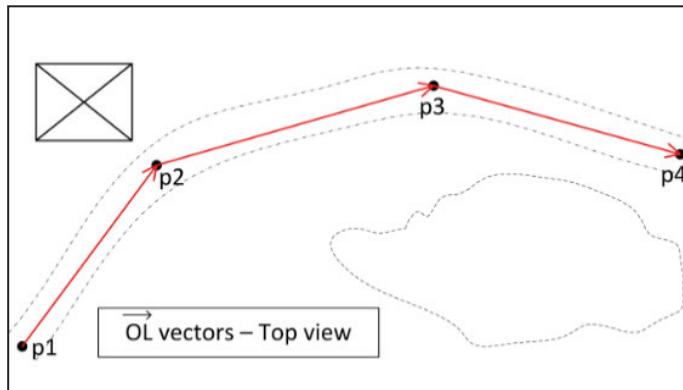


Figure 19: OL Vectors on a Road Linear Record

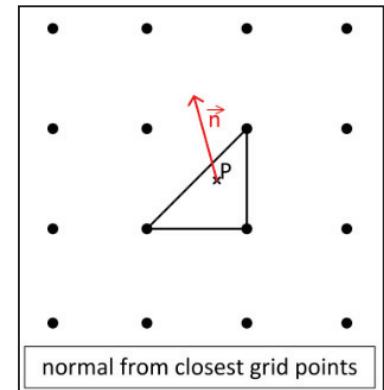


Figure 20: Normal From Three Points Defining Plane

- **n:** given a geolocated point, the normal to the terrain at that location is retrieved using the plane defined by the 3 closest elevation grid points (Figure 20).

**Procedure n**

**Input:** 3 elevation points A, B and C

**Output:** n: normal of terrain at centroid of triangle ABC

$AB = OL(A,B); AC = OL(A,C); //\text{find first and second vectors } AB \text{ and } AC$

$N = \text{cross}(AB,AC); //\text{normal to } AB \text{ and } AC \text{ using right hand rule system}$

**EndProc**

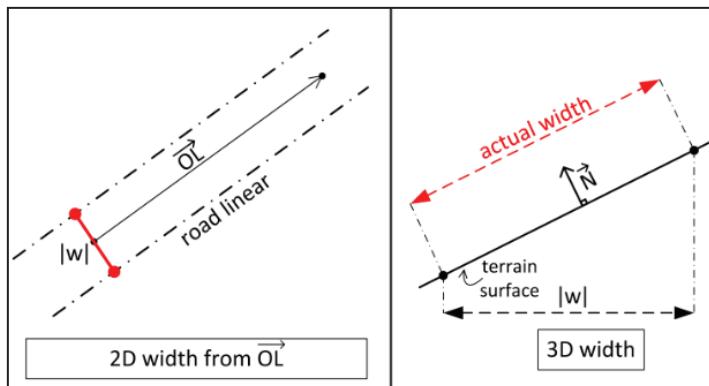


Figure 21: Width Using OL

- **W:** The user usually defines the Shapefile record points in the middle of the actual visible road in the imagery. SocetSet does so too. This extractor calculates the orthogonal vector to the orientation vector in the image plane (given by the OL extractor above) and scans the imagery (which could be segmented) along the orthogonal in both the positive orientation and the negative orientation. It returns at least two coordinates defining an edge that is normal to OL at origin of OL on the road. This edge includes all roadways and separators that are part of this road. The width of the edge does not represent the actual road width since it is not 3D projected. However, this value is sufficient for draping road textures onto the terrain geometry. To obtain the actual width, the elevation data if available is used to return the 3D coordinates for each point in the edge by querying the point's terrain altitude. (Figure 21)

```

Procedure W
Input: OL, imagery, elevations (optional)
Output: W
O = [0,0,0]; Z = [0,0,1]; //define the origin and the image plane normal
w = n(O,OL,Z); //find the orthogonal to OL in the image plane
//analyze imagery along w and find all points (roadways and separators)
W = analyze(imagery, w);
If elevations then
for k=1:sizeof(W)
    Zw = elevations.getAltitude(W(k)(1), W(k)(2)); //find terrain altitude for pt
    W(k) =[W(k)(1), W(k)(2), Zw]; //change pt to 3D
End
Endif
EndProc
```

- **Start/End Edges:** uses the OL and W extractors to determine the two points defining the starting edge orthogonal to the OL vector at the start/end point defining the object being processed. The returned edge is stored in Well-Known Text (WKT) format in the KB.

```

Procedure StartEdge
P1 = SGIS.get_ref_pt(instance);
V1 = OL(P1,P1.next()); V0 = OL(P1.prev(), P1) //OL vectors involving start point
W1 = W((V0+V1)/2, SGIS); //get the orthogonal at average between the two OL vectors
KB.addDataProp(CertaintyAnnotation(W1.certainty), start_edge_prop, instance, W1.WKT());
EndProc

```

- **N:** A few objects or features span across several terrain grid points, for example, a road width might span 3 or 4 grid points if the terrain elevations data is defined at 1m resolution. Calculating the normal from the 3 closest points only is non-realistic as this might slant the road incorrectly. In order to more accurately determine the plane defining the base of the object in such a case, several grid points are analyzed using n above and then averaged together (Figure 22).

**Procedure N**

**Input:** A: array of normals, n: number of elements in A  
**Output:** N: average normal within circle defined by (point, width)

```

V = [0,0,0]; //initialize V
for k = 1:n
    V = V + A(k,:); //add all vectors in A together
end
V = V./n; //divide each cell by the number of the elements to get the mean
EndProc

```

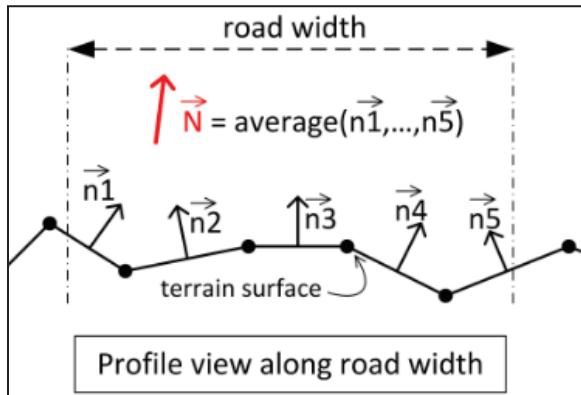


Figure 22: Average Normal

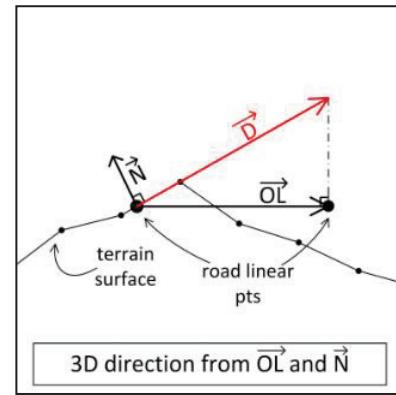


Figure 23: Direction Vector D from N and OL (Z-axis is up)

- **D:** This extractor calculates the 3D orientation and length corresponding to OL (which is only in the imagery plane) in the direction of the terrain tangent by using

the normal vector (Figure 23). D is in the plane defined by (OL, Z-axis), and its direction is defined as orthogonal to N where Z-axis is orthogonal to OL in the upward direction in the figure.

```
Procedure D
Input: OL, N
Output: D: actual 3D orientation vector corresponding to OL
// D in plane (OL, Z-axis): Dx = OLx and Dy = OLy, we need to find Dz.
// D.N = 0 => DxNx + DyNy + DzNz = 0 => Dz = (-DxNx - DyNy)/Nz
// => replace: Dz = - (OLxNx + OLyNy)/Nz
D = [OL(1), OL(2), -(OL(1)*N(1)+OL(2)*N(2))/N(3)];
EndProc
```

- **Thickness:** retrieves the thickness of a deck by estimating the value using shadows (shape from shadow techniques) or from other datasets such as LIDAR. At this time, we calculate this using a factor based on the width and material type of the object. E.g. a 5m wide bridge made of concrete will result a value of 1.5 meters.
- **Separators:** most North American road separators are identified if there are yellow-line, double line, or wide-insert (concrete blocks, elevated garden, etc...) separators between the roadways. The extractor scans the normal to OL in the length and direction of W for such artifacts. It reports the number of separators it finds and an array of origins for the identified separators and roadways. The number of roadways is generally the number of separators + 1.
- **Road\_Directionality:** identifies whether the road is one way, two way or other. It uses the number of separators identified by the Separators extractor. If the number is zero, then the road is of type one way. If it finds one then, the road is of type two way. Otherwise, the road is considered to be other.
- **Way\_Directionality:** most North American roads have, at some point, a white horizontal line defining the stop location for cars at crossings. The extractor scans the

roadway in a direction parallel to the OL vector starting from the origin defined by the Separators extractor for a white line, if this line happens just before a crossing then the direction of the roadway is towards that line, if it happens after the crossing, then the direction is the opposite way. Highway directionality can be deduced from arrow directionality separating the highway from the ramp (Figure 24).



Figure 24: Separators with Markings Showing Traffic Direction

- **Number\_Of\_Lanes**: calculates the number of lanes in a specified roadway. This extractor will also add new Lane instances corresponding to the number of lanes to the KB and relate them as Part\_Of the road segment instance. The number is retrieved by analyzing the imagery as in the W extractor by counting the lane separators encountered along the scan line and keeping a record of the width of each lane identified. This procedure is done at the location of the roadway defined by the Separators extractor along the normal to OL and is done another time again at a delta

distance in the direction of OL due to stripped lane separators often available.

Alternatively, this can be guessed based on roadway width.

- **Roadside\_Type:** uses the imagery and the road segment instance to find if a road element has a sidewalk or overhang. If it does, it retrieves the type, the width as well as the visible height per element on each side of the road.
- **Midsection\_Type:** applies for each Midsection identified by the Separators extractor. It determines what type of middle section defines the separation. It extracts the type and width.
- **Lane\_Definition:** identifies specific markings along the lane to classify it into a certain type and identifies the type of ground material for that lane as per the Has\_Material Object Property Extractor definition.
- **Cover\_Type:** identifies the type of cover on a thoroughfare. Specifically used to identify coverage and type of coverage on Bridge classes.

### **5.3.3 Adding Object-Object Relationships**

Our system evaluates all object relationships for all pairs of instances in the knowledge base based on their initial inserted type using *addShapeRelationsToKB* within the GIS2KB algorithm. Since the relationships we are interesting are defined for all instances based on the initial input type (some shown Figure 25), these relationships could be extracted after the initial mapping (*mapShapefile*) without the need to realize the knowledge base. Moreover, an instance of type Polygon, Polyline or Point will never be inferred to a different type from these values (Polygon, Polyline and Point are disjoint concepts). The set of topological relations is therefore always consistent based on the

initial basic type of the instance. We, however, realize the knowledge base using a call to *ConsistentKB* before *addShapeRelationsToKB* once in order to resolve any properties that need not be extracted (inferred) and after in order to infer other properties and check the consistency of the evaluations.

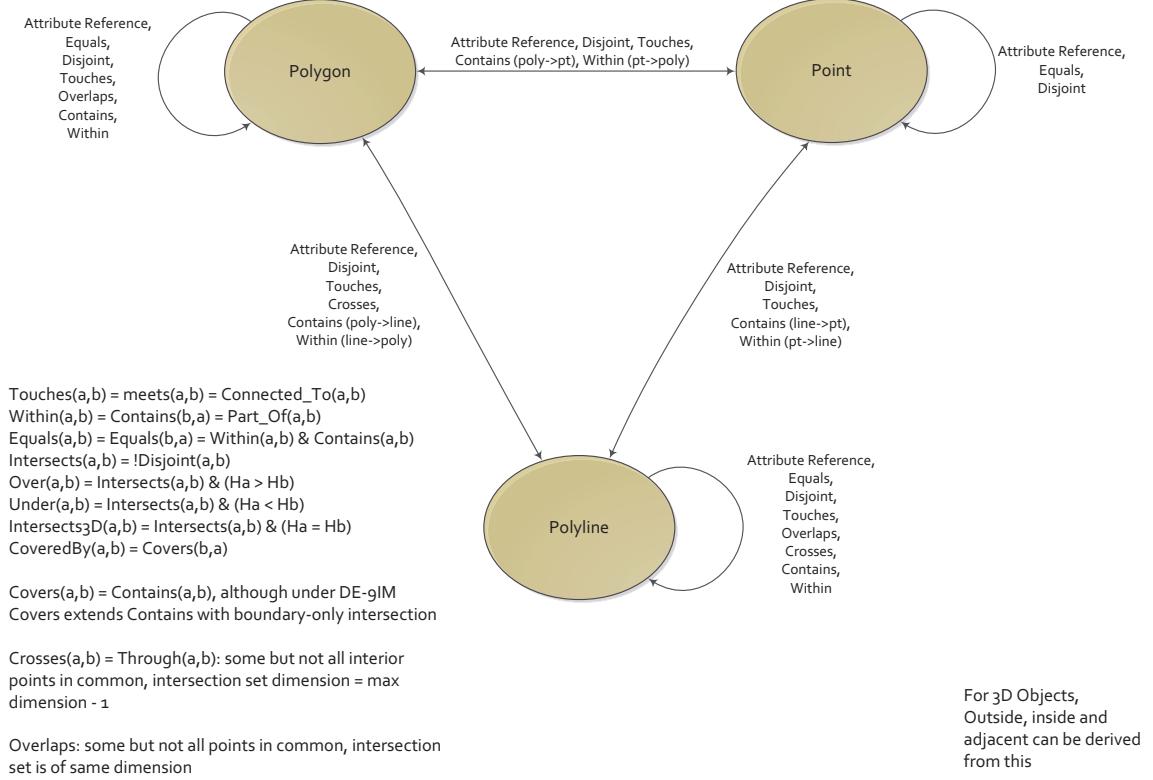


Figure 25: Relations on Geometries

*addShapeRelationsToKB()* preprocesses objects and determines some object-object relationships based on the DE-9IM model which we also extended to 2.5D properties such as Over, Under, etc... As shown in Figure 25, we resolve some of these semantically in the knowledge base if the dependent information is available. Properties such as Equals, Disjoint, Intersects, Touches (Meets), Overlaps, Crosses, Contains, Within (Inside), Covers, CoveredBy are computed using the DE-9IM model and matrix.

Others are inferred, verified for consistency, or processed using Object Property extractors defined in the framework. As described by the Data Extractors framework, an Object Property extractor  $R_e$  is associated with every object-object relationship it represents through an annotation on the relationship in the SD ontology. The *addShapeRelationsToKB* algorithm is presented as follows:

```

Procedure addShapeRelationsToKB
Input: KB, SDOntology, ExtractorsList, SRI
Output: KB(modified)
instanceList = KB.GetInstances(); //can also filter by instances type; we consider all TD instances.
For every instance I1 in instanceList
    element1 = SRI.shapeLayersList.getShape(I1.ShapeRef);
    List otherInstances = instanceList - I1;
    For every instance I2 in otherInstances
        element2 = SRI.shapeLayersList.getShape(I2.ShapeRef);

        //compare both geometries for <a relation b>
        matrix = DE9IM(element1.geometry, element2.geometry);

        //Get all possible relations for object type (based on initial class). No need for realization. SD extends TD.
        // based on domain and range of properties
        I1relations = KB.getRelationsList(I1.class, I2.class, SDOntology);

        For every objectProperty in I1relations
            r_evaluator = ExtractorsList[extractorKey(objectProperty)]; // retrieve  $R_e$ 
            //run  $R_e$  if it exists and if this property is not already defined between I1 and I2 by some other means
            If (r_evaluator and KB.notDefined(objectProperty, I1, I2)) then
                If (objectProperty.type = de9im) then
                    r_assertion = r_evaluator(matrix); //e.g. disjoint(I1, I2) = FF*FF**** matrix
                else
                    //for other than DE9IM relations e.g. impassability relation between a line and an area
                    //could add new assertions about instances in the knowledge base.
                    //E.g. equivalent relations or an annotation with a certainty value.
                    r_assertion = r_evaluator(I1, I2, SRI, KB);
                Endif
                If (r_assertion) then //add object property assertion when true
                    KB.addObjectPropertyAssertion(objectProperty, I1, I2);
                Else //add negative object property assertion when false
                    KB.addNegativeObjectPropertyAssertion(objectProperty, I1, I2);
                Endif
            Endif
        End
    End
    //instance was evaluated with all other instances and can be omitted for further processing
    // by adding properly defined relations between (I1, I2) pairs in knowledge base, a relation between (I1,I2)
    // will result in an inferred relation between (I2, I1)
    instanceList.remove(I1);
End
EndProc

```

The return value of the evaluation on the relationship, *r\_evaluator*, determines whether the assertion is true or false. The relationships are added either as an object property assertion or as a negative object property assertion accordingly. This is done for the purposes of completeness. Since order of instances in relationship evaluation and DE-9IM is not important (relationships are inverse or symmetric), the evaluation of every instance pair against their possible relationships is done only once and after processing each instance against all others, the instance is removed from the list. The same pair is not processed again. This reduces the complexity by having the first instance process all others and the last instance only processed by the one preceding it. At the end of this procedure, all topological relationship properties between every pair of instances would be added to the KB.

#### **5.3.4 Knowledge Base Realization**

We tried several approaches for incorporating knowledge base realization (*ConsistentKB*) into our process. Initially, a solution was attempted where the knowledge base was realized after every added element. This was done (1) in order to identify any clashes in the knowledge base right after an insertion and (2) in order to keep the most accurate specialization at all times for all instances in the knowledge base thus limiting the number of extractors required to execute.

Realizing the knowledge base frequently is compute-intensive since its procedure is complex and even hyper-tableaux based algorithms can take up to double exponential time [Faddoul, 2011]. We therefore run the realization step once after all information from feature data is mapped to the knowledge base and a second time after evaluating all

object-object relationships. These are performed to check for consistency, and generate ambiguity/inconsistency report, at those levels and to infer the most specific types for a complete analysis of possible relationships such as extractors other than spatial relations related to a specific subclass. Moreover, if some relationships are semantically defined, these would avoid running unnecessary extractors and specialize instances to the most specific subclass for object and data property relationship extraction.

Due to the Open World Assumption (OWA) that semantic web provides along with consistency and satisfiability principles, elements can only be specialized into more accurate definitions in the final realized hierarchy, otherwise a contradiction (or clash) would result in the knowledge available. We optimize by running *ConsistentKB* only if a missing property value was assigned a value using its property extractor. Any assertion might result in some specialization of an instance type which in turn might have missing property values. Doing this process repeatedly (several iterations might be required) while realizing the KB ensures that the most specific specialization w.r.t. available input is achieved and all missing property values have been extracted and asserted.

## **5.4 OWL Link Adaptation**

We use the OWL Link specification [Liebig et al., 2008] as the common API to communicate between the different sub-processes through the semantic reasoner infrastructure and framework. OWL Link allows a standardized communication mechanism and API to the knowledge base for querying, TBox and ABox manipulation.

Semantic Web mostly uses Java because of its flexibility, portability and to comply with practices in the World Wide Web. OWL Link was implemented using existing frameworks like the OWL API, a Java API based on the W3C OWL2 Specification.

Our system needs the flexibility to be able to add knowledge and extract knowledge from different systems written in different programming languages. It was a challenge to find a feasible solution to allow this. The current systems we use are MapWindow GIS, implemented in C#, the Semantic Web Framework, with most interfaces available only in Java, and our 3D scene generator, implemented in C++. In order to be able to communicate and share knowledge between the different systems for the purposes of prototyping and validation, we have created an OWL Link .Net compatible framework based on the original OWL Link Java framework that can be used in both C# and C++ programming languages.

It seems that this implementation is a valuable contribution to the OWL Link user community specifically and the Semantic Web community in general. While one of the main authors of OWL Link, Olaf Noppens, showed great interest in making this work available through the OWL Link website, several Semantic Web developers showed interest in this work because they were looking at implementing applications and agents in different computer languages and, this framework would make it easier to interact with other Semantic Web framework components. This work is publicly available as a downloadable package on the internet. Appendix B contains our notes on OWL Link porting to the .Net framework.

We are confident that this work will be used by others in the Semantic Web field.

## **5.5 Simple Deck Bridge 3D Model – An Example**

Our process first uses the available basic spatial elements (parts, segments and points of the Shapefile linear record) as well as the property values and adds them as assertions to the knowledge base as defined in the Initial Mapping algorithm. New segments are therefore identified and inserted in the knowledge base with properties such as  $(segments1, segments2):Connected\_To$  where  $segments1$  and  $segments2$  are instances of class *Thoroughfare* in the KB. After analysis for high slope values of the elevation patterns under the segments, further shapes are identified and facts are added such as a *high\_slope\_area* instance of class *ImpassableArea*.

Possible relationships are then analyzed using their associated extractors between all instances and added as assertions as well. In our case,  $(segments2, high\_slope\_area):over$  was analyzed using the DE-9IM model and was added as a positive object property. The other properties, which are false, are added as negative properties. The KB is then realized and the type of  $segments2$  is inferred to *Bridge*. This reclassification of  $segments2$  is possible since the KB contains an equivalence axiom for *Bridge* in the TD. Using reasoning, inference services available through the reasoner allows for the ABox realization where all instances are processed and classified according to their most specific subclasses (specialization). If several segments happen to represent a single bridge entity, then each segment results in a bridge definition (span) which altogether forms the complete bridge entity. At least one segment is required to represent a bridge; if a segment spans over a bridge (only a part of it is a bridge), the segment is not dissected, which might result in a wrong representation.

We have 21 different types of bridges with 94 possible properties. To generate a simple deck only definition for example, referring to the property listing in Appendix A, initial data, spans, angles, and a deck definition are necessary while LOD and textures are optional. For initial data, shown in the first section of Figure 18, the object class, start edge, and end edge of each bridge definition are required.

If the Shapefile record contains all the required property values explicitly (available in the KB after mapping and realization), then the definition could be complete (no missing property values). However, since *segments2* was inferred to a new class type, it most probably doesn't have values for all the new class' properties. The extractor associated with every missing property value is executed as per the Data Extractors framework and GIS2KB.

Some extractors are defined by monotonic formulae that arithmetically derive the required information. If each bridge span segment is inferred as a *Bridge* in the knowledge base, then each bridge span is generated independently. Otherwise, a bridge definition representing the connected segments would be available. The maximum ground height under the OL vector representing the bridge is retrieved using the Altitude extractor. Vertical and horizontal start and end angles for the bridge are retrieved using the D and OL extractors. They determine the start and end angles at which the bridge arches, bows, or curves. By default, the algorithm generates values for the bridge to arch smoothly between the two edges. The deck definition includes a number of subdivisions, thickness and overhang definitions. The number of deck subdivisions is based on the curviness defined by the start and end edges over the OL vector. A higher number is

generated for high curviness with a maximum of 10 subdivisions for a curviness of 45 degrees.

Other extractors can use the raster layers (elevation data, imagery texture) as well as the available knowledge and the instance data to derive a value through pattern recognition, analysis or computer vision techniques. Since the imagery is properly scaled corresponding to the projection used, these extractors could segment and analyze an imagery subset as per application requirements along the linear segment (the road texture) for patterns or texture extraction. They can extract property values and add them as further assertions to the KB. Following on our example, the road width is retrieved using the W extractor, which also ensures proper connectivity between entities. The deck thickness uses the Thickness extractor. The left and right overhang widths and heights are retrieved using the Roadside\_Type extractor. Overhangs are always within the structure of the deck width (defined by the start/end edges). Moreover, other property extractors for class Bridge include Cover\_Type while all Road class property extractors shown in Figure 18, are required to extract values such as *no\_of\_lanes*, *lane\_type*, etc... These could also infer a different object class depending on their values. Pointers to data objects can also be added as data properties referring to pieces of data that will be used by the 3D rendering procedure. For example, the extracted bridge texture can be stored in a specific format and referred to, through a Uniform Resource Identifier (URI). The visualization system would use the URI to locate and load the texture for the bridge.

The procedural algorithm, with the above values for properties, will generate a deck only bridge (Figure 26). The linear feature points are shown between the OL vectors. The normals and the direction vectors, respective to the OL vectors and scaled

down for clarity purposes, are also shown; the values produced ensure a smooth connection with the road segments connecting with the bridge. OL2 vector, in this figure, represents the inferred bridge (*segments2*). The ground at D1 is in the upward direction at almost 10 degrees and 5m higher than the end of the bridge. The bridge, given these values, will be modeled to follow terrain smoothly. We then position this bridge at the location between the edges returned by the normal to the OL extractor and the W extractor and projected on the terrain surface. Properties for more complex bridges can be extracted similarly.

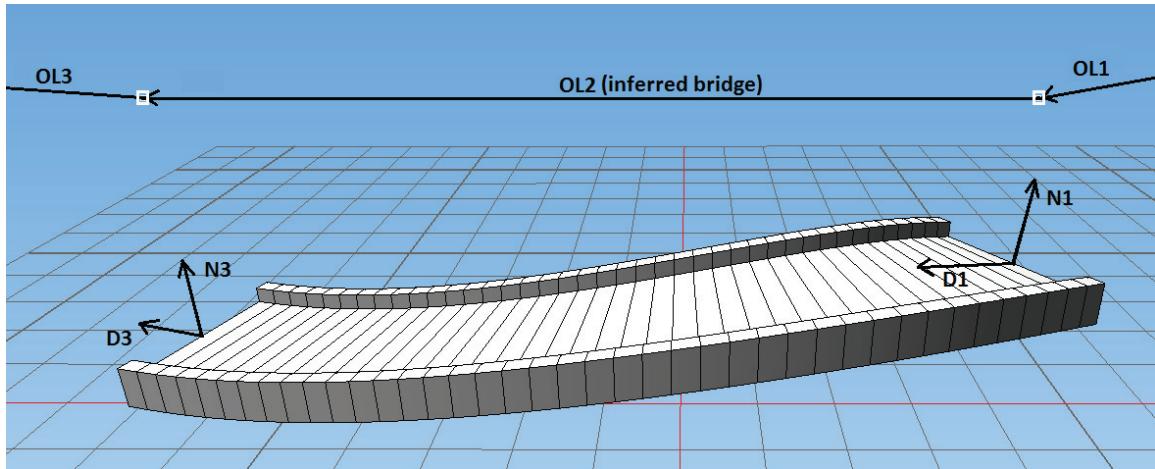


Figure 26: Deck Only Example

# ***Chapter 6***

## ***Handling Data Uncertainty***

[Poole et al., 2009] advocate for the separation between assertional axioms and uncertainty as a good design principle where probabilities can always be dealt with separately in their proper context. This separation allows the definition of assertional knowledge axioms that can be reused without having the theories about the probabilities within the same context. This is helpful especially since the theories about the probabilities might change independently from the domain knowledge, the meanings and the facts. We can use probabilities reasoning in side-processes and interface with regular DL for deductive reasoning. [Laskey, 2009] and [Li et al., 2006] show a good summary of the state of the art in Probabilistic reasoning and introduce a Probabilistic Description Logic Program that allows querying by analyzing all possible answers and returns the highest score. Our system can certainly benefit from reasoning with probability and can be extended to model it.

In our domain of application, uncertainty could happen in (1) the input GIS sources, (2) the definition (meaning) of a term or concept in an ontology, or (3)

associated with extracted data. We only address (3) as we consider that (1) the input GIS sources always contain a full representation (implicit or explicit) for the 3D Real world (after Assumption 1) and (2) we implement separation between uncertainties and knowledge based on [Poole et al., 2009]. Moreover, all our probability related relationships are in the form of relationship extractors which can handle uncertainties within the extractor procedure. We therefore address the problem of representing uncertainties of data assertions in the knowledge base and calculating the resulting uncertainties of entailed assertions.

Extractors are procedures that users associate with a concept that represents the procedure's return value(s) (defined per concept class). For example, a *roadwidth* extractor may make use of a procedure that classifies certain pixels with some certainty as part of the road and others of a different category. Similarly, many data extractors can be associated with some uncertainty or thresholds such as *PrecedenceByImagery*, *Next\_to*, *Impassable\_Through*, *Between*, *Thickness*, *Separators*, *Road\_Directionality*, *Number\_of\_Lanes*, as well as many others. Extractors such as *Position*, *Altitude*, *OL*, *n*, *N*, and *D* (see Data Property Extractors in section 5.3.2.3) as well as spatial relationship extractors are defined by formulae (no uncertainty) that arithmetically derive the required information. We define these as Type A extractors while those that have an associated uncertainty are defined as Type B extractors.

Existing work does not handle our problem adequately and default knowledge base services do not handle uncertainties under the expressivity chosen ( $SROIQ^{(D)}$ ). As an example, if the *roadwidth* extractor for a certain instance  $I$  returns true or false for value 3.2(meters) then in a conventional knowledge base under  $SROIQ^{(D)}$  expressivity,

we can only assert  $(I, "3.2") : roadwidth$  (positive assertion) or  $\neg(I, "3.2") : roadwidth$  (negative assertion). This data property relationship would therefore either exist or we know it is not going to exist (different from the relationship is unknown) and does not mean that the inverse of the property exists. A certainty value could be stored as part of the knowledge base in the form of an annotation on the relationship (serving as a storage-only mechanism). In this example, the certainty of the width value can be stored as an annotation on the *roadwidth* object-value relationship as  $((I, "3.2") : roadwidth, 0.7) : A_{certainty}$  denoting 70% certainty for this assertion. This annotation has no semantic value within the chosen expressivity but it can be used by computer programs or, using some other expressivity constraints, by probabilistic description logic reasoners such as Pronto (pellet-based).

Probabilistic description logic calculates uncertainties on a per concept basis, but we believe that our needs are much simpler. (1) There is a difference between the probability of existence of a concept or instance and the probability of existence of an assertion about an instance. In our case, we only need to model the latter since instances are created as per the shape definitions and therefore there is no doubt about their existence. (2) Uncertainty in concept definition need not be modeled as part of the ontology as discussed earlier. (3) In our case, the certainty of an assertion does not depend on the certainty of the elements used by the assertion or vice versa. For example, consider an axiom A defined as  $(i1, w1) : coverType$ . The axiom's certainty is considered independent from the certainties of the facts about *i1* or *w1*. The *coverType* extractor evaluates the absolute certainty of A with respect to available facts and inserts it into the KB.

## 6.1 Zadeh Semantics with KB Justifications

In Figure 4, we have shown that most effort is spent in defining shapefile data that is representative of what is required for detailed 3D representation. Since we have replaced the effort consuming shapefile definition step in Figure 4 with our Data Extractors framework shown in Figure 13, it is required that we address uncertainties on the extracted results and axioms produced. Only Type B extractors need to associate a certainty value in the knowledge base as an annotation. Since our process includes the entailment of new information, we also calculate the resulting uncertainty value of an entailment based on the justifications of each entailment. For this, explanations (a set of axioms) are necessary to calculate the certainty value of an assertion (axiom).

We modify our *ConsistentKB* method to extend the default knowledge base realization service and then compute all resulting certainty values for each assertion entailment discovered by the system. All certainty annotations are instance-specific and are therefore saved in the ABox per assertion.

We use fuzzy variables under the Zadeh semantics to represent our uncertainties. We chose Zadeh semantics as its definition of conjunction (t-Norm) and disjunction (t-Conorm) between two axioms is sufficient and necessary to represent instance class entailments and their explanation axioms. In a previous attempt, we have tried using equivalence axioms in a class definition to calculate the resulting fuzzy variable value for an instance. An equivalence axiom defines a set of concepts and object or data properties that together define the class. However, that only worked if exactly one equivalence axiom is defined. Explanation services available with semantic engines allow the support

of multiple equivalence axioms and provide the minimum set of assertions and axioms resulting in an entailment. Moreover, they can return multiple explanations for a certain entailment. At the moment, we are only interested in concept class entailments for instances. We therefore ignore all other entailment types because we assume there is always an explicit minimum set of axioms (or some implicit made explicit by a previous pass) that results in this concept class entailment.

We recall that, according to [Horridge et al., 2008], a certain explanation for a class entailment of an instance is a minimum list of assertion axioms that directly results in the entailment. We are particularly interested in all assertion axioms of the explanation that explicitly contain the specific instance being processed. We call this set “the variable set”. This set is sufficient if each extractor uses the certainties of the concerned facts when evaluating a property assertion and assigning its corresponding certainty. The least certain axiom in every variable set represents a certainty value for the entailed class type. If two or more explanations exist for a certain entailment, then the entailment’s certainty value is the most certain value resulting from each explanation. This algorithm is presented as part of *ConsistentKB* in the following section which is called every time we realize the knowledge base in our process.

We have seen that, in *SROIQ* semantics described previously, ABox *assertions* for two instances a and b are defined as:

- $a: C$  (where  $C$  is a Concept)
- $(a, b): R$  (where  $R$  is an object-object relationship)

We extend these by associating a fuzzy variable  $v$  that represents a certainty value to each *assertion* as an annotation  $(assertion, v): A_{certainty}$ . OWL2 allows such annotations and therefore these values can be stored within the KB without change in expressivity. If  $v$  is not specified, then the assertion is considered to be attributed 100% certainty, this includes all initial data from Shapefiles.

Consider an explanation  $E$  of a concept class entailment assertion  $i:C$  for an instance  $i$  where  $E \models (i:C)$ . For each assertion axiom  $x \in E$ , there exists  $(x, v): A_{certainty}$  defining a certainty value  $v$ . Since  $i:C$  is entailed using the conjunction of all axioms in  $E$ , the resulting certainty,  $u(E)$ , of  $i:C$  is defined as, under Zadeh semantics, the least certain axiom in  $E$ . This handles the case of one equivalence axiom in a class definition (see Example 2 under section 6.3.2). The variable set is determined by whether an instance  $i$  is an element of assertion  $x$ .  $\inf$  is a function that takes the set of values  $v$  and returns the minimum value in the set. Function  $u$  is defined as follows:

$$\forall x \in E, ((i \in x) \cap (x, v): A_{certainty}) \rightarrow (v \subseteq V)$$

$$u(E) = \inf(V)$$

If more than one explanation exists in a set  $J$ , since  $i:C$  is entailed by the disjunction of all explanations in  $J$ , the resulting certainty,  $v(J)$ , is defined as, under Zadeh semantics, the most certain result  $u(E)$  obtained from each explanation  $E$  in  $J$ . This handles multiple equivalence axioms or disjunctions in the equivalence axioms (see Example 3 under section 6.3.3).  $\sup$  is a function that takes the set of values  $u(E)$  associated with each explanation  $E$  in  $J$  and returns the maximum value in the set. Function  $v$  is defined as follows:

$$\begin{aligned}\forall E \in J, u(E) &\subseteq K \\ v(J) &= \text{sup}(K)\end{aligned}$$

## 6.2 Algorithm *KBConsistent*

The *ConsistentKB* procedure extends the semantic engine knowledge base realization service, which computes the subsumption hierarchy on the knowledge base and generates the consistency report, if required. The procedure then computes concept class uncertainties based on KB justification services, and commits the changes on the knowledge base.

```

Procedure ConsistentKB
Input: KB
Output: KB(modified)
//realize the ABox
KB.realize();
If (!KB.isConsistent()) then
    GenerateInconsistencyReport();
    return ERROR();
Endif

//compute uncertainty of entailed concepts of instances
Forevery Instance in KB
    Explanations = KB.explain(instance.type);
    If (Explanations) then
        Variablesets[Explanations.size];
        Result_e = 0;
        Forevery explanation in Explanations
            Result_a = 1;
            Forevery assertion in explanation
                If (assertion.contains(instance.id)) then
                    Variablesets[explanation.id].add(assertion);
                    V = assertion.annotations.fuzzyValue;
                    Result_a = min(Result_a, V);
                Endif
            End
            Result_e = max(Result_e, Result_a);
        End
        (instance.type).annotations.fuzzyValue = Result_e;
    Endif
End
EndProc

```

## 6.3 Examples

An assertional axiom  $A$  annotated with a fuzzy certainty value  $x$  will be denoted as  $(A)(x)$  for the purposes of brevity.

### 6.3.1 Example 1

Consider the following TBox and ABox:

$$A \equiv \exists prop. B$$

- (1)  $(i1; B)(x)$
- (2)  $((i2, i1); prop)(y)$

Our system will first insert  $i1$  and  $i2$  with respective basic types as part of *mapShapefile* (complete certainty is assumed). If (1) is explicit and is asserted to the KB before (2), then, as part of *addShapeRelationsToKB*,  $prop$  is evaluated using the associated extractor function. Since the type of  $i1$  is already known ((1) is asserted), the extractor of  $prop$  can use the uncertainty variable  $x$  in its evaluation and assign a corresponding value to  $y$  as an annotation after asserting (2). After all relations are evaluated, the knowledge base is realized and  $KB \models (i2; A)$ . The explanation is given as:  
 $\{((i2, i1); prop)(y), (i1; B)(x), (A \equiv \exists prop. B)\}$ . The variable set for this explanation is  $\{((i2, i1); prop)(y)\}$  and  $(i2; A)(u)$  has certainty value  $u = y$ .

On the other hand, if (2) is added before (1), a limitation exists where the extractor of (2) does not know about the type of  $i1$  and there is no relationship between  $y$

and  $x$ . This is avoided by the fact that we execute initial mapping and the KB is realized before relationship extraction.

### 6.3.2 Example 2

Consider the following TBox and ABox:

$$A \equiv \exists prop. B$$

$$C \equiv \exists prop. A \sqcap \exists prop. B$$

- (1)  $(i1:B)(x)$
- (2)  $((i2,i1):prop)(y)$
- (3)  $((i3,i2):prop)(z)$
- (4)  $((i3,i1):prop)(w)$

After realizing the knowledge base,  $KB \vDash \{(i2:A), (i3:C)\}$ , which are available in explicit form with explanations:

- $\{((i2,i1):prop)(y), (i1:B)(x), (A \equiv \exists prop. B)\} \vDash (i2:A)(u)$
- $\{((i3,i2):prop)(z), ((i3,i1):prop)(w), ((i2,i1):prop)(y), (A \equiv \exists prop. B), (i1:B)(x), (C \equiv \exists prop. A \sqcap \exists prop. B)\} \vDash (i3:C)(v)$

In this case, entailment  $(i3:C)$  is dependent on that of  $(i2:A)$ . In the current algorithm, since all object property relationships are evaluated and asserted after basic class insertion and before the knowledge base is realized, the fuzzy variables are independent unless an explicit assertion is available before the value is used. In this

example, (1) is an explicit assertion with variable  $x$ . If (1) is added before (2), then the extractor of (2) can use  $x$  and assign a value for  $y$  as in Example 1. When the extractor of (3) is executed for  $(i3, i2):prop$ , the type of  $i2$  is not explicit and therefore no relationship is known between the associated variables. When the extractor of (4) is executed for  $(i3, i1):prop$ , if (1) was inserted previously,  $x$  can be used in the analysis and calculation of  $w$ .

After adding all object properties and realizing the knowledge base, the values for  $u$  and  $v$  would be properly assigned with  $u = y$  (where  $y$  depends on  $x$  since  $(i1:B)$  is known) with variable set:  $\{( (i2, i1):prop \ i1)(y) \}$  and  $v = w \cap z$  with variable set:  $\{ ( (i3, i2):prop )(z), ( (i3, i1):prop )(w) \}$  (where  $z$  is independent of  $u$  since  $z$  was calculated by the extractor before  $(i2:A)(u)$  was known and  $w$  dependent on  $x$  since the type of  $i1$  was known when calculating  $w$ ).

If  $(i3, i2):prop$  extractor is to use  $u$  and assign a value for  $z$ , then  $(i2:A)$  entailment is required before  $(i3, i2):prop$  extractor is executed. This would be possible only if *ConsistentKB* is called after every object property assertion is inserted. This is already done for data property assertions.

### 6.3.3 Example 3

Consider the following TBox and ABox:

$$Tunnel \equiv \exists Through. (NaturalObject \sqcup Structure)$$

$$\sqcup \exists Under. (NaturalObject \sqcup Structure \sqcup Thoroughfare)$$

- (1)  $(i2:\text{NaturalObject})(u)$
- (2)  $(i3:\text{Structure})(v)$
- (3)  $(i4:\text{Throughfare})(w)$
- (4)  $((i1,i2):\text{Under})(x)$
- (5)  $((i1,i3):\text{Through})(y)$
- (6)  $((i1,i4):\text{Under})(z)$

If (1), (2) and (3) are added to the ABox prior to the object property relationships, then the extractors of *Under* and *Through* can use the corresponding variables to analyze and calculate the resulting variable (e.g.  $((i1,i2):\text{Under})(x)$  can use  $u$  and calculate  $x$ ). Otherwise, all variables are independent.

After knowledge base realization however,  $KB \models \{(i1:\text{Tunnel})(p)\}$  and the explanations are given as:

- $\{((i1,i4):\text{Under})(z), (i4:\text{Throughfare})(w), (\text{Tunnel} \equiv \exists \text{Through}. (\text{NaturalObject} \sqcup \text{Structure}) \sqcup \exists \text{Under}. (\text{NaturalObject} \sqcup \text{Structure} \sqcup \text{Throughfare}))\}$
- $\{((i1,i2):\text{Under})(x), (i2:\text{NaturalObject})(u), (\text{Tunnel} \equiv \exists \text{Through}. (\text{NaturalObject} \sqcup \text{Structure}) \sqcup \exists \text{Under}. (\text{NaturalObject} \sqcup \text{Structure} \sqcup \text{Throughfare}))\}$
- $\{((i1,i3):\text{Through})(y), (i3:\text{Structure})(v), (\text{Tunnel} \equiv \exists \text{Through}. (\text{NaturalObject} \sqcup \text{Structure}) \sqcup \exists \text{Under}. (\text{NaturalObject} \sqcup \text{Structure} \sqcup \text{Throughfare}))\}$

The variable sets associated with each explanation respectively are:  $\{((i1, i4): Under)(z)\}$ ,  $\{((i1, i2): Under)(x)\}$ , and  $\{((i1, i3): Through)(y)\}$ . As mentioned above, we define the final variable value as the disjunction of all results from every variable set. Therefore,  $p = \max(z, x, y)$ . If we assume only one assertion is 100% certain then  $(i1: Tunnel)$  would be 100% certain.

## 6.4 Observations

We have introduced a novel process for calculating uncertainty. The major difference being that our method is independent of the expressivity used or chosen for the knowledge base as long as reasoner explanation services, as defined by [Horridge et al., 2008], are available for this expressivity. We, therefore, do not have to extend the semantics used for the purposes of uncertainty (adding uncertainty to concepts, calculating and generating resulting values); it is contained within the process and transparent to the user. This promotes the separation of knowledge and probabilities as described in [Poole et al., 2009].

Our method reduces the complexity of calculating the probability/possibility within the reasoning process due to the reduced set of axioms in the variable set. The variable set contains only axioms that are directly related to the instance being processed. This condition is sufficient to calculate the entailed axiom's certainty value in our process. By adding the relationships using the Data Extractors framework, we also extract relationship properties (such as inverse relations, functional, transitivity, etc...) and

related inferences. For example, if there exists two axioms in the ABox,  $(i1, i2): r$  and  $(i2, i3): r$  and  $r$  is transitive. Using the Data Extractors framework, there must be an added axiom  $(i1, i3): r$  as well (all instances are evaluated against all relationships). In order to maintain the calculation of certainties for entailments simple, any certainty attribution for an axiom other than concept class entailment has to be inserted by an extractor. Entailed relationships other than concept class entailments will not be addressed if an uncertainty attribution is not present (its certainty will be 1 in this case).

Lukasiewicz in [Lukasiewicz, 1998] specified that restricted deduction problems that are P-complete for classical logic programs are already NP-hard for probabilistic logic programs. [Lukasiewics & Straccia, 2008] specified that his probabilistic reasoning tableaux extension  $P - SHOIN^{(D)}$  (after  $SHOIN^{(D)}$  expressivity with class  $NEXPTIME$ ) has best case  $FP^{NEXPTIME}$  complexity class (as described in section 3.5).  $SROIQ$  is already  $2 - NEXPTIME$  class. Since, we associate uncertainties with respect to assertion data only and calculate the resulting uncertainties of entailed instance class assertions, we believe that our method of calculating the resulting uncertainties significantly reduces the algorithmic complexity compared to using probabilistic description logic since we only evaluate uncertainties for entailed class assertions using their explanations.

# ***Chapter 7***

## ***Results and Examples***

This chapter shows some examples that demonstrate our process. Section 7.1 shows a simple example using our framework without the use of uncertainty. Section 7.2 demonstrates our handling of uncertainty process and the Knowledge Base ABox Realization extension. Sections 7.3 through 7.5 show concrete examples from a real world ROI. Section 7.6 shows our reproduction of a complex bridge structure. Finally, section 7.7 shows a different application of using knowledge and uncertainties of extracted features of an ROI.

### ***7.1 Generating Thoroughfares and Their Details***

Shp0:

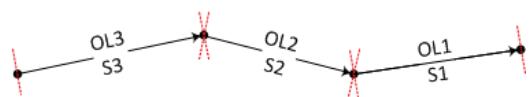


Figure 27: Shp0 Polyline Record



Figure 28: Generated *Connected\_Segments*

**Objective:** Specialize polyline in shapefile and property filling using the Data Extractors framework.

Figure 27 shows a polyline record (Shp0) with overlaid OL segments. This record has 1 part with 4 points defined. A segment is defined between each two consecutive points in a part with a total of three segments in this case (S1, S2, and S3).

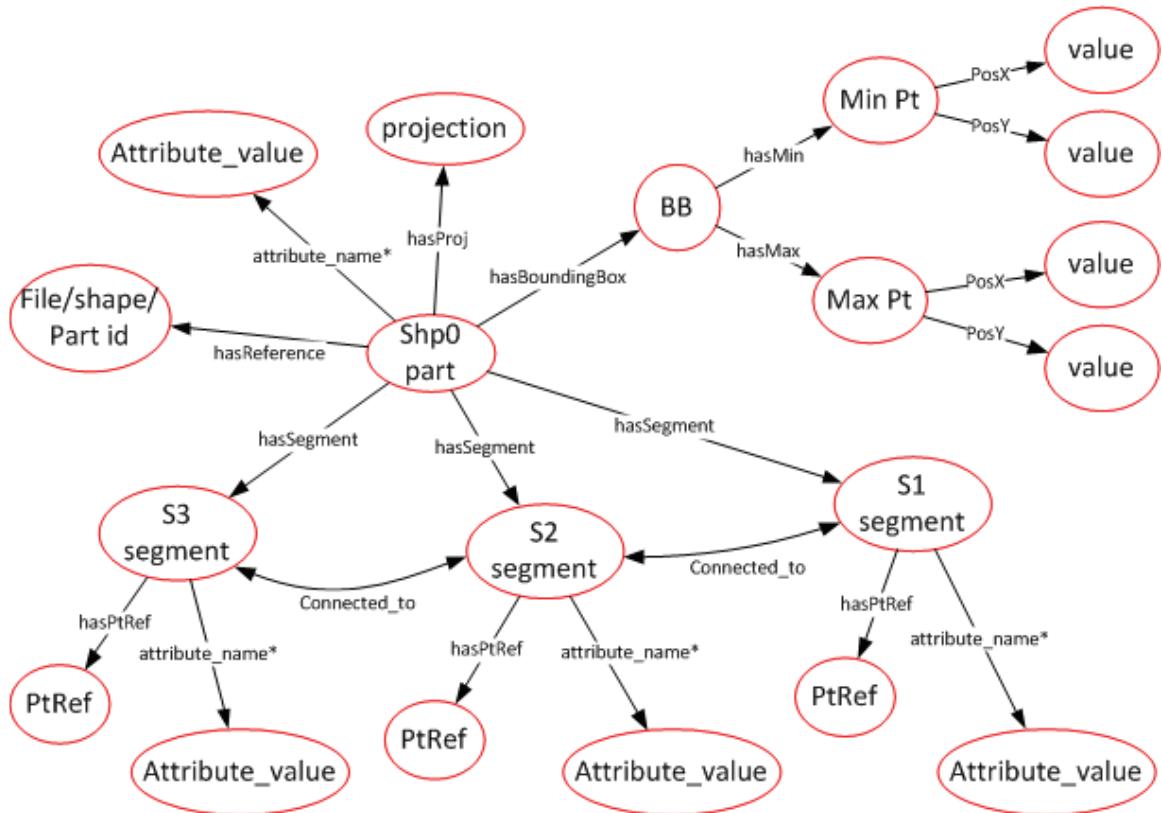


Figure 29: Knowledge Graph Defining Shp0

Our Initial Mapping procedure defined in section 5.3.1 generates the knowledge graph for Shp0 as shown in Figure 29 and adds it as facts to the knowledge base. The part is of type *Connected\_Segments* which is equivalent to a *Thoroughfare* entity. Each segment  $s$  is added as an instance to the KB. Relationships are evaluated between available instances and added as facts to the KB. In this case, no relationships exist

except *Connected\_To* which was added by the Initial Mapping procedure. Each segment is equivalent to a *Thoroughfare\_part* which is also defined as Road in TD and has the properties shown previously in Figure 18. Data properties of each instance are extracted using the Data Extractors framework. Initial data (mapped by Initial Mapping) is generally considered certain unless otherwise specified as part of the record's attributes and mapped with an appropriate SD ontology. Some data extractors such as the Start/End Edge, Directionality, or others associate some certainties with added facts. Every extractor has a handle on the knowledge base and the collection of GIS data sources as well as the entity (in case of data properties) or entities (in case of binary object properties) for use in the evaluation. Any fact found by the extractor gets added to the KB with associated uncertainties as annotations. The system calculates the certainties of inferences based on available explanations under Zadeh semantics.

Each segment in our case defines a road (part) entity with property values. The result is shown in Figure 28. The same definitions are used by more complex transportation feature types in the system. For example, a bridge entity contains 1 or more segments part of the bridge and each segment defines a road (part). This allows the generation of changing road definitions such as a secondary road becoming a major road at some location or a two-way road becoming one way.

## 7.2 Uncertainty of Inferences

**Objective:** Associate uncertainties using KB explanation services.

We show here some results on the addition of uncertainties to KB assertions and computing the resulting uncertainties based on KB explanation services described earlier in Chapter 6. The knowledge base in this example contains 51 axioms (for brevity) including 8 axioms with certainty annotations. *uncertaintyVariable* is an annotation defined by our process that can be used on all axiom types. Consider the following axioms:

```

Monument ≡ Area ⊓ Ancient
Park ⊑ Area
Street ≡ Thoroughfare ⊓ ∃within.BuiltUpArea
Avenue ≡ Thoroughfare ⊓ ∃≥1NextTo.(Monument ⊔ Park)
Boulevard ≡ Avenue ⊓ ∃Number_Of_Lanes.(xsd:short:> 2)
Bridge ≡ Thoroughfare ⊓ ∃Over.ImpassableArea
CoveredBridge ≡ Bridge ⊓ ∃hasCover.Cover
Cover ≡ Wood ⊔ Metal
Metal ⊑ Cover
Wood ⊑ Cover
Disjoint(Metal,Wood)
i1:Thoroughfare
(0.6,(i1,i2):NextTo):uncertaintyVariable
(0.4,(i1,i3):NextTo):uncertaintyVariable
(0.6,i2:Ancient):uncertaintyVariable
(0.8,i2:Area):uncertaintyVariable
i3:Park
(1.0,i4:Avenue):uncertaintyVariable
(0.6,(i4,i2):NextTo):uncertaintyVariable
(0.5,(i4,i3):NextTo):uncertaintyVariable
(0.8,(i4,3):Number_Of_Lanes):uncertaintyVariable

```

After Realization using our *ConsistentKB* procedure, the instances in the knowledge base are reclassified as follows with calculated certainties:

```

(0.8,i4:Boulevard):uncertaintyVariable
(1.0,i4:Avenue):uncertaintyVariable
(1.0,i3:Park):uncertaintyVariable
(0.8,i2:Area):uncertaintyVariable
(0.6,i2:Monument):uncertaintyVariable
(0.6,i2:Ancient):uncertaintyVariable
(0.6,i1:Avenue):uncertaintyVariable

```

The following is the program output showing details of inferences and their explanations as well as the calculation of resulting uncertainties as per the process discussed in Chapter 6. Verification of facts can be done for example by the user as shown in Figure 30.

```

Loaded OntologyID(OntologyIRI(<http://se>))
Classifying ...
    ... finished
Realizing ...
    ... finished

individual: <http://se#i4>
for type assertion: Boulevard
explanation 1/3:
    ClassAssertion(<http://se#Avenue> <http://se#i4>)
        with certainty value: 1.0
    DataPropertyAssertion(<http://se#no_of_lanes> <http://se#i4> "3"^^xsd:short)
        with certainty value: 0.8
explanation 1/3 axioms result(min): 0.8
explanation 2/3:
    ObjectPropertyAssertion(<http://se#Next_to> <http://se#i4> <http://se#i3>)
        with certainty value: 0.5
    DataPropertyAssertion(<http://se#no_of_lanes> <http://se#i4> "3"^^xsd:short)
        with certainty value: 0.8
explanation 2/3 axioms result(min): 0.5
explanation 3/3:
    ObjectPropertyAssertion(<http://se#Next_to> <http://se#i4> <http://se#i2>)
        with certainty value: 0.6
    DataPropertyAssertion(<http://se#no_of_lanes> <http://se#i4> "3"^^xsd:short)
        with certainty value: 0.8
explanation 3/3 axioms result(min): 0.6
all explanations (3) result(max): 0.8

individual: <http://se#i3>
for type assertion: Park
explanation 1/1:
    ClassAssertion(<http://se#Park> <http://se#i3>)
explanation 1/1 axioms result(min): 1.0
all explanations (1) result(max): 1.0

individual: <http://se#i2>
for type assertion: Monument
explanation 1/1:
    ClassAssertion(<http://se#Area> <http://se#i2>)
        with certainty value: 0.8
    ClassAssertion(<http://se#Ancient> <http://se#i2>)
        with certainty value: 0.6
explanation 1/1 axioms result(min): 0.6
all explanations (1) result(max): 0.6

individual: <http://se#i1>
for type assertion: Avenue
explanation 1/4:
    ObjectPropertyAssertion(<http://se#Next_to> <http://se#i1> <http://se#i2>)

```

```

        with certainty value: 0.6
    ClassAssertion(<http://se#Thoroughfare> <http://se#i1>)
explanation 1/4 axioms result(min): 0.6
explanation 2/4:
    ObjectPropertyAssertion(<http://se#Next_to> <http://se#i1> <http://se#i3>)
        with certainty value: 0.4
    ClassAssertion(<http://se#Thoroughfare> <http://se#i1>)
explanation 2/4 axioms result(min): 0.4
explanation 3/4:
    ObjectPropertyAssertion(<http://se#Next_to> <http://se#i1> <http://se#i3>)
        with certainty value: 0.4
    ClassAssertion(<http://se#Thoroughfare> <http://se#i1>)
explanation 3/4 axioms result(min): 0.4
explanation 4/4:
    ObjectPropertyAssertion(<http://se#Next_to> <http://se#i1> <http://se#i2>)
        with certainty value: 0.6
    ClassAssertion(<http://se#Thoroughfare> <http://se#i1>)
explanation 4/4 axioms result(min): 0.6
all explanations (4) result(max): 0.6

Consistent: true
file saved testBoulevard2.owl

The classified Hierarchy is:
{owl:Thing}
  {<http://se#Bridge>}
    {<http://se#CoveredBridge>}
  {<http://se#Cover>}
    {<http://se#Wood>}
    {<http://se#Metal>}
  {<http://se#Ancient>}
    {<http://se#Monument>}
  {<http://se#Area>}
    {<http://se#Monument>}
    {<http://se#Park>}
  {<http://se#Thoroughfare>}
    {<http://se#Avenue>}
    {<http://se#Boulevard>}

Printing individuals list:
individual <http://se#i4>
has type ([<http://se#Boulevard>]) with certainty value 0.8
has type ([<http://se#Avenue>]) with certainty value 1.0
individual <http://se#i3>
has type ([<http://se#Park>])
has type ([<http://se#Park>]) with certainty value 1.0
individual <http://se#i2>
has type ([<http://se#Area>]) with certainty value 0.8
has type ([<http://se#Monument>]) with certainty value 0.6
has type ([<http://se#Ancient>]) with certainty value 0.6
individual <http://se#i1>
has type ([<http://se#Avenue>]) with certainty value 0.6
has type ([<http://se#Thoroughfare>])

```

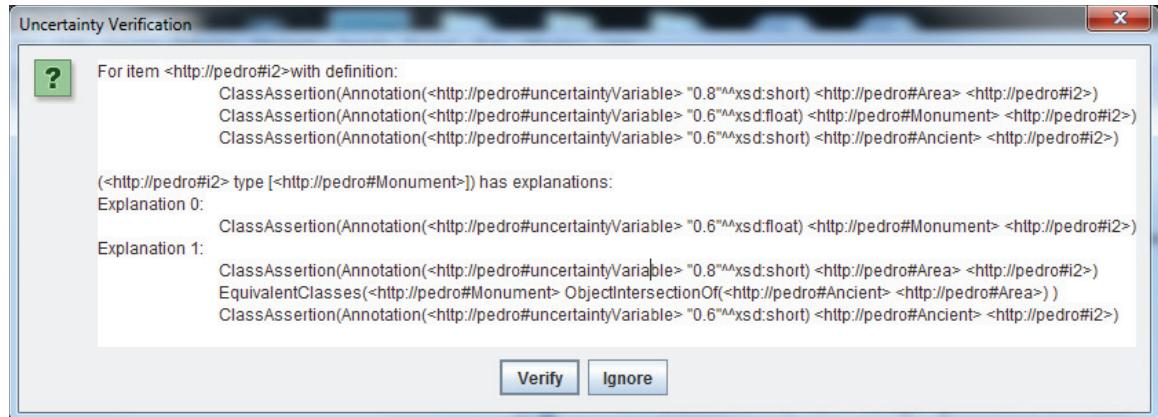


Figure 30: Example Dialog Querying User for Fact Verification

### 7.3 A Bridge Example in Honolulu

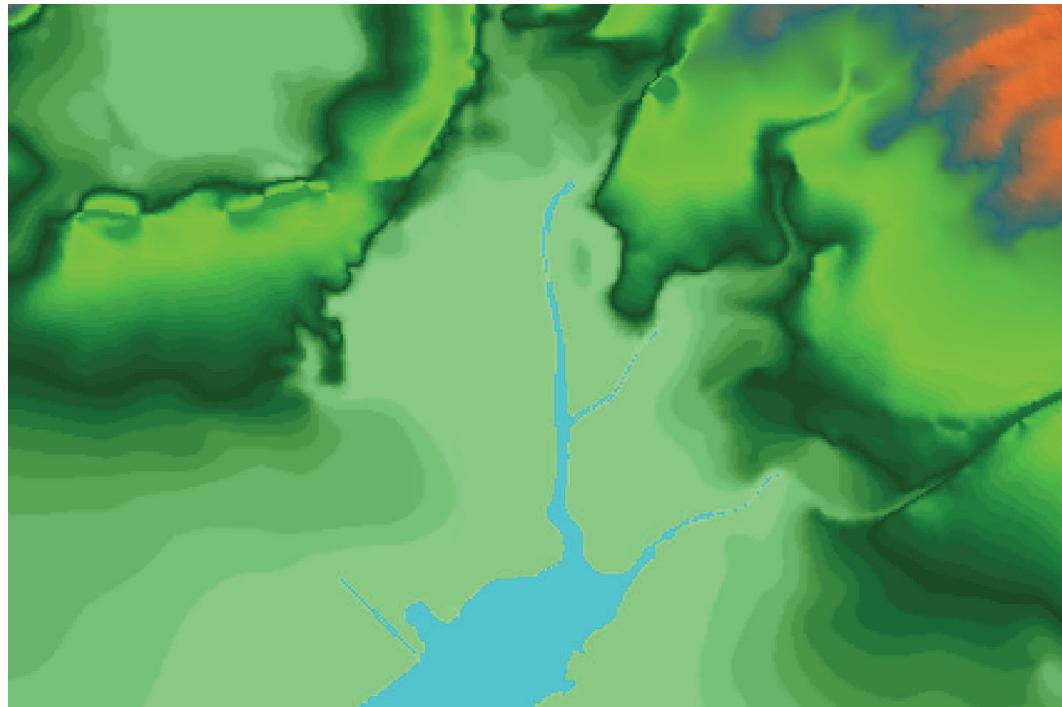


Figure 31: 10m Resolution Elevation Data



Figure 32: Shapefile Data Sets in Points, Linears and Polygons



Figure 33: 1-foot Resolution Partial Imagery Data

**Objective:** Reconstruct a bridge in a region from GIS data sources.

This example details the previously shown examples under section 1.6. The above figures represent data sources of an area in Honolulu, Hawaii for which we were able to acquire publicly available data (NGA, 2014). We are interested in the bridge structure shown at the top right of Figure 33. Figure 34 shows the 1-foot satellite imagery of our ROI along with the correlated line feature. A human can clearly identify the bridge stretching from the bottom-left to the top-right of that area.

Figure 35 shows the same area without the satellite imagery revealing the elevation, the linear and areal features. The elevation source is a DTM (Digital Terrain Map) of type GridFloat. The linear feature does not contain information about its elevation over ground which is typical with raw feature data. However, the linear is defined along the middle of the roadway.

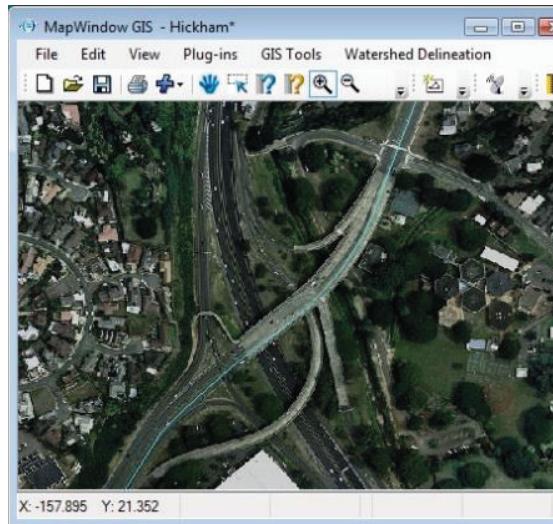


Figure 34: Honolulu Bridge Satellite View

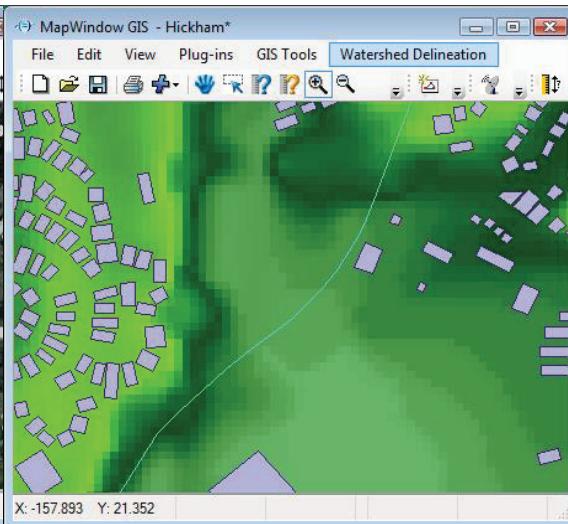


Figure 35: GIS Data of Honolulu Bridge

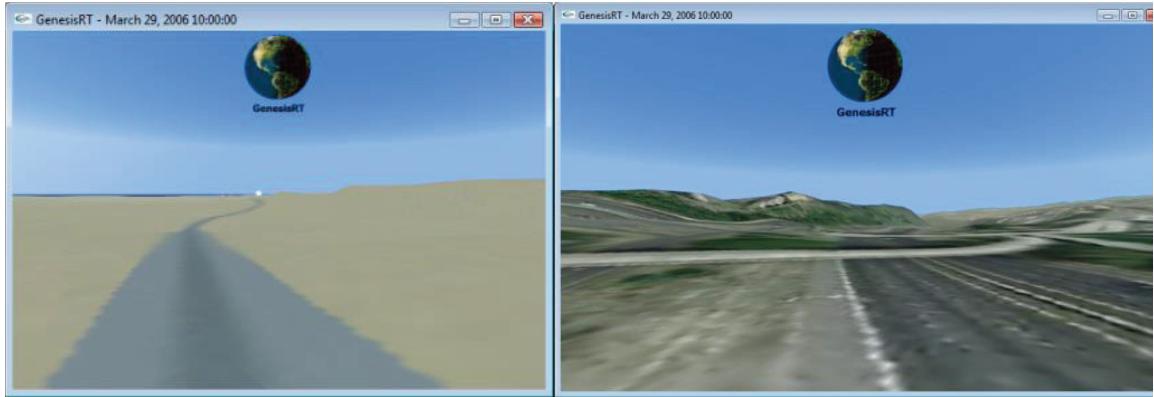


Figure 36: Ground View using GenesisRT    Figure 37: Ground View using GenesisRT without imagery

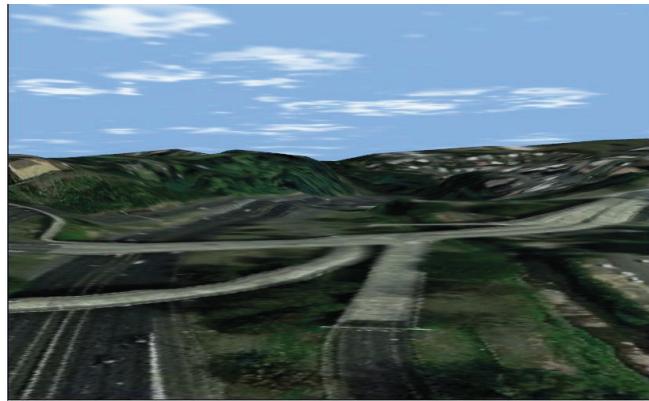


Figure 38: Ground View Visualization using Presagis Terra Vista

Current real-time visualization systems would render this information without modeling a 3D object for the bridge. They wrap the generated terrain using the satellite image as a texture, and if necessary drape a road texture along the linear feature as shown in Figure 36. Two state-of-the-art tools, DVC GenesisRT and Presagis TerraVista, were used to generate a runtime 3D model of the terrain using the GIS data sources and some configuration files that we implemented. When the systems have completely finished loading with satellite imagery as texture, the fidelity presented is shown in Figure 37 and Figure 38 respectively. In this case, there is very short turnaround time (as defined in Chapter 1) between acquiring the GIS data sources and having a usable 3D

visualization of the data; little manual attribution or content authoring was done to achieve these results.

In our system, after entities are mapped using the Initial Mapping procedure to knowledge base instances and while evaluating instance relationships, the *Impassable\_Through* relationship extractor (on the *Connected\_Segments* instances) uses elevation and imagery data along with instance information to automatically determine if an entity or part of it spatially crosses another known entity or feature such as some impassable terrain profile. The TBox is defined as follows:

<b>Classes</b>	<b>Roles</b>
Bridge	Over
Thoroughfare	Impassable_Through
Segment	Impassable_By
ImpassableArea	Connected_To
Connected_Segments	
<b>Properties</b>	<b>Axioms</b>
See Appendix A	$\text{Connected\_Segments} \equiv \text{Thoroughfare}$ $\text{Bridge} \equiv \text{Thoroughfare} \sqcap \exists \text{Over}. \text{ImpassableArea}$ $\text{ImpassableArea} \equiv \exists \text{Impassable\_By}. \text{Connected\_Segments}$ $\text{Impassable\_By} \equiv \text{inverseOf}(\text{Impassable\_Through})$

The Shapefile GIS source layer given as input in this example defines the line feature record as *ShapeID* = 4, *ShapeType* = *SHP\_POLYLINE*, and *numPoints* = 26. Here is a subset of the points that we are concerned with. They are listed in the file using WGS84 projection (defined in the file):

(-157.896959,21.348452)		(-157.896704,21.348709)		(-157.896440,21.348956)
(-157.896166,21.349193)		(-157.895860,21.349429)		(-157.895595,21.349641)
(-157.895406,21.349792)		(-157.895228,21.349943)		(-157.895063,21.350116)
(-157.894912,21.350301)		(-157.894775,21.350497)		(-157.894652,21.350702)
(-157.894546,21.350916)		(-157.894474,21.351122)		(-157.893998,21.352351)

Figure 39 shows the curve returned by sampling the elevation data under the concerned segments with  $y$  as elevations and  $x$  as linear feature length. The solid line section of the graph represents the plot of the latitude/longitude points used in this example. The steep slopes under the bridge entity are recognizable and span around 450m (between  $x=1500$  and  $x=1950$ ) with elevation differences of almost 20m (between lowest and highest points). A new instance  $a$  of type *Area* (defined between points 2 and 15) and an instance  $b$  of type *Connected\_Segments*, which includes the affected segments, are added as new facts to the knowledge base.  $a$  and  $b$ 's relationships including *Impassable\_Through* relationships between  $b$  and  $a$  are added to the knowledge base along with associated certainties. Other relationship extractors are then evaluated as required. Since relationships of  $a$  are also evaluated, then the *Over* relationship based on DE-9IM should be part of the knowledge base for any segment part of  $b$  as well as for  $b$  itself.

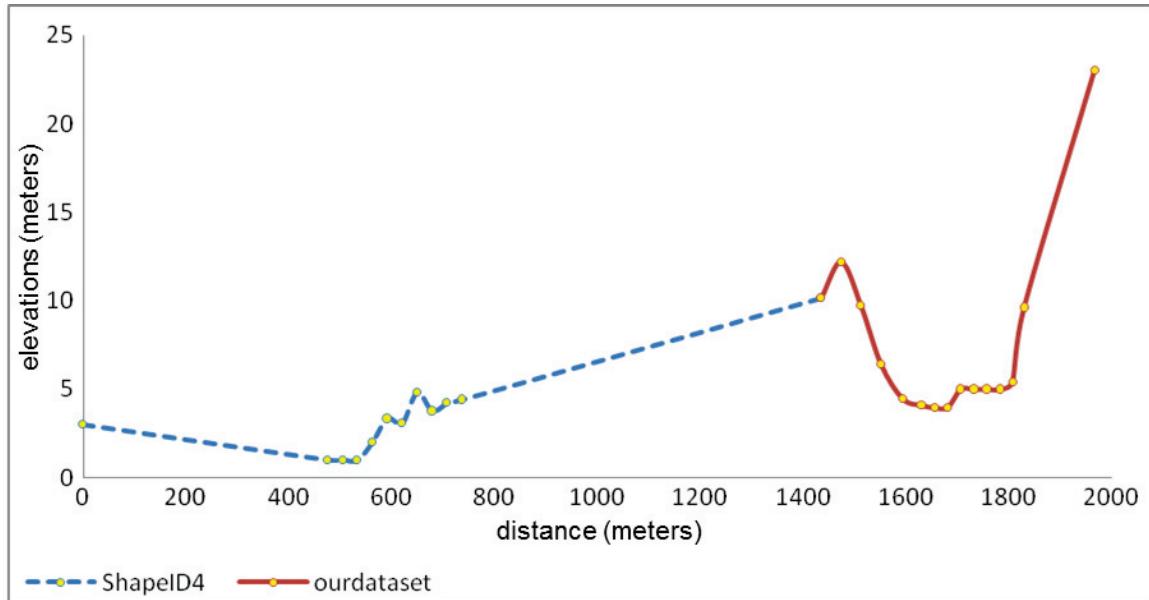


Figure 39: Elevation Profile of ShapeID 4

The following facts are added by our system to the knowledge base:  $(s_1, b):Part\_Of$ ,  $(s_1, a):over$ ,  $(b, a):Impassable\_Through$ ,  $(s_2, b):Part\_Of$ ,  $(s_2, a):over$ ,  $(b, a):over$ , and so on. Note that  $(s_1, s_2):Connected\_To$  was added by the initial mapping process. After realizing the knowledge base and if it is consistent, some facts become explicit such as:  $(b, a):Impassable\_Through \models (a, b):Impassable\_By \models a:ImpassableArea \models b:Bridge$ . The system now has identified a series of segments as a bridge instance and made it explicit in the knowledge base. Data property extractors are then executed to fill the property values required by the bridge  $b$ .

Finally, Using the RC ontology, the Geometry Definition Engine extracts data from the knowledge base and explicitly defines the geometry and parameters for each processed element from the ABox. It retrieves the specific class type of instance  $b$  from the knowledge base using the query in Figure 15 and selects the corresponding definition in the RC ontology. It uses the generic SPARQL query described in Figure 16 to identify and extract all necessary property values for  $b$ . Figure 40 shows the results returned by the queries.

We used the Procedural Bridge Framework from Presagis Creator Studio® to represent our definition. A 3D model of the bridge is created and shown in North view in Figure 41. The inferred type is used to select the Cantilever Beam bridge parameter model algorithm. The remaining property values are used as parameters for the selected procedural modeling algorithm to generate the needed representation of the feature instance. The result of our process within a scene was shown previously in Figure 3 in Chapter 1.

<b>Basic Properties</b>	<b>Value</b>	<b>Explanation</b>
Bridge Type	Cantilever Beam	
Start Edge	(21.348452, -157.896959, 10)	(latitude, longitude, altitude)
Start Edge Width	28m	
Start Angles	43.04, 3	(horizontal, climb) in degrees
End Edge	(21.351122, -157.894474, 23)	(latitude, longitude, altitude)
End Edge Width	28m	
End Angle	67.31, 0	(horizontal, climb) in degrees
Span Dividers	5	5 supports
Start/End Dividers	Enabled	Form closed sections at extremities
Ground Height	-20m	
Round Subdivisions	10	
Deck Thickness	1m	
Deck Subdivisions	14	14 segments defining bridge
Left/Right overhang	1m	
Overhang Height	1m	
<b>Supports</b>	<b>Value</b>	
Type	Pillar	
Partitions	2	Two visible pillars at each divider
Webbing	N/A	
Sections	N/A	
Gap Size	N/A	
Top Scale Ratio	N/A	
Width/Height	12, 5	(width, depth) in meters

Figure 40: Retrieved Values for the Honolulu Bridge

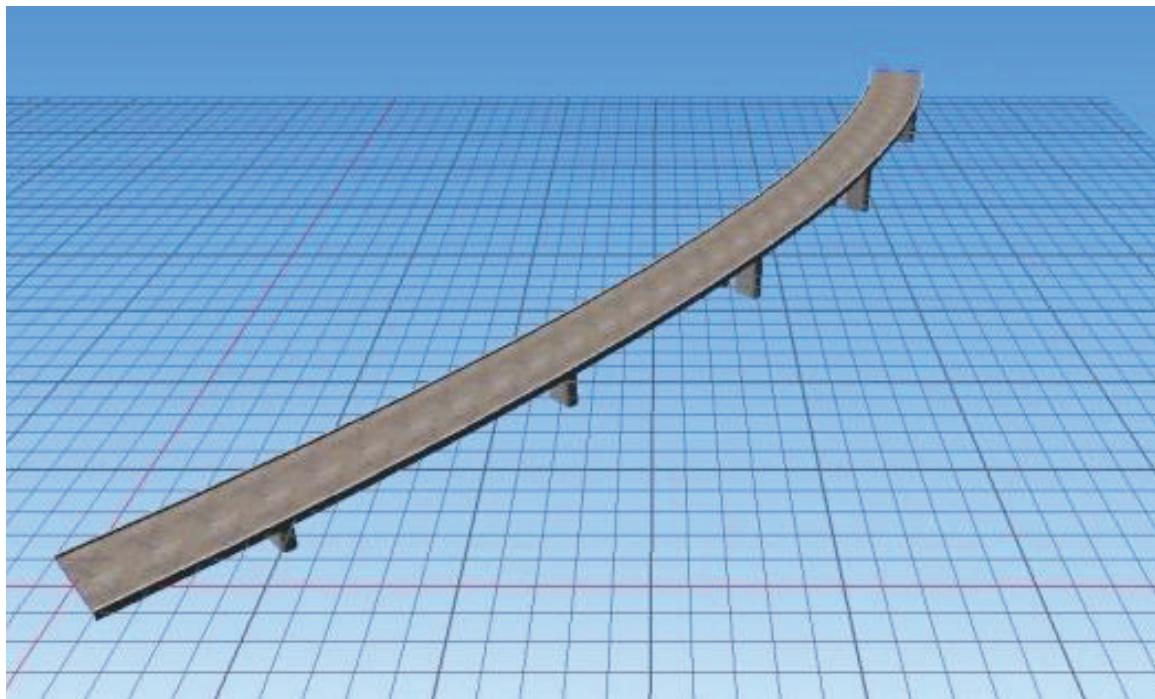


Figure 41: Generated Cantilever Beam (North View)

## 7.4 An Overpass

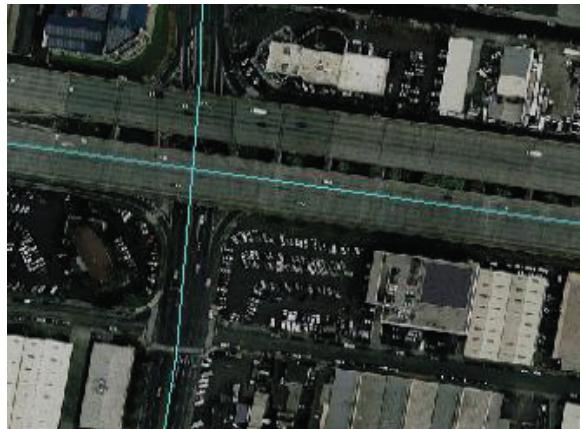


Figure 42: Overpass Areal View

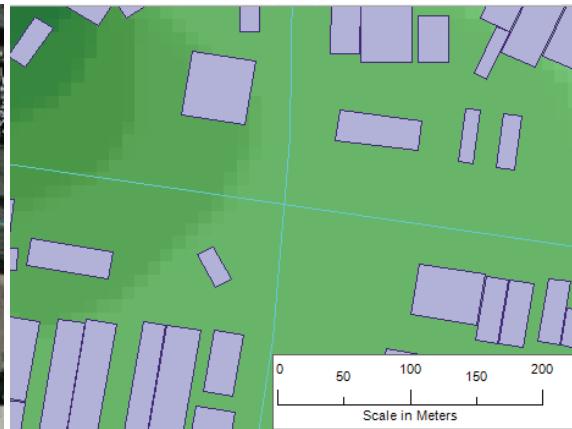


Figure 43: Overpass GIS Data Sources

Basic Properties	Value	Explanation
Bridge Type	Cantilever Beam	
Start Edge	(21.335418, -157.901399, 8)	(latitude, longitude, altitude)
Start Edge Width	25m	
Start Angles	0, 0	(horizontal, climb) in degrees
End Edge	(21.335418, -157.901399, 8)	(latitude, longitude, altitude)
End Edge Width	25m	
End Angle	0, 0	(horizontal, climb) in degrees
Span Dividers	10	10 supports
Start/End Dividers	Enabled	Form closed sections at extremities
Ground Height	-9m	
Round Subdivisions	10	
Deck Thickness	1m	
Deck Subdivisions	1	1 segments defining bridge
Left/Right overhang	1m	
Overhang Height	1m	
Supports	Value	
Type	Pillar	
Partitions	2	Two visible pillars at each divider
Webbing	N/A	
Sections	N/A	
Gap Size	N/A	
Top Scale Ratio	N/A	
Width/Height	5, 3	(width, depth) in meters

Figure 44: Retrieved Values for Overpass

**Objective:** Reconstruct an overpass in a region from GIS data sources.

In this example, also in Honolulu, Hawaii, the DE-9IM relationship extractors analyze the spatial layouts between entities and the two orthogonal segments shown in this example define a *crosses* relationship. Also, the *PrecedenceByImagery* extractor is executed by the *Over* relationship extractor when a *crosses* relationship exists and no altitude values are available.

The *PrecedenceByImagery* extractor could use the segment definitions to scan the segmented imagery and return which overlays the other based on continuity. In this case, the horizontal segment is defined to be *Over* the vertical one and therefore a bridge is inferred. The data extractors fill the necessary property values as shown in Figure 44. The result of the representation using our process is shown in Figure 45.

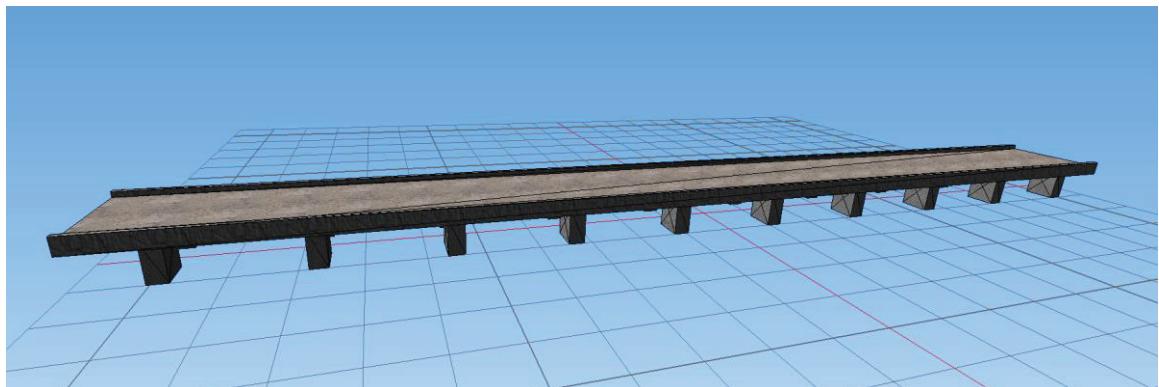


Figure 45: Generated Overpass

## 7.5 Overwater Bridge

**Objective:** Reconstruct an overwater bridge in a region from GIS data sources.

In this example (also in Honolulu), the elevations below the feature record shown in Figure 47 define water level values (bathymetric elevations). The *Altitude* or *Over* extractors detect these values and add  $(s, \text{waterbody}): \text{Over}$  to the KB where  $s$  is the concerned segment and *waterbody* is a persistent instance in the knowledge base relating any element that is defined over a water body. A bridge is similarly inferred and generated provided that  $\{\text{waterbody}\}$  has type *ImpassableArea*.

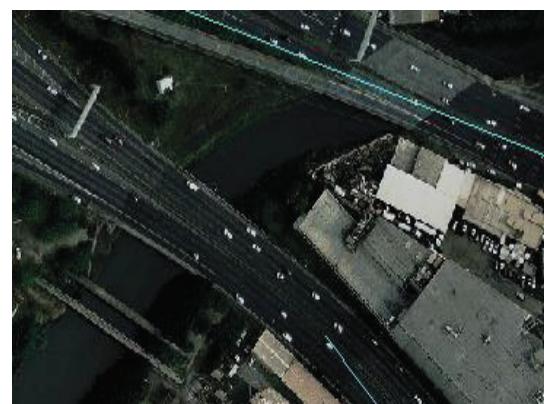


Figure 46: Overwater Bridge Areal View

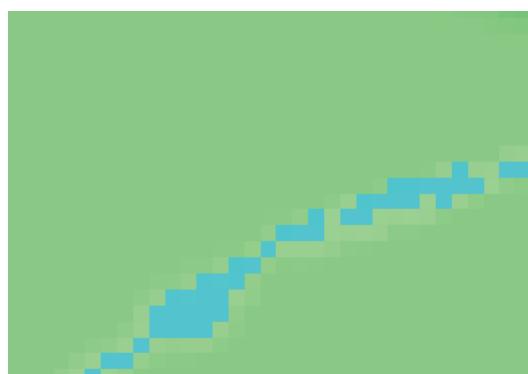


Figure 47: Overwater Elevation Data Source

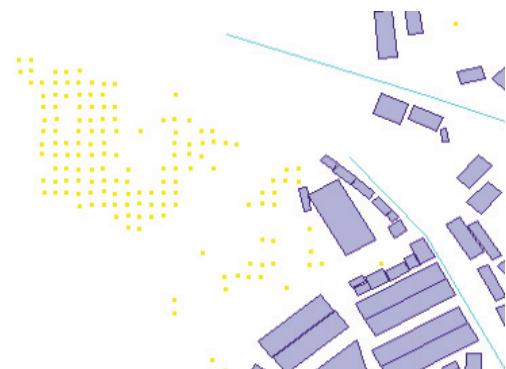


Figure 48: Overwater Road, Building, and Tree Features Combined

## **7.6 The Champlain Bridge**



Figure 49: Side View of the Champlain Bridge

**Objective:** Reconstruct a heritage bridge, the Champlain bridge in Montreal.

The Montreal Champlain Bridge is a 1957 construction of type steel truss cantilever made from pre-stressed concrete beams and deck [Wikipedia, 2014]. We selected this bridge due to the complexity of its representation and as it is a symbol of Montreal and its art. The length of the main part of the bridge is 7.412km (14.5km with approaches). Most of the bridge is a multi-span structure similar and can be generated similarly to what we have shown in section 7.4. For this example, we are interested in the steel superstructure part of the bridge, referred to as Section 6 of the bridge shown in Figure 49 between supports 1 and 4. Although its main architecture is based on steel truss

cantilever, the bridge is a complex architecture of several basic bridge types fused together, which is more common with modern designs and iconic landmarks. Such designs are hard to generate automatically because they include an artistic element that makes them unique. We however identified 3 basic types, referring to the types listed in Appendix A, that approximate the bridge's 3D representation: (1) a Tied Arch bridge type between supports 2 and 3 (curved arch middle section) referred to as the Middle Span, (2) a Cantilever Open Spandrel type (the under structure of the bridge) between supports 1 and 4, and (3) a Cantilever Through Arc type (the over steel structure) between supports 1 and 4.

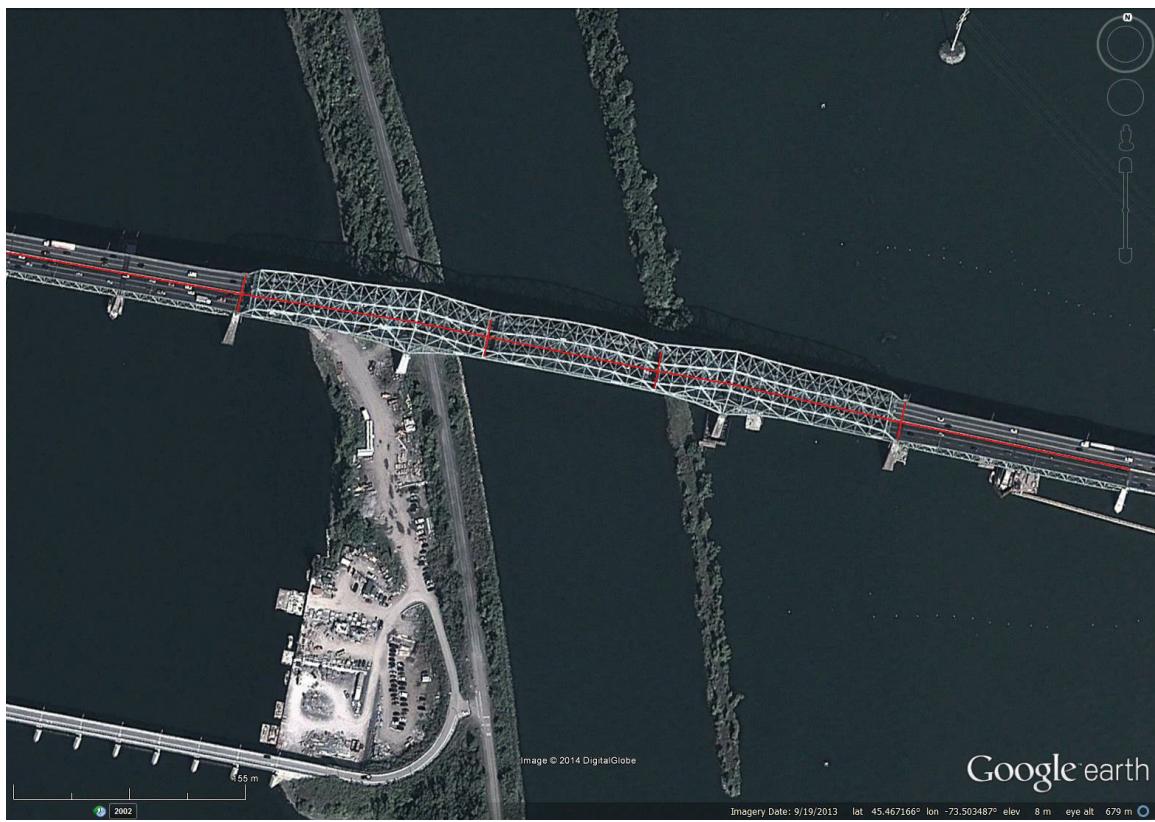


Figure 50: Overlay with Shape Data (Google Earth®)

The linear shape definition defining the spans is overlaid on top of the aerial imagery (in red) as shown in Figure 50 with four noted locations (cross edges).

While some data is available about Section 6 from the official bridge site (<http://jccbi.ca/bridges-structures/champlain-bridge/technical-data/>), extractors from our framework retrieve the missing information. Technical data facts of the Champlain Bridge Section 6 and the Middle Span are presented below. We also show the property values and the corresponding generated representations after executing our process.

Type	Cantilever with center suspended span		
Material:	Steel Superstructure, Concrete beams and deck		
Width:	23.1648 meters		
Lanes:	6, with separators		
Height above water:	36.6 meters		
Section 6 Span	Point Location	Latitude	Longitude
	South East	45.467381	-73.500701
	South West	45.467157	-73.500746
	North East	45.467841	-73.506422
	North West	45.467618	-73.506472
Section 6 Length:	450.5 meters		
Longest Span Length:	215.5 meters (includes Middle Span)		
Anchor Arms (2):	117.5 meters		
Section 6 Middle Span	South East	45.467751	-73.502824
	South West	45.467349	-73.502873
	North East	45.467689	-73.504295
	North West	45.467466	-73.504335
Middle Span Length:	116 meters		

We retrieved the properties below and generated the middle span (Figure 51) using edges 2 and 3 (segment 3 of the linear) shown in Figure 50:

<b>Basic Properties</b>	<b>Value</b>	
Bridge Type	Tied Arch	
Span Dividers	0	
Start/End Dividers	Disabled	
Ground Height	-36.6m	
Round Subdivisions	10	
Angles Start/End	0 degrees	
Deck Thickness	1m	
Deck Subdivisions	1	
Left/Right overhang	1m	
Overhang Height	1m	

<b>Arch Profile</b>	<b>Value</b>	<b>Explanation</b>
Arch Height	20m	
Displacement	20m	Middle Span height
Curviness	100%	
Half Span Supports	5	Middle Span has 10 sections in total
Support Subdivision	10	Affects arch support bending
Arch Supports	0.2m	All support rods are set to 0.2m width and height
Main Supports	0.5m	Thicker main supports
Round Supports	false	None are round/cylindrical

<b>Arch Webbing</b>	<b>Value</b>	
Partitions	2	1 middle separator
Outer Cables	1	A single side webbing
Inner Cables	1	A single side webbing
Top Scale Ratio	50%	
Clearance	4.0m	Roadway Clearance
Webbing Top	Type X	X structure viewed from top
Webbing Side	Subdivided Warren	Webbing viewed from sides
Webbing Vertical	Variant 2	The webbing between the clearance and arch top
Vertical Sections	1	Unknown details, set to 1 by default



Figure 51: Tied Arch Bridge using Edges 2 and 3

Similarly, using edges 1 and 4, we retrieve the necessary properties and generate the open spandrel (Figure 52). Segments 2, 3, and 4 in this case define a single KB instance. Properties listed in Appendix A and which are not mentioned in the listings are not applicable, or use previously defined values (unchanged). The result is shown merged with the previously defined part.

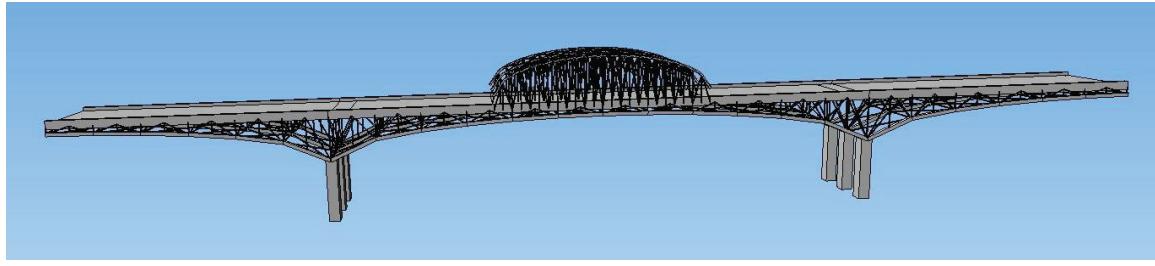


Figure 52: Cantilever Open Spandrel using Edges 1 and 4

<b>Basic Properties</b>	<b>Value</b>	<b>Explanation</b>
Bridge Type	Cantilever Open Spandrel	
Span Dividers	2	Two sections, two supports
Start/End Dividers	Enabled	Form closed sections at extremities
Ground Height	-36.6m	
Round Subdivisions	10	
<b>Supports</b>	<b>Value</b>	
Type	Pillar	
Partitions	3	Three visible pillars at each divider
Webbing	N/A	
Sections	N/A	
Gap Size	N/A	
Top Scale Ratio	N/A	
Width/Height	4m	4m each side
<b>Arch Profile</b>	<b>Value</b>	
Arch Height	15m	
Displacement	3m	Separation with top of arch
Curviness	100%	
Half Span Supports	5	Span has 10 sections in total
Support Subdivision	6	6 rods defining arc
Arch Supports	0.2m	All support rods are set to 0.2m width and height
Main Supports	0.5m	Thicker main supports
Round Supports	false	None are round/cylindrical
<b>Arch Webbing</b>	<b>Value</b>	
Partitions	2	1 middle separator
Outer Cables	1	A single side webbing
Inner Cables	1	A single side webbing
Top Scale Ratio	50%	
Clearance	4.0m	Roadway Clearance
Webbing Top	Type X	X structure viewed from top
Webbing Side	Subdivided Warren	Webbing viewed from sides
Webbing Vertical	Variant 2	The webbing between the clearance and arch top
Vertical Sections	1	Unknown details, set to 1 by default

Finally, the cantilever truss bridge is defined on edges 1 and 4 again. The final result is shown merged with the previously defined parts in Figure 53 and Figure 54.



Figure 53: Cantilever Through Arc using Edges 1 and 4

<b>Basic Properties</b>	<b>Value</b>	<b>Explanation</b>
Bridge Type	Cantilever Through Arch	
Span Dividers	2	Two sections
<b>Arch Profile</b>	<b>Value</b>	
Arch Height	25m	
Displacement	25m	Separation with top of arch
Curviness	100%	
Half Span Supports	10	Span has 10 sections in total
Support Subdivision	2	Actual subdivisions define a curvature in real world entity, but due to a limitation we approximate these.
Arch Supports	0.2m	All support rods are set to 0.2m width and height
Main Supports	0.5m	Thicker main supports
Round Supports	false	None are round/cylindrical
<b>Arch Webbing</b>	<b>Value</b>	
Partitions	2	1 middle separator
Outer Cables	1	A single side webbing
Inner Cables	1	A single side webbing
Top Scale Ratio	50%	
Clearance	4.0m	Roadway Clearance
Webbing Top	Type X	X structure viewed from top
Webbing Side	Subdivided Warren	Webbing viewed from sides
Webbing Vertical	Variant 2	The webbing between the clearance and arch top
Vertical Sections	1	Unknown details, set to 1 by default

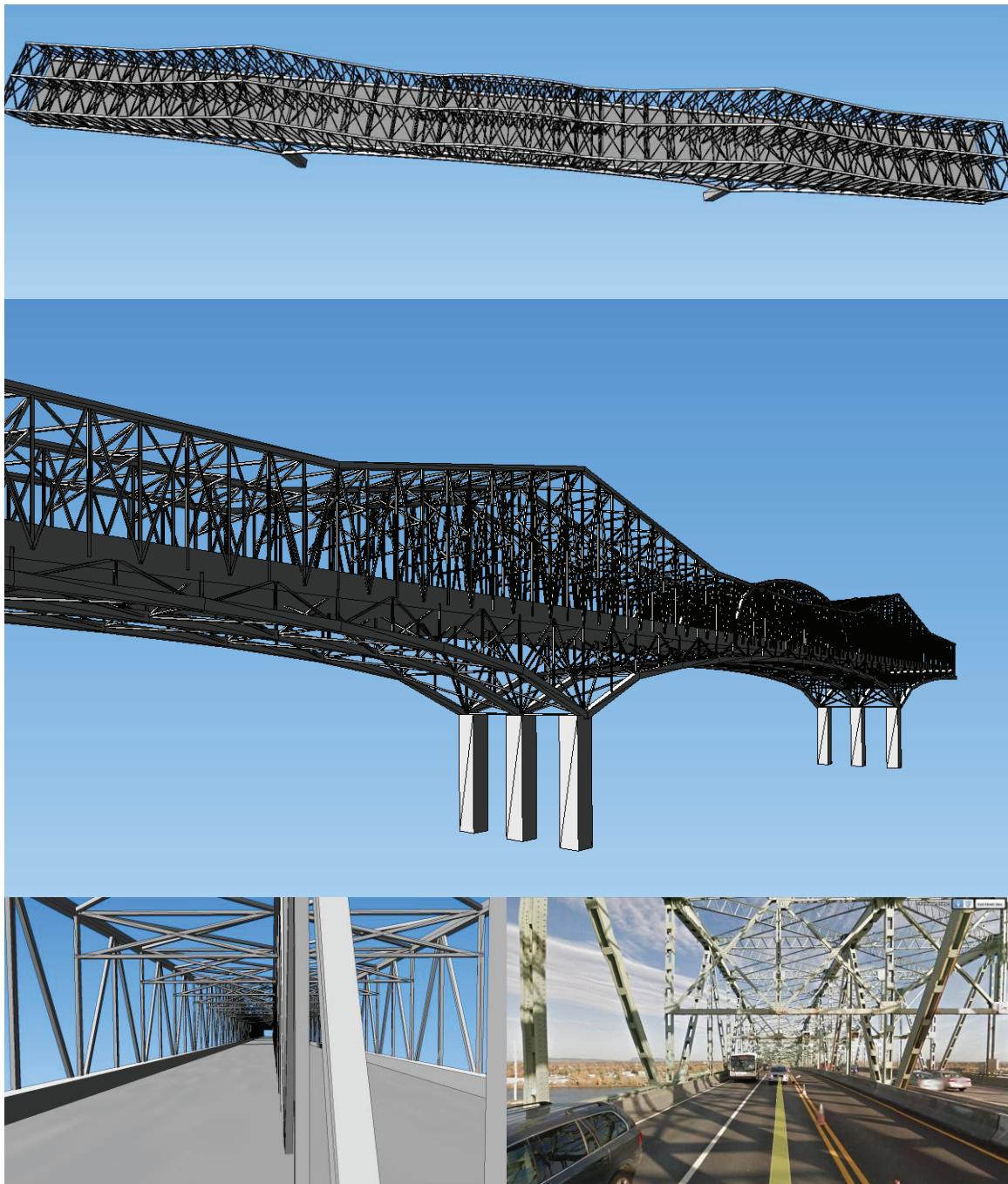


Figure 54: Champlain Bridge Results

## 7.7 School Zone Example: Ste. Marguerite

**Objective:** Find all Thoroughfare entities but not highway entities that are less than 500 meters distance from a school.

Using  $SROIQ^{(D)}$  data properties along with our data extractors framework, we can define a new property for all Thoroughfare instances (representing entities):  $Distance\_To\_School$ . However, in order to restrict the processing domain and since our objective only requires ( $Thoroughfare \sqcap \neg Highway$ ), we define this property for that specific class conjunction only. We add this new property to the terminology with the following definition:

**Data Property Name:**  $Distance\_To\_School$

**Domain:**  $Thoroughfare \sqcap \neg Highway$

**Range:** Double

**Annotation Extractor:**  $dist\_to\_school$

We also added a data extractor called  $dist\_to\_school$  associated as an annotation with the new property as defined in the TBox property definition above. Using known school entities from the KB,  $S_{GIS}$  for layout, and the entity's definition, this extractor will retrieve the distance to the closest school for the entity. This extractor is defined as follows:

```
Procedure dist_to_school
Input: KB,  $S_{GIS}$ , instance
Output: KB (modified)
//for thoroughfare segment instance
//assuming same working coordinate system here
//get pts (2) of instance
P1 =  $S_{GIS}.get\_ref\_pt(instance);$ 
P2 = P1.next();
//find closest school to segment (considering segment center)
```

```

min_distance = null;
For every SchoolArea in KB
    Vector_to_school = (P1+P2)/2 - SchoolArea.getCenter();
    distance_to_school = length(Vector_to_school);
    If (min_distance == null or min_distance > distance_to_school) then
        min_distance = distance_to_school;
    Endif
End
KB.addDataProp(Distance_To_School, instance, min_distance);
EndProc

```

After executing our process, all *Distance\_To\_School* properties for instances categorized of type (*Thoroughfare*  $\sqcap \neg$ *Highway*) will have values for this property as per our *p\_filling* process.

To find all *Thoroughfare* entities that are or might not be of type *Highway* and that are less than 500 meters distance from a school, we define a SPARQL query as follows (assuming the extractor stored values for the property in meters):

```

Select ?instance
Where {
    ?subclass rdfs:subClassOf* :Thoroughfare .
    ?instance rdf:type ?subclass .
    FILTER NOT EXISTS {?instance rdf:type :Highway .}
    ?instance :Distance_To_School ?distance .
    FILTER (?distance < 500.0)
}

```

Figure 55: Retrieving Entities <500m Distance from a School Using SPARQL

Since we added the *Distance\_To\_School* property only to entities that are defined as (*Thoroughfare*  $\sqcap \neg$ *Highway*) in this example, then the NOT EXISTS filter in the above query is redundant. The query retrieves the entities that we are interested in for this example. This includes entities missing information about being of type *Highway* or not. If all information is known, then {owl:complementOf :Highway} could be used for type filtering. It is also possible to query with properties such as :*next\_to* or :*through* defined

between instances using GeoSPARQL provided these properties were inserted after our *r\_mapping* process.

Another way to retrieve the needed information is possible by adding a new class in the ontology, say *SchoolZoneRoads*, with a similar definition as per this query. This is possible under *SROIQ<sup>(D)</sup>* expressivity, which allows data properties and qualified restrictions in concept equivalence axioms. In that case and after reasoning, all instances we are interested in will be classified of type *SchoolZoneRoads* and we will be able to retrieve all instances of this type representing the real world entities.

We apply the above on a specific ROI around the Sainte Marguerite Primary School in Laval-Des-Rapides, Laval. Figure 56 is taken from Google Earth® at coordinates (45.546851, -73.710239) on April 14, 2014. It represents our ROI. We chose this area due to its proximity to a highway (s72) as well as it being dense enough to showcase the capability. The instances created in the KB are shown in Figure 57. We have a total of 133 segments and a few areas (represented as polygons) including the school, a few parks, and the river.

Note that if we increase the *Distance\_To\_School* value filter, say to 1000m for example, the segments on the south of the river shown in the image will also be returned by the query in this case. To resolve this, we can add a Borough or Region property and use it as a constraint in the query. We can also consider this in the *dist\_to\_school* extractor which could return only values in the required case(s). The results of the query in Figure 55 are shown in Figure 58. Note the missing s72 entity as well as entities further than 500m that do not satisfy the query.

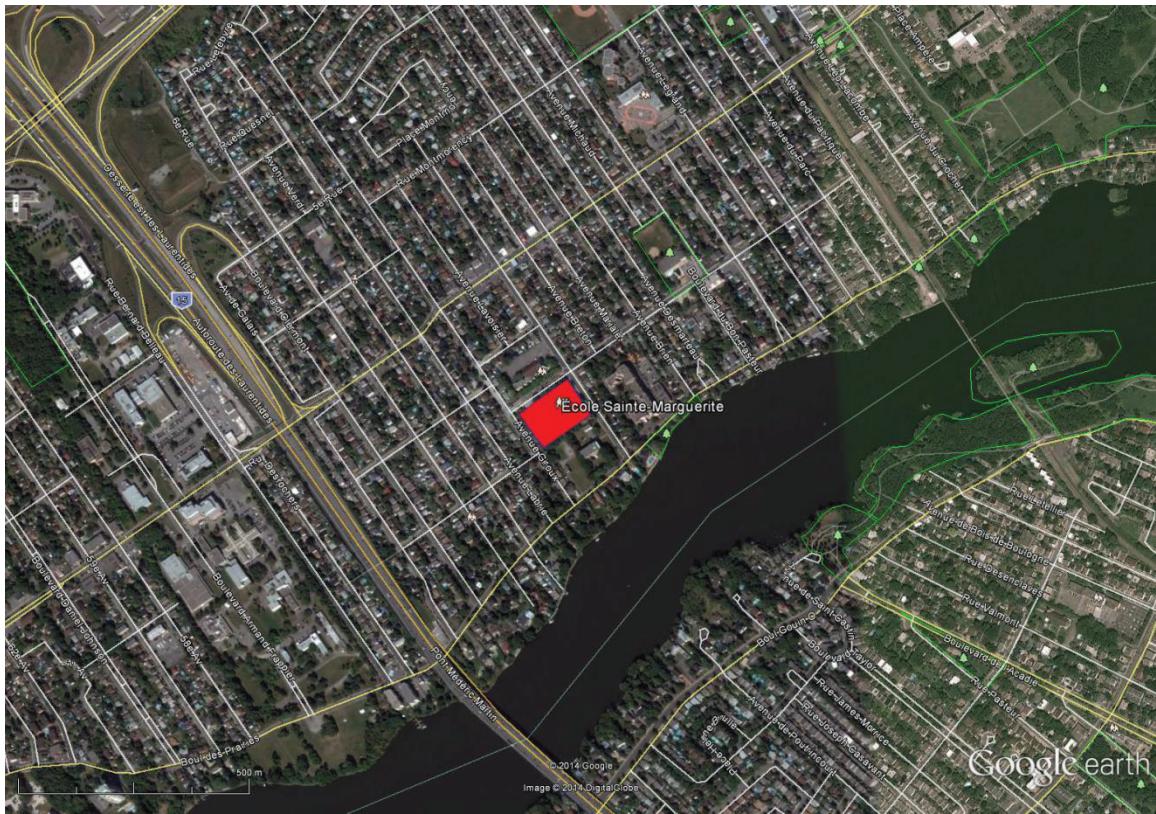


Figure 56: View around Ste Marguerite, Laval-Des-Rapides (Google Earth®)



Figure 57: Instances (133) of ROI layed over Google Maps®

<b>Instance</b>	<b>Distance</b>	<b>Instance</b>	<b>Distance</b>	<b>Instance</b>	<b>Distance</b>
s65	497.98	s102	472.84	s5	193.11
s30	351.91	s128	480.16	s29	330.48
s63	271.99	s9	57.49	s68	204.41
s20	483.37	s45	247.43	s81	336.29
s83	333.32	s94	176.84	s126	482.35
s43	485.04	s82	333.3	s31	417.5
s53	468.35	s8	119.54	s27	385.28
s64	361.88	s91	349.37	s114	248.3
s67	239.13	s112	411.23	s3	272.1
s1	94.59	s116	128.05	s55	483.93
s84	354.92	s115	185.25	s93	218.1
s86	409.36	s6	184.74	s7	104.75
s127	441.47	s80	350.13	s57	368.99
s54	437.46	s2	147.98	s25	286.02
s70	447.73	s16	423.97	s58	427.24
s18	417.17	s69	299.5	s11	262.74
s90	430.74	s92	273.73	s13	185.72
s113	331.62	s47	227.44	s46	same as s6
s110	498.95	s85	376.6	s48	270.49
s50	405.07	s44	387.61	s79	389.41
s49	330.67	s95	309.62	s12	257.6
s56	396.94	s96	380.57	s15	266.51
s14	149.47	s87	446.66	s28	278.3
s52	73.4	s97	475.2	s24	238.96
s10	179.35	s26	336.58	s78	441.37
s51	495.74	s62	306.54		
s88	489.7	s4	492.08		

Figure 58: Results from Query in Figure 55 on Ste Marguerite Primary School

# ***Chapter 8***

## ***Conclusion and Future Work***

### ***8.1 On Achieving Objectives***

Mapping facts in source data to assertions in a knowledge base: We have defined three modular ontologies extending on work by Bitters' VOTT, the OGC's Simple Feature ontology, and GeoSPARQL. A general, low-maintenance, domain ontology hierarchy (TD) specifically describes the transportation domain under the Geographical and Visual Objects domains with concept relationships and properties. Two other ontologies (SD and RC) bridge and map between input/output data (GIS sources and representation definitions respectively) and domain concepts in order to achieve context separation and modularity. If a system capability is to be added or a data format is changed only the related ontology needs to be modified.

Initial GIS source data mapping (or *s\_mapping*) and specifically vector-based features and the facts they define are initially mapped as explicit knowledge base instance assertions using the mapping ontologies and reasoned upon.

Relationship extraction (or *r\_mapping*) evaluates possible relationships (defined by the ontologies) between knowledge base instances and adds further facts to the knowledge base.

Inferring implicit information in the knowledge: The Data Extractors framework (or *p\_filling*) automatically fuses derivable or implicit data from different heterogeneous GIS dataset layers and types. We extracted 3D model properties and other facts that make some implicit information in the GIS source data explicit. Data extractors, associated with property and relationship concepts defined by the ontologies, can be replaced or extended. Each extractor can choose the layer(s) to use based on each used layer's properties such as resolution or type. If in a certain layer the information is missing or unclear, the next best layer can be chosen or different layers can be analyzed to form a single result for the missing property value.

Handling uncertainty: The framework adds all associated data certainties to the extracted facts for use by our extended KB Realization procedure that calculates certainties of inferences based on its KB Justifications.

Procedural creation of detailed digital representations: The components of our process are independent where the extraction of facts step happens separately from the representation definition. For the representation generation from knowledge base facts, we defined the Geometry Definition Engine (or *v\_mapping*). It uses SPARQL querying language and an

ontology of parameterized model definitions (RC ontology) to construct queries, retrieve visual property values, and create definitions that can be used by procedural models. Procedural models construct detailed 3D model representations of entities for subsequent rendering.

Complying with public standards: In our system we use the OWL 2 W3C standard as well as OWL Link, OWL API and SPARQL to achieve communications between subprocesses about knowledge and to manipulate and query the Semantic Web knowledge base. We have extended OWL Link to the .Net Framework and used it as part of our plugins to the open MapWindow GIS C# project which provides tools and interfaces to manipulate the GIS source data input. Our core implementations also use known and open source software projects and APIs.

## ***8.2 Advantages and Limitations***

Using a Semantic Web Knowledge Base provides automated reasoning on the data set with several advantages. However, it adds processing overhead compared to procedural processes based on expressiveness and tableaux methods and completion rules complexities.

Our process reduces expert user involvement by automatically merging data, and detecting and managing inconsistencies in the data. Other advantages include the easier maintenance and definition of the ontology and rule-base with established tools and services provided by existing semantic web reasoners, the possibility of different

applications sharing knowledge in a single knowledge base, taking advantage of new advances in semantic reasoning, OWL and its querying languages, and supporting knowledge manipulation tools and frameworks.

Ordnance Survey has mentioned that queries such as “where is the closest mall next to London’s town hall?” become possible with semantic web. We are using similar queries to define 3D landscape details. The model definition engine is a system agent that constructs queries to complete a definition representing an entity that will then be procedurally generated and positioned in the landscape. It produces queries such as “is instance b a covered bridge span?” and “what is the set of values defining the orientation vector of the covered bridge span b?”

The following sections list the major advantages that are available due to the use of Semantic Web Technology in our process and some of the limitations it imposes.

### ***8.2.1 Inherited Properties***

A. The separation of domain knowledge (TBox), which includes the bridge ontologies, and instances (ABox) defined as part of the ALC syntax is useful for reuse, portability, and dynamic extensions of the knowledge. Domain experts create the TBox once and maintain it separately from the actual ABox. The TBox is organized into hierarchies which allow modularity and bridging between domain and data formats. Legacy processes have domain experts working with data and knowledge. Having separation between TBox and ABox allows:

1. Less dependence on human experts which, after domain knowledge (TD) is defined, would only have to define the mapping ontologies (SD/RC).
2. More users are able to use the process and create the ABox (define instances) based on available definitions in the TBox by using the mapping ontologies.
3. More automation in ABox creation due to already defined machine readable TBox which requires less user involvement (illustrated in SD mapping procedure).
4. Different applications using the same domain can reuse our ontologies. The Knowledge Base can service other types of applications that require knowledge about an ROI.

B. Description Logic formal axiom definition allows the system to be independent of the order in which the user inputs the knowledge and provides services for verifying and querying it. This ensures that the system will reliably always use the most specific semantics automatically and does not required high maintenance based on order of knowledge input. User's therefore focus instead on the semantics of the objects they need to represent. Services allow the automatic verification of the knowledge consistency and coherence. Many tools also exist in manipulating and viewing the knowledge represented in the knowledge base. Using KB inference vs. procedural rules also adds several advantages to our process:

1. Knowledge bases are easier to maintain and debug especially using the services and tools mentioned. This makes adding new properties or change in the defined knowledge or properties easy to apply and verify.

2. Knowledge base TBox/ABox creation is a form of declarative programming which removes the complexity of element interdependence.
  3. No impact from programming side effects (order of evaluation, history) and does not require knowledge about the context and its possible consequences.
- C. Consistency checking (satisfiability) is defined as finding an interpretation ( $C^I \subseteq D^I$ ) for an axiom ( $C \subseteq D$ ). The KB is satisfiable if all axioms are satisfied by the same interpretation. If  $\{(C \subseteq D)\} \cup \text{KB}$  results in  $a \sqcap \neg a$  (defined as a clash) where  $a$  is some KB concept then  $(C \subseteq D)$  is not satisfiable w.r.t. KB. Consistency checking allows the detection and traceability of inconsistency in the domain knowledge (TBox). The TBox Coherence service lists all unsatisfiable concepts in an ontology.
- D. Inferencing (subsumption) can be reduced to satisfiability by adding a complement of an axiom A to the KB. If this results in a clash then  $\text{KB} \models A$  and A can be made explicit. Inference uses tableaux and completion rules to make implicit KB facts explicit through object identities (equivalence axioms, keys), object relationships (domain and range definitions) and topological relationships with rule (SWRL) support. The Subsumption Hierarchy service computes a taxonomy from the subsumed hierarchy of concepts.
- E. ABox Satisfiability is similar to TBox Satisfiability but instead is processed on the set of instances (assertion axioms). It could detect offending assertions (inconsistencies in the data) if the appropriate closure axioms are available. The Instance Checking service verifies if a certain concept subsumes a certain instance and the ABox

Realization service computes the most-specific concept names for all individuals.

Each instance is classified to its most specific subclass based on the available facts.

F. No Closed World Assumption (CWA), or in other words Open World Assumption (OWA), and no Unique Name Assumption (UNA) allows for reasoning without an assumption of a complete data set and about possibly equivalent instances respectively. These are features inherited from Description Logic.

OWA denotes that available knowledge is always assumed as incomplete unless explicitly stated and the lack of knowledge of a fact does not immediately imply knowledge of the negation of a fact. Consider the following example ABox axioms:  $\{a:bridge, b:lane, (b,a):part\_of\}$ . This does not mean that bridge has only one lane. Another lane might exist that we don't know about.

Our use case deals with GIS data that could be both incomplete and ambiguous. The flexibility of having OWA and no UNA gives advantages in our application domain such as being able to add and introduce new knowledge as needed and assuming by default that two instances even having different names could actually represent the same entity (if Same Individual check service is available). Particularly, data property restrictions already provide text value matching under  $SROIQ^{(D)}$ . Same Individual service could therefore be used where properties such as *hasRefPoint* contain the same Well-Known Text (WKT) value string to automatically merge the instance definitions together. The knowledge base will remain consistent with new inferred information as new facts are added to the knowledge base. There are methods to

enforce UNA (usually invoked for performance reasons). It is also possible to explicitly define that an instance is equivalent to another.

G. Explanation services trace the completion rules of axioms to show the minimum set of facts that result in a certain inference or clash. They allow understanding what resulted in the clash and, as a result, fix the offending facts. We use explanation services to determine the certainty of an inferred fact. We also use them to generate reports that help the user identify an inconsistency and fix the problem accordingly.

H. Axiom annotations on TBox data property concepts (associating specific data extractors) and on ABox facts (certainty attribution) allow the separation between knowledge and extensions; they are not processed by KB services. However, they allow to maintain appropriate associations with related elements of the system and can be used for extending functionality such as resulting certainty calculation.

I. Rule and query languages e.g. nRQL and SWRL modify the ABox with further expressivity without impacting other definitions in the system. nRQL querying through a compatible reasoner also enables the calculation of the subsumption hierarchy for only the elements involved in the query (a subset of the KB) which could improve performance in realizing the KB. SPARQL and GeoSPARQL allow the querying of the underlying RDF graph of the knowledge.

J. Insertion and Retraction services allow axiom insertion and retraction dynamically after KB creation. Internally, retraction is often managed as destroying the KB and recreating it as  $\text{KB} \setminus \{A\}$  where A is an axiom to be retracted (by omitting the retracted axiom). We thought of using this mechanism as part of an earlier attempt

(section 4.2.3) in order to resolve satisfiability issues as they are introduced, however, it proved to make the process more complex and we finally adopted a different method.

### **8.2.2 Ambiguity**

We categorize ambiguity into two different types:

1. Two elements in the input source data ( $S_{GIS}$ ) define two entities redundantly
2. A knowledge base entity is associated two properties that are
  - i. complementary (can coexist)
  - ii. contradicting (cannot coexist)

If proper equivalence and closure axioms are defined in the TBox (complete ontologies), occurrence of 1 is identified as a redefinition of the same actual entity using the Same Individual service. For example, Keys is an addition to OWL DL provided by the  $SROIQ^{(D)}$  expressivity which allows automatic identification of same individuals (KB instances). In our implementation, we used Keys with WKT definitions and real world entity identifiers to ensure entity uniqueness. Moreover, we can specify different or same individual assertions if the fact is derived by ABox manipulation, facts querying, insertions and retractions. After same individuals are identified and their properties automatically merged, if contradicting facts exist, the knowledge base becomes inconsistent.

Occurrence of 2 denotes a certain fact (assertion) having two meanings (TBox definitions). These meanings can be complementary such as one being a specialization of

the other or using both as a satisfiable class conjunction to define an instance property. This does not make the knowledge base inconsistent and is permitted under the expressivity used provided the ontologies are complete. In the other case, if the meanings are contradicting, realizing the knowledge base with proper disjointness and closure axioms and well-defined concepts (complete ontologies) would make the inconsistencies explicit. To address this issue, explanation services are used to identify the offending facts and the reasoning behind each inconsistency. The user would then have to correct the input data. For example, an instance that is defined as both a tunnel and a bridge in the data is inconsistent in a KB that contains *disjoint(tunnel, bridge)*. In legacy processes, the user would commonly generate a representation of the ROI and inspect any inconsistencies, often visually (which might not be obvious), and would have to go back to correct the input data sets and repeat until an acceptable representation is achieved. With our process, the inference engine is able to pinpoint the offending facts with an explanation before a representation is generated. The semantic information could then be used to correct the source data.

Moreover, the main reason for our addition of uncertainty attribution to facts in the system is to allow ambiguity (complementary or contradicting) to be expressed as part of the facts. This also addresses GIS data generalization. Strongest certainty would propagate to inferred facts based on relevant facts in the explanations of the inference as described in Chapter 6. Several facts could also co-exist with related certainties and users can use semantics and context to form a decision that validates/invalidates a fact with respect to  $S_{GIS}$  and change the knowledge base using insertion/retraction services.

### **8.2.3 Incompleteness in Data Set**

Considering complete ontologies defined as per the definition in section 4.1.3, we categorize incompleteness into two different types:

1. Input data sources ( $S_{GIS}$ ) do not contain all the data compared to real world (3D Real)
2. An entity does not contain all the required information explicitly (based on  $S_S$ )

Occurrence of 1 happens when it is impossible to find all the data for a certain entity in  $S_{GIS}$ . This already violates our first assumption. However, this could be addressed by user queries or defaults as part of data extractors where the most specific known subclass can be used. Occurrence of 2 is resolved using our Data Extractors framework. Given a knowledge base instance with an identified subclass, all the properties are assumed to be assigned values after data extraction using the framework.

### **8.2.4 Limitations**

There are some disadvantages to using Semantic Web. These include:

- A. Relational and probabilistic reasoning and complex datatype value analysis with Semantic Web are yet to mature.

$ALCRP^{(D)}$  expressivity allows for ternary property relations with implicit chained properties that are satisfiability verifiable. This is a good advantage for GIS topological processing e.g.  $\{ \text{inside}(x,y) \cap \text{touches}(x,z) \Rightarrow \text{connection}(y,z) \}$  in GIS interpretations. We used SWRL rules to easily achieve these kinds of deductions, but, in this case, TBox inconsistency cannot be identified. Inconsistencies will only be identified at the assertions level (ABox) with reasoners supporting  $SROIQ^{(D)}$ . The TBox

inconsistency would only be captured if the reasoner supports  $ALCRP^{(D)}$ . Another way to define this is with chain axioms using inverse properties. However, this adds complexities in the defined knowledge (inverse properties are required to be defined). Our choice to use rules to reason about entities involved in multiple properties is purely to have complete separation between TBoxes and ABoxes. Nominals could be used in TBox axioms in  $SROIQ^{(D)}$  to replace such rules if the set of instances involved is static between different input data sets. The involved ABox instances are required to be defined along with the TBox axiom that uses them.

Algebraic reasoning does not address the problem of handling complex datatype values in the knowledge base. State of the art available standards concentrate on making simple calculations as part of the datatype property definitions and queries, which we have made use of. However, complex value handling extensions and knowledge interpretation is not very well addressed.

We work around the issues of probabilities, datatype value reasoning and topological processing with our Data Extractors framework which derive explicit facts that can be used as part of conventional  $SROIQ^{(D)}$ . The framework derives information by using procedures that generalize the method of retrieving a certain property value or relationship along with its certainty.

## B. Complexity and learning curve.

We cannot ignore the complexity of using Knowledge Bases and DL inferencing. Although these provide extremely powerful constructs to reason about facts, we recognize that our process increases automation at the cost of complexity of defining

reusable knowledge. Reusable knowledge requires that it is fully generic and contained in the TBox and independent of the actual instances (ABox). We, therefore, require the power of the expressivity and services provided for  $SROIQ^{(D)}$  and thus we depend on the complexity of its ABox Realization algorithms and justifications computation services. Although highly automated, these services currently are far from being capable of real-time processing.

Although describing concepts and knowledge might seem easier than defining procedural rules, the former requires a good understanding of logic (such as description logic) and constructs to be able to define expressive, useful and reusable knowledge. Many tools exist to assist users to do this, however they are still very technical with little support to general audience. GIS experts do not necessarily have the background skills and knowledge required to define powerful ontologies. Experts in Semantic Web are required.

### **8.3 Future Work**

Currently, the process assumes that the facts extraction step does all the additions to the knowledge base prior to  $v\_mapping$ . However, users generally define properties on a per-need basis. Further research is required to reduce the search space for data needed for the representation rather than attempt to collect all available facts needed by the knowledge base. This could be achieved, for example, by assessing extractors of required properties only taking into account those that could define specializations. Moreover, it

would be more efficient to reduce clutter in the KB. For example, although we process relationships in *r\_mapping* on all entities in the ROI, experiments are required for assessing if it would be efficient and would give correct results to only consider immediate vicinity of an instance (rather than relationships with all other).

As expressed earlier, *SROIQ<sup>D</sup>* is very expressive for our needs but it suffers in available algorithm performance. It would be crucial to have an efficient ABox Realization implementation for this methodology to be used effectively in a dynamic environment. Using nRQL [Haarslev et al., 2004] allows speeding up processes by triggering inference services only when needed. To our knowledge, this is one of the unique features available in RacerPro, a professional scale semantic web reasoner, and avoids the need for ABox Realization on all instances. Moreover, easier and generic tools are required for more generic users.

Extension capability is required in SPARQL queries and knowledge base data reasoning. For example, there is currently no way to define data reasoning procedures to extract facts from complex data values. DE-9IM efficiently serializes the intersection matrix between two entities as a string value, however, there is no way to customize how strings are used in the reasoning process natively. It is currently required to insert as many assertions as such a string implicitly encodes.

Actual use case comparison studies by GIS experts are needed to compare our methodology with available state-of-the-art methodologies addressing GIS to 3D conversion. The intention is to gather temporal and difficulty metrics that will show the practicability of the techniques used in our methodology.

Although our methodology can be used for defining an ROI semantically, it would be interesting to explore potential of using a semantic knowledge base for spatial querying. It wouldn't be very efficient to use queries such as line of sight unless a 2D/3D representation is generated and then used. However, it could directly serve queries such as occlusion, smart path navigation, and processes that would use object categorizations and identified properties and facts. This helps define more automated agents.

We would like to investigate and further elaborate the specific case of generating various entities in the transportation domain in further detail, for example, bridge spans including more specialized types, combinations of spans such as one span with multiple definitions, and hidden information deductions such as support structure details such as using other objects available e.g. underpass layout to determine approximate spacing of supports. It would be interesting to extend this work beyond transportation to other domains.

We currently calculate the uncertainty of a concept class entailment using the variable set, which is a subset of the explanation of the entailment. We do not think that maintaining the uncertainties and their results is feasible within the knowledge base and its realization due to the dependence on the data extractors used. However, further research is required to assess whether, in some cases, the certainties can be maintained based on the available knowledge and independent of the extractors.

Lastly, a declarative specification method for defining properties and plug-in extractors would enable this approach to be used widely.

## **8.4 Final Remarks**

The use of knowledge for generating representations and specifically for 3D rendering purposes has been studied but not very well addressed in the past due to the vast collection of information that exists in a scene. We showed a new process that enables further automation and assistance to users in the generation of knowledge bases from GIS data sets and corresponding representations. We believe that through a categorization of the domain knowledge and a modularization of the methods used to retrieve values for the properties required to create a certain 3D digital model with uncertainty support, we can address the problem of heterogeneous data fusion and automating representation better.

Our process extracts facts including relationships and properties using the Data Extractors framework and includes automatic inference and calculation of associated uncertainties. The system will infer implicit information while interrogating the user for the remaining to form a definition that models the object with higher detail when compared to generating a representation based on an entity definition in shapefiles.

A central location for reasoning is defined which treats data dependencies semantically rather than based on format definition. This allows to semantically synthesis further details for required representations using emerging and rigid Semantic Web constructs that are more easily verified and reused. In order to test and evaluate this process in terms of feasibility and efficiency, an application framework was developed and presented in Chapter 5. Referring to the differences between Figure 4 and Figure 13, our process supports legacy terrain modeling and processing through the mapping of

concepts from the required format(s). It is able to take source data in a specified format defined by the SD ontology as input to the system. Our process incorporates the semantic reasoning as part of terrain modeling and processing. This simplifies the process with less manual iteration and user verification to attain the same results as in legacy processes. It also maintains the flexibility of manual user input and control. Our use of collective reasoning reduces considerably the high-involvement of the domain expert in time and effort when compared to other known processes.

This same methodology can theoretically be used by any system that requires converting between data formats and semantics and between semantic definitions and needed output representations and demonstrated by a simple example in section 7.7. Data analysis happens on the semantic level and makes all implicit information explicit for the output representation.

Data is added as facts to a knowledge base, independent of the type of the GIS data source, and can be shared among several applications, making any system requiring the use of a single coherent and semantically sound knowledge base of an ROI to create a certain representation, able to use our methodology.

## **Bibliography**

ArcGIS (2014). *A platform for designing and managing solutions through the application of geographic knowledge.* ESRI. Available from <http://www.esri.com/software/arcgis>. Internet. Retrieved 26 January 2014.

Arpinar I.B., Sheth A., Ramakrishnan C., Usery E.L., Azami M. & Kwan M. (2006). *Geospatial Ontology Development and Semantic Analytics.* In proceedings of Transactions in GIS 10, 551–75.

Baader, F., Horrocks, I., & Sattler, U. (2003). *Description Logics as Ontology Languages for the Semantic Web.* In D. Hutter and W. Stephan, editors, Festschrift in honor of Jorg Siekmann, Lecture Notes in Artificial Intelligence. Springer, 2003

Bechhofer S., van Harmelen F., Hendler J., Horrocks I., McGuinness D.L., Patel-Schneider P.E., & Stein L.A. (2004). *OWL Web Ontology Language Reference.* Available from <http://www.w3.org/TR/2004/REC-owl-ref-20040210>. Internet. Retrieved on 26 January 2014.

Berners-Lee, T., Hendler, J. & O. Lassila (2001). *The SemanticWeb.* Scientific American. 284 (5), 2001, pp. 34–43.

Bitters, B. (Spring 2005). *A Geographical Ontology of Objects in the Visible Domain*. Ph. D. Dissertation submitted to Department of Geography. University of West Florida, Florida.

Bitters, B. (July 2006). *Geospatial Perpetual Motion: Finding Hidden Information Within Existing Geospatial Databases*. In proceedings of IMAGE 2006 Conference. The IMAGE Society. Scottsdale, Arizona.

Bitters, B. (July 2007). *Very High-Detail Depictions of Forests in Virtual Environments*. In proceedings of IMAGE 2007 Conference. The IMAGE Society. Scottsdale, Arizona.

Bitters, B. (December 2008). *Spatial Relationship Networks: Network Theory Applied to High Detail Virtual Environments*. Proceedings of I/ITSEC 2008, Orlando, Florida.

Blaschke, T. and Drăguț, L. (2003). *Integration of GIS and object-based image analysis to model and visualize landscapes*. ISPRS workshop “Challenges in Geospatial Analysis, Integration and Visualization II”, September 8- 9, 2003, Stuttgart, Germany, 18-23.

Blundell, J.S, & Opitz D.W. (2006) *Object recognition and feature extraction from imagery: The feature analyst approach*. In proceedings of OBIA conference 2006.

Bobillo, F., Delgado, M., Gómez-Romero, J. (2012). *DeLorean: A reasoner for fuzzy OWL 2*. Expert Systems with Applications. Volume 39, Issue 1, January 2012, pp. 258-272, ISSN 0957-4174.

Bobillo, F., & Straccia, U. (2011). *Fuzzy Ontology Representation using OWL 2*. In International Journal of Approximate Reasoning, 52(7):1073-1094, 2011.

Brockway, D. (2002). *Architecture for Managing Vector, Raster, and 3D Geometry in GIS*. ESRI International User Conference Paper 1068. MultiGen-Paradigm, Inc.

Brodaric, B., & Hastings, J. (2002). *An Object Model for Geologic Map Information*. Proceedings of the 10<sup>th</sup> International Symposium on Spatial Data Handling (SDH 2002). Ottawa, Canada.

Buchholz, H., Dollner, J., Ross, L. & Kleinschmit, B. (2006). *Automated Construction of Urban Terrain Models*. Proceedings of the 12th International Symposium on Spatial Data Handling (SDH 2006), 547-562.

CDB (October 2008). *Common Database (CDB) Specification for USSOCOM*. Prepared for the U. S. Army Program Executive Office for Simulation, Training, and Instrumentation (PEO STRI) and Product Manager for SOF Training Systems (PM STS). CAE USA Inc., Tampa, Florida.

Clementini, E., Sharma, J., & Egenhofer, M. J. (1994). *Modeling topological spatial relations: Strategies for query processing*. Computers & Graphics 18 (6): 815–822. doi:10.1016/0097-8493(94)90007-8.

Daconta, M. C., Obrst, L. J., & Smith, K. T., (2003). *The Semantic Web: A Guide to the Future of XML*. Web Services and Knowledge Management. Wiley Publishing, Inc., Indianapolis.

DCCatalog. (2014). In *District of Columbia Geographic Information Systems*. Government of District of Columbia official website, Washington. Available from <http://dcatlas.dcgis.dc.gov/catalog/>. Internet. Retrieved on 26 January 2014.

De Kok, R., Schneider T., Baatz M., Ammer U. (2006) *Object based image analysis for high resolution data for alpine forest area*. In proceedings of OBIA conference 2006.

Delenne, C., Rabatel, G., Agurto, V. & Deshayes, M. (2006). *Vine plot detection in aerial images using Fourier analysis*. In proceedings of the ISPRS OBIA 2006 conference Vol. XXXVI – 4/C42. ISSN 1682-1777. 4-5 July 2006. Salzburg University, Austria.

Deng, F., Zhang Z., & Zhang J. (2006) *Construction 3D Urban Model from LIDAR and Image Sequence*. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. 34, Part XXX.

Dewberry & Davis (2002). *GIS/CAD Data Dictionary v.4.1.1*. Office of the Chief Technology Officer. Department of Public Works, District of Columbia, Washington.

DFAD (1994). *Military Specification Digital Feature Analysis Data (DFAD) Level 1*. DMA MIL-SPEC MIL-D-89005, 89006, and 89017, (draft). National Imagery and Mapping Agency, United States Geological Survey (USGS). 5 August 1994.

Ding, Z., & Peng, Y. (2004). *A Probabilistic Extension to Ontology Language OWL*. In Proceedings of the 37th Hawaii International Conference on System Sciences IEEE 2004. University of Maryland Baltimore County.

DTED (May 2000). *Performance Specification – Digital Terrain Elevation Data (DTED)*. Departments and Agencies of the Department of Defense, US. MIL-PRF-89020B. National Geospatial Agency. October 2007

DVC (2014). In Diamond Visionics Corporation. Diamond Visionics, LLC. Available from <http://www.diamondvisionics.com/>. Retrieved 26 January 2014.

Egenhofer, M.J., & Franzosa, R.D. (1991). *Point-set topological spatial relations*. International Journal of Geographical Information Systems, vol.5, no.2, 161-174.

Eid, P., & Mudur, S.P. (February 2009). *Use of Semantic Web Technology for Adding 3D Detail to GIS Landscape Data*. In proceedings of the Canadian Conference on Computer Science and Software Engineering 2009 (C3S2E'09), 205-215. Published by the Association of Computing Machinery (ACM). Montreal, Canada.

Eid, P., & Mudur, S.P. (December 2009). *Automating Extraction of 3D Detail from GIS Data using Semantic Web Technology*. In proceedings of Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2009. Published by the National Training and Simulation Association (NTSA). Orlando, Florida.

Eid, P., & Mudur, S.P. (May 2010). *Synthesizing high fidelity 3D landscapes from GIS data*. Proc. of 1<sup>st</sup> International Conference on Computing for Geospatial Research & Application 2010 (COM.Geo'10). ACM. Article 15. Washington, DC. May 2010.

Eid, P., & Mudur, S. (2013). *Generating Bridge Structure Model Details by Fusing GIS Source Data Using Semantic Web Technology*. Computing for Geospatial

Research and Application (COM. Geo), 2013 Fourth International Conference on. IEEE, 2013.

ERDAS (2014). In Intergraph | Products. Available from <http://geospatial.intergraph.com/products/ERDAS-IMAGINE/Details.aspx>. Internet. Retrieved 26 January 2014.

ESRI (1998). *ESRI Shapefile Technical Description – An ESRI White Paper*. Environmental Systems and Research Institute, Inc. ESRI, U.S.

Faddoul, J. (2011). *Reasoning Algebraically With Description Logics*. Ph. D. Dissertation submitted to Department of Computer Science and Software Engineering. Concordia University, Montreal.

Fillmore, R. (2006). *The MGCP is making big strides towards getting global high resolution data common across the board*. Military Geospatial Technology, 23 March 2006, V 4:1.

Fonseca, F.T., Egenhofer, M.J., Agouris, P., & Câmara, G. (2002). *Using Ontologies for Integrated Geographic Information Systems*. In proceedings of Transactions in GIS 6(3), 231-257. Blackwell Publishers Inc.

Gahegan, M. & Flack, J., (2002). *The Integration of Scene Understanding within a Geographic Information System: A Prototype Approach for Agricultural Applications*. Transactions in GIS 3(1) pp. 31-49.

GeoGenesis (2014). By IAVO. Available from <http://www.geogenesis.net/>. Internet. Retrieved 26 January 2014.

GeoSPARQL (2012). *OGC GeoSPARQL - A Geographic Query Language for RDF Data*. OGC® Implementation Standard. 10 September 2012. Version 1.0. Ref# OGC 11-052r4. Available at <http://www.opengis.net/doc/IS/geosparql/1.0>

GeoTools (2014). GeoTools Open Source Java API. OSGeo. Available from <http://www.geotools.org/>. Internet. Retrieved 26 January 2014.

Gerla, G. (Oct 1994). *Inferences in probability logic*. Artificial Intelligence. Vol 70, Issues 1–2, pp. 33-52. ISSN 0004-3702

Glimm, B. & Kazakov, Y. (2008). Role Conjunctions in Expressive Description Logics. In Logic for Programming, Artificial Intelligence, and Reasoning (pp. 391-405). Springer Berlin Heidelberg.

Goodwin, J. (2005). *What Have Ontologies Ever Done For Us – Potential Applications at a National Mapping Agency*. OWL: Experiences and Directions workshop. Galway, Ireland.

Gruen, A., & Wang, X. (August 1999). *CyberCity Modeler, a tool for interactive 3-D city model generation*. Proceedings of the “Germany Photogrammetry Week”. Stuttgart, Germany.

Haarslev, V., Möller, R., & Wessel, M. (2004) *Querying the Semantic Web with RACER + nRQL*. Proceedings of the KI-2004 International Workshop on Applications of Description Logics (ADL'04). Ulm, Germany.

Haarslev, V. (2014). *Description Logics: A Logical Foundation of the Semantic Web and its Applications*. Dept. of Computer Science, Concordia University, Montreal, Canada. Available from <http://users.encs.concordia.ca/~haarslev/publications/dl-semweb.pdf>. Internet. Retrieved on 26 January 2014.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE Transactions on Systems Science and Cybernetics SSC4 4(2): 100–107.

Hitzler, P., Krotzsch, M., & Rudolph, S. (2011). *Foundations of Semantic Web Technologies*. Chapman & Hall /CRC Press. ISBN 978-1-4200-9050-5.

Horridge, M. (2011). *Justification Based Explanation In Ontologies*. Ph. D. Dissertation submitted to School of Computer Science. University of Manchester, UK.

Horridge, M., Parsia, B., & Sattler, U. (2008). *Laconic and Precise Justifications in OWL*. The Semantic Web-ISWC 2008, pp. 323-338, 2008.

Horridge, M., Parsia, B., & Sattler, U. (2009). *Explaining Inconsistencies in OWL Ontologies*. SUM 2009, LNAI 5785, pp. 124–137, 2009.

Horrocks, I., Kutz, O., & Sattler, U. (2006). *The Even More Irresistible SROIQ*. In Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR2006), AAAI Press, (June 2006), pp. 57-67.

Hudelot, C., Atif, J., & Bloch, I.. (2008). *Fuzzy spatial relation ontology for image interpretation*. Fuzzy Sets and Systems. Volume 159, Issue 15. 1 August 2008, pp. 1929-1951. ISSN 0165-0114.

Hummel, B., Thiemann, W., & Lulcheva, I. (2008) *Scene Understanding of Urban Road Intersections with Description Logic*. Proceedings of Dagstuhl Seminar 08091, paper 1616. Schloss Dagstuhl, Saarland University, Deutschland.

ISAR (2007). *Inverse Synthetic Aperture Radar (ISAR) Imagery Feature Extraction and Database*. Navy SBIR 2007.1 - Topic N07-044. Solicitation closed on 10 January 2007.

Jain R. C., & Jain A.K. (1990) *Analysis and Interpretation of Range Images*. Springer-Verlag. 1990. ISBN 0-387-97200-5.

Jenner, B., Toran, J. (1995). *Computing functions with parallel queries to NP*. Theoretical Computer Science. Vol. 141. Pp. 175–193.

Kalogerakis, E., Christodoulakis, S., and Moumoutzis, N. (2006). *Coupling Ontologies with Graphics Content for Knowledge Driven Visualization*. In Proc. of IEEE Virtual Reality 2006, Alexandria, VA, USA.

Kalyanpur, A., Parsia, B., Horridge, M., & Sirin, E. (2007). *Finding all justifications of OWL DL entailments*. In The Semantic Web (pp. 267-280). Springer Berlin Heidelberg. 2007.

Kim, K.S., Lee, J.S., Cho, S.Y., Lee, S.H., & Chang, E. (2013). *GeoSpatial Semantic Web Conversion tool for topographic maps*. National Geographic Information Institute. ISO/TC 211 Standards in Action Workshop. 29 May 2013.

Klinov, P., & Parsia, B. (2013). *Pronto: A practical probabilistic description logic reasoner*. In Uncertainty Reasoning for the Semantic Web II (pp. 59-79). Springer Berlin Heidelberg.

Kraak, M.J. & Ormeling, F.J. (1996 & 2011) *Cartography: visualization of spatial data*. New York, London, Pearson Education, 1996 & 2011. ISBN: 978-1-60918-193-2.

Lang, S., Albrecht, F. & Blaschke, T. (2006). *OBIA Tutorial – Introduction to Object-based Image Analysis*. Version 1.0, Centre of Geoinformatics, Paris-Lodron University, Salzburg. 2006.

Lang, S., & Blaschke T. (2006) *Bridging remote sensing and GIS – What are the main supportive pillars?* In proceedings of 1st international conference on Object-based image analysis (OBIA 2006).

Laskey, K. B. (2009). *Axiomatic First-Order Propability*. In Proceedings of the Fifth International Workshop on Uncertainty Reasoning for the Semantic Web (URSW

2009), collocated with the 8th International Semantic Web Conference (ISWC-2009). Washington DC, USA. October 26, 2009.

Li, L., Liu, Q., Tao, Y., Zhang, L., Zhou, J., & Yu, Y. (2006). *Providing an Uncertainty Reasoning Service for Semantic Web Application*. In Proceedings of the 8th Asia-Pacific Web conference on Frontiers of WWW Research and Development (APWeb'06). Springer-Verlag Berlin, Heidelberg.

Liebig, T., Luther, M., Noppens, O., Rodriguez, M., Calvanese, D., Wessel, M., Möller, R., Horridge, M., Bechhofer, S., & Tsarkov, D. (2008). *OWLlink: DIG for OWL 2*. In 5th OWL Experienced and Directions Workshop, 2008.

Lin, K., & Ludäscher, B. (2003). *A system for semantic integration of geologic maps via ontologies*. Semantic Web Technologies for Searching and Retrieving Scientific Data (SCISW), Sanibel Island, Florida.

Lukasiewicz, T. (1998). *Probabilistic Logic Programming*. In European Conference on Artificial Intelligence, 1998, pp. 388-392.

Lukasiewicz, T. (2008). *Expressive probabilistic description logics*. Artificial Intelligence, 172(6), 852-883.

Lukasiewicz, T., & Straccia, U. (2008). *Managing uncertainty and vagueness in description logics for the Semantic Web*. Web Semantics: Science, Services and Agents on the World Wide Web, vol. 6, no. 4.

MacEachren, A. M., Gahegan, M., Pike, W., Brewer, I., Cai, G., Lengerich, E., and Hardisty, F. (2004). *Geovisualization for knowledge construction and decision-support*. IEEE Computer Graphics & Applications 24 (1):13-17.

MapWindow GIS (2014). MapWindow GIS Open Source Project. Available from <http://www.mapwindow.org/>. Internet. Retrieved 26 January 2014.

Marszalek, M., & Schmid, C. (2007). *Semantic Hierarchies for Visual Object Recognition*. IEEE Conference on Computer Vision & Pattern Recognition (LEAR 2007).

McCarthy, P. (2005). *Search RDF Data with SPARQL*. IBM DeveloperWorks white paper, 2007. Available from <http://www.ibm.com/developerworks/xml/library/j-sparql/>. Internet. Retrieved 26 January 2014.

McKeown, D., Giuliani, J., de la Cruz, J., & Sotomayor, T. (July 2007). *Automating the Generation of Urban Details*. In proceedings of IMAGE 2007 Conference. The IMAGE Society. Scottsdale, Arizona.

Mizen, H., Dolbear, C., & Hart, G. (29 November 2005). *Ontology Ontogeny: understanding how an Ontology is created and developed*. In First International Conference on GeoSpatial Semantics (GeoS 2005), 15-29. Mexico City, Mexico.

NGA. (2014). In *National Geospatial-Intelligence Agency*. Department of Defense, United States of America. Available from <http://www.nga.mil/>. Internet. Retrieved 01 February 2014.

OGC (2014). <http://www.opengeospatial.org>. Open Geospatial Consortium. Internet. Retrieved 26 January 2014.

OpenFlight (2007). *OpenFlight Scene Description Database Specification, version 16.4*. November, 2007. Multigen-Paradigm website. Available from <http://www.presagis.com/files/standards/OpenFlight16.3.pdf>. Internet. Retrieved on 01 February 2014.

Opitz, D.W., Rao R., & Blundell J.S. (2006) *Automated 3D Feature Extraction from Terrestrial and Airborne Lidar*. In proceedings of OBIA conference 2006.

OSUK (2014). Ordnance Survey, Brittain's National Mapping Agency. Available from <http://data.ordnancesurvey.co.uk/ontology/>. Internet. Retrieved 26 January 2014.

OWL API (2014). The OWL API 3. Co-ODE project. Based on OWL 2. Available from <http://owlapi.sourceforge.net/>. Internet. Retrieved 26 January 2014.

OWL Datatypes (2012). *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*. W3C Recommendation 5 April 2012. Available from <http://www.w3.org/TR/2012/REC-xmldatatypes-20120405/>. Internet. Retrieved 26 January 2014.

OWL2 (2012). *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)*. The World Wide Web Consortium (W3C) Recommendation 11 Dec 2012. Available from <http://www.w3.org/TR/owl2-syntax>.

Pellet (2014). The Pellet OWL 2 Reasoner, Clark&Parsia. Available from <http://clarkparsia.com/pellet/>. Internet. Retrieved 26 January 2014.

Pendris, J. (2006). *Real-Time Terrain Modification Using Cultural and Feature Data*. In proceedings of IMAGE 2006 conference. The IMAGE Society. Scottsdale, Arizona.

Poole, D., Smyth, C., & Sharma, R. (2009). *Ontology Design for Scientific Theories That Make Probabilistic Predictions*. IEEE Intelligent Systems, vol. 24, no. 1, pp. 27-36, January/February, 2009

Presagis (2014). In Products | Content Creation | Presagis. Presagis Canada Inc. Available from <http://www.presagis.com>. Internet. Retrieved 01 February 2014.

Protégé (2014). The Protégé Platform. Available from <http://protege.stanford.edu/>. Internet. Retrieved 26 January 2014.

Selman, A. (1994). *A taxonomy of complexity classes of functions*. Journal of Computer and System Sciences. Vol. 48 Issue 2. Pp. 357–381.

Sirakov, N. M. (2006). *Digital Image Databases and 3D Visualization – Applications to Science and Industry*. Dept. of Mathematics and CS, Texas A&M University Commerce. October 2007.

SocetSet (2014). In *BAE Systems Geospatial eXploitation Products*. British Aerospace Ltd. Available from <http://www.socetset.com/>. Internet. Retrieved on 26 January 2014.

Stanzione, T. (2006). *Using ArcGIS to Create Semantic Information for Modeling and Simulation*. In Proc. of the ESRI International User Conference 2006. San Diego, California.

Stelle, D. (2003). *Processing LIDAR for 3D Urban Visualization*. In proceedings of IMAGE 2003 conference. The IMAGE Society. Scottsdale, Arizona.

Town, C. (2004). *Ontology-driven Bayesian Networks for Dynamic Scene Understanding*. In proceedings of Computer Vision and Pattern Recognition Workshop 2004 (CVPRW'04). IEEE Computer Society Conference. Washington, DC. 2004

Vanegas, C.A., Aliaga, D.G., Benes, B., & Waddell, P. (2009) *Visualization of Simulated Urban Spaces: Inferring Parameterized Generation of Streets, Parcels, and Aerial Imagery*. IEEE Transactions on Visualization and Computer Graphics, Vol 15, No 3, May/June 2009.

Van der Sterren, W. (2001). *Terrain Reasoning for 3D Action Games*. In Proceedings of Games Developer's conference 2001 (GDC2001). San Jose, CA

VectorData. (2014). In *Virtual Terrain Project*. Virtual Terrain Project Organization on-line. Available from <http://www.vterrain.org/Culture/vector.html>. Internet. Retrieved 01 February 2014.

VMAP (2014). In Vector Map Level 0 (VMap). The National Geospatial-Intelligence Agency. Available from <http://earth-info.nga.mil/publications/vmap0.html>. Internet. Retrieved 01 February 2014.

Wiegand, N., & Garcia, C. (17 June 2007). *A Task-Based Ontology Approach to Automate Geospatial Data Retrieval*. Transactions in GIS Vol. 11. Blackwell Publishing Ltd. 355–376

Wikipedia. (2014). In *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation Inc. Encyclopedia on-line. Available from <http://en.wikipedia.org/wiki/>. Internet. Retrieved 01 February 2014.

Woodward, T., Upton, G., & Simons, R. (2001). *Developing a Framework for IG-Independent PC-Based Dynamic Terrain*. In proceedings of I/ITSEC 2001. Orlando, Florida.

Yin, X., Wonka, P., & Razdan, A. (2009). *Generating 3d building models from architectural drawings: A survey*. Computer Graphics and Applications, IEEE, 29(1), 20-30.

Zadeh, L.A. (1965). *Fuzzy sets*. Information and Control 8 (3): 338–353.

Zhang, B., Miller, S., Walker, S., & Devenecia, K. (11 May 2007). *Next Generation Automatic Terrain Extraction Using Microsoft Ultracam Imagery*. In proceedings of ASPRS 2007. Tampa, Florida.

Zhang, R. & Zhang, Z. (2004). *Hidden Semantic Concept Discovery in Region Based Image Retrieval*. In Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'04).

## ***List of Abbreviations***

- $\Delta^I$ : domain of interpretation (transportation domain)
- 3D Real: the real world 3D Region of Interest
- 3D Schema: Schema definition of possible representation procedures of 3D Real
- ABox: Assertional Axioms Box
- CBIR: Content-Based Image Retrieval
- CWA: Closed World Assumption
- DE-9IM: Dimensionally Extended nine-Intersection Model
- DEM: Digital Elevation Model
- DFAD: Digital Feature Attribute Data standard 1994
- DL: Description Logic
- DTED: a common DEM format developed by the NGA (MIL-PRF-89020B)
- DTM: Digital Terrain Model
- DVC: Diamond Visionics Corporation
- ERDAS: Earth Resources Data Analysis System Incorporated
- ESRI: Environmental Systems Research Institute
- GIS: Geographic Information Systems.
- ITAR: International Traffic in Arms Regulations
- JPEG2000: Joint Photographic Experts Group (JPEG) 2000 format
- KB: Knowledge Base
- LIDAR: Light Induced Detection And Ranging standard
- NGA: National Geospatial-Intelligence Agency

- OBIA: Object Based Image Analysis
- OGC: Open Geospatial Consortium
- OTW: Out-The-Window view (restricted and controlled)
- OWA: Open World Assumption
- OWL: Ontology Web Language
- RC: Representation Capabilities ontology
- RDF: Resource Description Framework
- ROI: Region of Interest
- SD: Source to Domain mapping ontology
- SPARQL: SPARQL Protocol And RDF Query Language
- TBox: Terminological Axioms Box
- TD: Transportation Domain ontology
- TIN: Triangulated Irregular Network
- UAV: Unmanned Airborne Vehicles
- UNA: Unique Name Assumption
- USGS: United States Geological Survey agency
- UTM: Universal Transverse Mercator coordinate system
- VMAP: Vector MAP (MIL-PRF-89039) standard 1999
- VOTT: Visual Objects Taxonomy and Thesaurus
- W3C: World Wide Web Consortium
- WGS84: World Geodetic System 1984
- WKT: Well-Known Text format for representing geometry definitions
- XSD: XML Schema Definition

## ***List of Publications Resulting from this Work***

1. Eid, P., & Mudur, S.P. (February 2009). *Use of Semantic Web Technology for Adding 3D Detail to GIS Landscape Data*. In proceedings of the Canadian Conference on Computer Science and Software Engineering 2009 (C3S2E'09), 205-215. Published by the Association of Computing Machinery (ACM). Montreal, Canada.
2. Eid, P., & Mudur, S.P. (December 2009). *Automating Extraction of 3D Detail from GIS Data using Semantic Web Technology*. In proceedings of Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2009. Published by the National Training and Simulation Association (NTSA). Orlando, Florida.
3. Eid, P., & Mudur, S.P. (May 2010). *Synthesizing high fidelity 3D landscapes from GIS data*. Proc. of 1<sup>st</sup> International Conference on Computing for Geospatial Research & Application 2010 (COM.Geo'10). ACM. Article 15. Washington, DC. May 2010.
4. Eid, P., & Mudur, S. (July 2013). *Generating Bridge Structure Model Details by Fusing GIS Source Data Using Semantic Web Technology*. Computing for Geospatial Research and Application (COM. Geo), 2013 Fourth International Conference on. IEEE, 2013.

## ***Appendix A***

### ***Property Listing and Extractor Associations***

In order to generate 3D model representations with details comparable to entities in 3D Real, we studied modeling capabilities and evaluated the sets of properties required for flexibility and customization (defining the 3D Schema). As a result, we created a list of 3D models representing transportation features classes especially *roads*, *bridges* and their elements with their associated properties. Different values for the properties allow the procedural generation of details representing the entity.

Knowledge available in the KB (mapped as elements from the GIS source data) could contain several instances defining a complex real world entity. For example, an *overpass* entity can be defined by several instances in the knowledge base each of which defines a span (a simple structure) in the *overpass*. If a point is recorded as part of the linear record at every location along the road where:

- the road type changes,
- the road width changes,

- the curvature of the road is high ( $>45$  deg),
- and as frequently as needed for higher detail (e.g. for smoothness of the curvature)

then a series of simple representations will allow detailed representations of a complex real world entity. In this example, the spans are generated independently but, together, represent the real world *overpass*.

Moreover, since an *overpass* is a subclass of *bridge* and the representation generation process, based on the properties listed, can only represent *bridges*, all *overpass* instances are generated as bridge elements using the relevant KB properties.

All transportation scene features under our domain  $\Delta^1$  are considered to be a combination of the above two representation classes (*road* and *bridge* generation procedures) with an adaptation of the models to fit the locations and environment they must exist in. For example, an *overpass* is a *bridge over a road* or another *bridge*. A *ramp* is a *bridge* or a *road connecting two roads* together. A *tunnel* is a **subsurface road**, etc... These classes, hierarchies and relationships as well as their definitions are available as part of the TD ontology. Although no elaboration is provided, a *tunnel* could be represented, for example, by modifying the terrain geometry (semantic information in the knowledge base about the instance being a tunnel is available) and representing a *road* way within the modified geometry.

For instances that have a reference to a static predefined model as part of their properties in the KB, that model is loaded and given with the instance's properties to the representation generator. These properties include object position and orientation which will allow the representation generator to position and orient the static or procedurally

generated model to match the corresponding real world object. The following section details the property listings we collected per representation class.

## ***Property Listing***

The listing, providing a definition for each property, is organized in sections of properties. For example, the Initial Data section is required data for any basic entity representation. It includes properties defining the object class, a start edge, and an end edge. These are required for both road and bridge generation procedures. The object class is determined by the most specific known class of the instance (mapped from a linear segment) in the KB (explicit or inferred). The start and end edges are determined using the Start Edge Extractor which uses the OL and W extractors to extract two points defining the edge on opposite sides of the roadway, defined by a start and an end point. Other properties are retrieved similarly using the shown Input extractor(s) and the mapping used, if applicable.

We generalize a definition (class and properties) for roads that includes all types of roads we can think of. This definition is presented under the Road Properties section and dependent subsections: Midsection Properties, Roadway Properties, and Lane Properties; a total of 19 properties. We have identified, using Presagis Creator Studio, 20 types of bridge classes and 94 properties. These properties are categorized into sections with applicable sections for each bridge class shown in the following matrix. The property listing follows. Images are borrowed from Presagis Creator's User's Guide.

Representation Class	Initial Data	Road Properties	Spans	Angles	Deck	Supports	Tower	Suspension Profile	Arch Webbing
Road	X								
Deck Only	X		X	X					
Truss	X		X	X	X				
Deck Truss	X		X	X	X				
Beam	X		X	X	X				
Cantilever Beam	X		X	X	X				
Solid	X		X	X	X				
Covered	X		X	X	X				
Solid Arch	X		X	X	X				
Cantilever Solid Arch	X		X	X	X				
Cable Stayed	X		X	X	X	X			
Trestle	X		X	X	X				
Suspension	X		X	X	X	X			
Converging Suspension	X		X	X	X	X			
Tied Arch	X		X	X	X	X			
Converging Tied Arch	X		X	X	X	X			
Open Spandrel	X		X	X	X	X			
Cantilever Open Spandrel	X		X	X	X	X			
Converging Cantilever Open Spandrel	X		X	X	X	X			
Cantilever Through Arch	X		X	X	X	X			
Converging Cantilever Through Arch	X		X	X	X	X			

Representation Class	Solid Arch Bridge	Arch Profile	Covered Bridge	Solid Bridge	Trestle Structure	Truss Structure
Road						
Deck Only						
Truss					X	
Deck Truss					X	
Beam						
Cantilever Beam						
Solid				X		
Covered			X			
Solid Arch	X					
Cantilever Solid Arch	X					
Cable Stayed						
Trestle					X	
Suspension						
Converging Suspension						
Tied Arch		X				
Converging Tied Arch		X				
Open Spandrel		X				
Cantilever Open Spandrel		X				
Converging Cantilever Open Spandrel		X				
Cantilever Through Arch			X			
Converging Cantilever Through Arch			X			

<b>Representation Class</b>	Cable Stayed Bridge	LOD	Common Texture Settings
Road	X	X	X
Deck Only	X	X	X
Truss	X	X	X
Deck Truss	X	X	X
Beam	X	X	X
Cantilever Beam	X	X	X
Solid	X	X	X
Covered	X	X	X
Solid Arch	X	X	X
Cantilever Solid Arch	X	X	X
Cable Stayed	X	X	X
Trestle	X	X	X
Suspension	X	X	X
Converging Suspension	X	X	X
Tied Arch	X	X	X
Converging Tied Arch	X	X	X
Open Spandrel	X	X	X
Cantilever Open Spandrel	X	X	X
Converging Cantilever Open Spandrel	X	X	X
Cantilever Through Arch	X	X	X
Converging Cantilever Through Arch	X	X	X

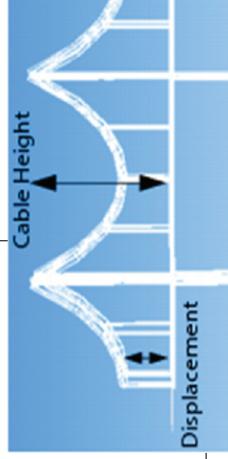
Section	Property	Input	Mapping	Definition
Initial Data				Initial Data that is required for all transportation structures: roads and bridges
Object Class	KB Inference		Determined based on available properties and facts in the knowledge base	The Object Classification: a Bridge type (from Types defined) or a Road Type
Start Edge	Start Edge Extractor using OL and W extractors		Ensures proper connectivity between connected segments	The Edge, defined by two coordinate vertices, that defines the cross section of the structure at given location. Elevation may be used to project on terrain. Ensures proper connectivity b/w connected segments.
End Edge	Start Edge Extractor at next linear vertex		same as above	
Road Properties				Properties required to graph a road transportation element, synthetically on top or instead of satellite imagery overlay. It generates a texture based on the type between the Start and End Edges.
Road Type	Object Class Property			The type of road from a classification list.
Directionality	Road_Directionality Extractor		= Separators + 1	One way, two way road, or other road. One way roads have 1 road way instance and do not have a midsection definition. For two way and other roads, each way will have a way instance with associated properties.
Location and Orientation	Start Edge, End Edge properties			The location and orientation of the generated road are defined by the locations of the Start and End Edges.
Length	Start Edge, End Edge properties			The length of the road is defined by the length of the curve defined by the direction vectors of the Start and End Edges.
Width	Start Edge, End Edge properties			The width of the road is defined by the average of the widths of the Start and End Edges.
No. of Roadways	Separators Extractor		= Separators + 1	The number of Roadways on the road segment between the Start and the End Edges.

<b>Section</b>	<b>Property</b>	<b>Input</b>	<b>Mapping</b>	<b>Definition</b>
Road Properties (cont.)	No. of Midsections	Separators Extractor	= Separators	There is a Midsection between each two roadways. This defines the number of Midsection instances for this road segment.
Sidewalk (profile, width)	Roadside_Type Extractor			There is a Sidewalk at each side of the road. This defines the type of Sidewalk, its pattern or texture and its width.
Midsection Properties				Each Midsection Road Property has a Midsection definition.
Location	Separators Extractor	location of Midsection ID	Defines the coordinate on the Start Edge where the Midsection starts.	
Midsection (profile, width)	Midsection_Type Extractor			This defines the type of Midsection, its pattern or texture and its width.
Roadway properties				Every road has 1 or more Roadway Road Property instances.
Location	Separators Extractor	location of Roadway ID	Defines the coordinate on the Start Edge where the Roadway starts.	
Directionality	Way_Directionality Extractor			Direction of the roadway instance.
No. of Lanes per way	Number_of_Lanes Extractor			This defines the number of lanes available within the roadway instance.
Lane Properties				Each Roadway has 1 or more lanes defined by the no. of lanes property.
Lane Type	Lane_Definition Extractor			The type of lane from a classification.
Ground Material Type	Lane_Definition Extractor			The ground type of the lane.
Lane Width	Number_of_Lanes Extractor			The width of the lane.

Section	Property	Input	Mapping	Definition
Spans			Considered always = 1; every OL segment will generate a bridge segment	This is available for all types of bridges. Number of towers or legs for the bridge. This specifies how many sections the bridge has; it can be any non-negative number. Some bridge types, such as Suspension, Cantilever, and Cable Stayed, require at least one span divider.
Span Dividers	1			We always consider we need divider definitions at the start and end of the span. The exception is only complex bridges such as covered solid which is created using two different definitions. In that case, the covered bridge definition has these properties disabled.
Start/End Dividers	Enabled			When set, creates a start or end divider for a bridge (not available for all bridge types). For a stand-alone bridge, dividers are usually set. If you are joining two bridge sections together, then dividers are not needed on the ends of both bridges where they join—just one divider is needed.
Reverse	Disabled			We always consider that the span is at 45 degrees max from previous span, so this is not required
Ground Height	OL and Altitude Extractors		Max altitude under the span defined by OL	Reverses the default selection of the polygons or construction edges that were selected before running the procedure. If you select this, it results in a bridge connecting the edges in the opposite direction of the default selection.
Round Subdivisions	10		This value is sufficient for most circular shapes	Allows you to manually adjust the length of the legs or towers. Increasing the number of subdivisions makes the round beams rounder.

Section	Property	Input	Mapping	Definition
Angles				This is available for all types of bridges. It defines the start and ending vertical and horizontal angle offsets of the bridge.
Vertical	D Extractor	See Bridge Angles procedure		The Start Angle and End Angle determine the angle at which the bridge arches or bows. Can be set to flat but default is an adjusted setting where the Vertical Start and End Angles such that the bridge arches smoothly between the two edges.
Horizontal	D Extractor	See Bridge Angles procedure		The concept for the Horizontal angles is like the Vertical described above. However, this parameter determines how the bridge curves when viewed top-down. The angle is calculated from the angles of the initial edges. The Start Angle and End Angle fields determine the angle at which the bridge curves or bows out.
Deck				These are parameters related to deck of bridge (the part that carries the roadway or walkway)
	Thickness	Thickness Extractor		Sets the thickness of the deck. Any value over 0 is valid.
Subdivisions				Sets the number of subdivisions for the entire bridge. T-vertices are automatically avoided, which means the number is not always precise. Also, for some bridge types, the number of deck subdivisions must correlate with the number of spans. For example, for Solid Arch bridges, if the number of span dividers is set to two, then the number of deck subdivisions is already six. In that case, values of less than 12 do not change the number of subdivisions.
Left Overhang and Right Overhang	Roadside_Type Extractor, Outside Structure disabled			Sets the amount of overhang on the sides of the bridge, for example, to create a walkway. If Outside Structure is enabled, it will move the towers or uprights to inside of the overhang, thereby putting the overhang outside of the bridge towers.
Overhang Height	Roadside_Type Extractor			Raises or lowers the height of the overhang in relation to the height of the deck. This is used to create raised or lowered walkways (a negative number to create a lowered walkway).
Texture			Reference	See Common Texture Settings

Section	Property	Input	Mapping	Definition
Supports				Parameters for the uprights or supports under the deck of the bridge.
	Type			Webbed or Pillar for the support type
Webbing Type				Available for Webbed type only. Many variants available. Variants differ based on the number of sections in the support columns.
Sections				the number of sections within each support column
Partitions				the number of divisions into which the support column is divided from side to side. Effectively, each section looks like its own support column under the bridge.
Gap Size				Changes the space between partitions (this is only available when Partitions is greater than one).
Solid				Makes the space between partitions solid (this is only available when Partitions is greater than one).
Top Scale Ratio				Makes the top section of the support larger or smaller compared to the bottom section(s).
Current				Each member of the support (verticals, horizontals, or diagonals) can be assigned a Member Width and Member Depth.
Round				makes the selected member round
Texture		Reference		See Common Texture Settings
Tower				Parameters for the towers on Suspension bridges and Cable Stayed bridges
	Clearance			Sets the "drive under" height for the horizontal tower members. That is, a higher value will raise the horizontal tower members so that the height from the deck to the bottom-most member. The other horizontal members are also raised to keep the space between them proportional. Top Scale Ratio adjusts the upper horizontal members.
Webbing Type				Many variants available. Variants differ based on the number of sections in the tower columns.
Sections				Sets the number of sections vertically within each tower
Partitions				Sets the number of partitions horizontally within each tower. Different variants have different effects.

Section	Property	Input	Mapping	Definition
Tower (cont.)	Gap Size			Sets the distance between inner or outer cables. Only available when Outer Cables or Inner Cables are set to more than one.
	Top Scale Ratio			Sets the height of the upper horizontal members relative to the lowest horizontal member. (Use Clearance Height to set the lower horizontal member height.)
	Outer Cables			Sets the number of cables on the outside of the bridge. Only available when Partitions is more than one.
	Inner Cables			Sets the number of cables on the inside of the bridge. Only available when Partitions is more than one.
	Current			Each member of the support (verticals, horizontals, or diagonals) can be assigned a Member Width and Member Depth.
	Round			makes the selected member round
	Texture	Reference	See Common Texture Settings	
Suspension Profile	Cable Height			Parameters for Suspension bridges and Converging Suspension bridges. A Suspension bridge is a bridge that supports its deck with many tension members attached to cables draped over tower piers. A Converging Suspension bridge is similar to a Suspension bridge except that at the ends the main cables extend all the way to the deck.
	Displacement			Sets the height of the towers relative to the arch Displacement. This value also affects the overall height of the towers, the cable height takes this value as a multiplier.
	Curviness			Sets the curve of the suspension cable. A higher value changes the angle at which the cable attaches to the tower, resulting in a more pronounced curve. Subdivisions below make the curve of the cable look more realistic.
	Supports			Sets the number of upright supports for the suspension cable.
	Subdivisions			Sets the number of sections into which the suspension cable is divided. More sections give a more curvy appearance.

Section	Property	Input	Mapping	Definition
Suspension Profile (cont.)	Current			Each member of the support (Main Cable, Vertical Cables, and Mid Arch Vertical Cables) can be assigned a Member Width and Member Depth. The Main Cable is the suspension cable that drapes over the towers; the Vertical Cables are the cables that run from the deck to the Main Cable; the Mid Arch Vertical Cables are the vertical cables at the mid point of the Main Cable (the lowest point in the dip of the Main Cable).
	Round		Makes the selected member round	
	Texture		Reference See Common Texture Settings	Parameters for Tied Arch, Open Spandrel, Cantilevered Open Spandrel, and Cantilevered Through Arch bridges.
Arch Webbing	Partitions			Sets the number of cross members on the top of the bridge.
	Gap Size			Sets the distance between the inner or outer cables. Only available when Outer Cables or Inner Cables are set to more than one.
	Outer Cables			Sets the number of cables on the outside of the bridge.
	Inner Cables			Sets the number of cables on the inside of the bridge.
	Top Scale Ratio			Sets the height of the upper horizontal members. (Use Clearance to set the lower horizontal member height.)
	Clearance			Sets the "drive under" height for the horizontal tower members. That is, a higher value will raise the horizontal tower members so that the height from the deck to the bottom-most member. The other horizontal members are also raised to keep the space between them proportional. Use Top Scale Ratio to adjust only the upper horizontal members.
	Webbing			Sets the Bottom, Side, and Vertical webbing to one of the available values. Set Secondary Arch to enabled to create a second arch beneath the current arch.
	Vertical Sections			Sets the number of End, Mid, and Intermediate sections.

Section	Property	Input	Mapping	Definition
Solid Arch Bridge	Arch Height			Parameters for the Solid Arch and Cantilevered Solid Arch bridges. Solid Arch bridges have an arch below the deck that is filled with masonry.
	Displacement			Sets the height from the top to the bottom of the arch.
	Leg Width			Sets the distance from the deck to the top of the arch. Increasing values moves the peak of the arch closer to the ground by increasing the thickness of the deck. This parameter also affects the overall height of the bridge, so you may want to adjust this value before setting the Arch Height.
	Vousoir Size			Sets the width of each leg of the bridge, effectively increasing the size of the arch between the legs.
	Half Arch Leg			Sets the thickness of the bridge section that lines the top of the arch.
	Curviness Current			Sets the number of subdivisions for each arch.
	Texture		Reference	Sets the number of subdivisions for each leg of the bridge. Usually this value should be set at one unless the bridge curves.
Arch Profile	Arch Height			Sets the curviness of the arch. Valid values are from 0 to 1.
	Displacement			Sets the texture on each bridge element (Spandrel, Vousoir, Soffit).

Section	Property	Input	Mapping	Definition
Arch Profile (cont.)	Curviness			Sets the curve of the arch. Valid values are from 0 to 1, with higher values providing a more pronounced curve.
	Supports			Sets the number of upright supports that extend between the deck and the arch of the bridge. Note that this value is for the half deck, that is, a value of 4 will put four supports on each side of the bridge, plus the center support, for a total of 9 supports.
	Subdivisions			Sets the total number of subdivisions for the arch of the bridge. A higher value increases the curve of the arch.
	Current			Each member of the support (e.g. Main Arch, Secondary, Webbing, Ends, etc...) can be assigned a Member Width and Member Depth.
	Round			Makes the selected member round.
	Texture		Reference	See Common Texture Settings
Covered Bridge				Parameters for the Covered bridge, which is a bridge that has solid covered sides and top.
	Partitions	1		Sets the number of covered sections for the width of the bridge. This is the section on the deck that is driven through, so the default value of 1 provides the most common configuration.
	Height			Sets the height for the covered section of the bridge.
	Entrance Angle			Sets the angle for the entrance. A value of 0 results in a right angle. Higher values (up to 89 degrees) make the entrance angle lean toward the middle of the bridge.
	Walls Angle			Sets the angle for the walls in relation to the inside of the covered section. A value of 0 results in a right angle. Higher values (up to 89 degrees) make the walls lean inward toward the center line of the roadway.
	Texture		Reference	See Common Texture Settings
Solid Bridge				Parameters for the Solid Bridge, which is a bridge that has solid sides reaching to the ground.
	Left and Right Angles			Sets the angle in degrees that the left or right side slopes outward. Valid values are from 0 to just less than 90 degrees.
	Current			Sets the right and left base textures of the bridge.
	Texture		Reference	See Common Texture Settings

Section	Property	Input	Mapping	Definition
Trestle Structure	Sections per Span			Parameters for a Trestle bridge, which is a railroad style bridge with many supports reaching to the ground.
	Height Sections			Sets the number of vertical sections per span by adding or removing upright supports.
	Left and Right Angles			Sets the number of horizontal sections per span by adding or removing horizontal supports. Note that this sets the value based on the part of the bridge with the highest ground clearance.
	Top Scale Ratio			Sets the angle in degrees that the left or right side slopes outward. Valid values are from 0 to just less than 90 degrees.
	Webbing Type			Sets the distance from below the deck to the first horizontal support. Note that the other horizontal supports are also moved in relation to the first horizontal support.
	Current			Sets a variant for the webbing.
	Round Texture		Reference	Each member of the support can be assigned a Member Width and Member Depth.
				Makes the selected member round.
Truss Structure	Side Type			See Common Texture Settings
	Top Type			Parameters for the Truss bridge, which is a railroad style bridge. Note that it is different from Trestle bridge because it does not have horizontal supports and webbing that extend to the ground.
	Sections per Span			Sets the type of truss for the side of the bridge.
	Partitions			Sets the type of truss for the top of the bridge.
	Angle			Sets the number of sections for each span of the bridge.
	Inside Height			Sets the angle for the walls. A value of 0 results in a right angle. Higher values (up to but not including 90 degrees) make the walls lean inward toward the center line of the roadway.
				Sets the height of the side supports. If this value is not set, the height is automatically determined by the bridge dimensions, truss type, and other properties.

Section	Property	Input	Mapping	Definition
Truss Structure (cont.)	Current			Each member of the support can be assigned a Member Width and Member Depth.
	Round		Reference	Makes the selected member round
Cable Stayed Bridge	Texture		See Common Texture Settings	Parameters for the Cable Stayed bridge, which is a bridge where the deck is supported by several cables reaching from the tower to the deck.
	Cables			Sets the number of cables extending from the tower to the deck for each half span.
Tower Height				Sets the height of the tower above the deck.
	Minimum and Maximum Height			Sets the minimum and maximum values for where the cables meet the tower. A value of 0 represents the deck and a value of 1 represents the top of the tower. Note that the Maximum Height must be equal to or greater than the Minimum Height.
Minimum and Maximum Position				Sets the minimum and maximum values for where the cables meet the deck. A value near zero represents the deck next to the tower and a value of 1 represents the deck at the end of the span. Note that the Maximum Position must be equal to or more than the Minimum Position.
	Current			Each member of the support can be assigned a Member Width and Member Depth.
Texture	Round		Reference	Makes the selected member round
	LOD	user defined ranges (view distances) for scene	Automatically generated based on view distances input by user	Up to 3 LODs can be generated automatically for each model based on a tolerance level and a distance from viewpoint in meters.
Common Texture Settings	Texture Extractor		Extracts imagery texture to be applied on bridge	Each element in the bridge can be either textured or colored. A color can be chosen, e.g. for the deck or supports or a texture can be assigned using the following parameters:
	Type	Texture Extractor, Real World Imagery	always extracted from	Can be either Real World, Nearest, or Tiled for both the U and the V direction.

<b>Section</b>	<b>Property</b>	<b>Input</b>	<b>Mapping</b>	<b>Definition</b>
Common Texture Settings (cont.)	Size	N/A		Sets the size of the texture along U and V. Note that Real World Size and Nearest use the RealWorld Size value specified in the texture attribute file (.attr). For Tiled, the value set here specifies how many times to tile the texture.
	Align	Bottom, Left		Choose Left, Center, Right, or Random for U and Bottom, Center, or Top for V. With Random, the model should be tested visually and the value tweaked until satisfaction.
	Rotate Texture	N/A		Rotates the assigned texture 90 degrees.

## ***Appendix B***

### ***OWL Link Porting***

OWL Link is a specification and protocol for communication (mainly over http) to access OWL 2 Reasoning Services. It facilitates the configuration of a reasoner, the sharing of OWL 2 information, and the access of reasoning services via a set of basic queries. It is extensible by allowing the addition any desired client-server functionality.

OWL Link API is a Java-based framework that implements the specification and protocol. It is built on top of the OWL API which enables applications to access remote reasoners or server applications implementing the OWL API. It also allows the mediation between OWL API versions through client and server adapters. Due to its flexible and reasoner-independent API, the OWL Link API is the framework of choice for application developers requiring the interchange of reasoners or testing different reasoning services. This makes it ideal for use in our implementation especially since research in semantic web is very active and reasoners are constantly improving.

There is, however, no native C++ or C# implementation of this protocol although a C++ application could be able to interact with the Java OWL Link API adapter over HTTP. We, therefore, added an objective in our implementation to port OWL Link API to C++ and C#. We have initially tried converting the OWL Link API Java source code to C# source code through a semi-automated code converter. However, available code converters do not work very well especially in terms of dependencies and unavailable source code (only binary Jars available). Our final implementation is based on the open IKVM toolkit which implements a Java virtual machine (based on OpenJDK) for the Mono or the Microsoft .Net frameworks. IKVM.NET includes the following components:

- A Runtime Java Byte Code Virtual Machine implemented in .NET
- A .NET implementation (DLLs) of the Java Kit libraries (based on OpenJDK)
- Binary and Conversion tools (mainly IKVMC) from Java to C#
- Tools that enable Java and .NET interoperability

These components would allow using the OWL Link API in any .Net application (including C++ applications) to interface with any other application using the OWL Link API (client-server communications). OWL Link API dependencies such as the OWL API and some other dependencies require conversion as well for the OWL Link API Jar library to work. These already have Java binary libraries (.jar) available which we used to develop a working package using IKVM. To help in this effort, some tools exist that analyze dependencies between Jar files (JarAnalyzer), generate a map for converting a list of Jar files to .Net DLLs for use under IKVM (Jar2IKVMC), and reconstruct .Net DLL source code and view references for debugging and verification (Reflector). No single tool works perfectly, so we had to customize input/output to have a final working

solution. JarAnalyzer (<http://www.kirkk.com/main/Main/JarAnalyzer>) had some problems specifically in resolving a complete list of references but is a good starting point as it works on the original Jar files of OWL Link API and its dependencies. We first retrieved all the existing Jar files in the solution:

owlapi-src.jar	20/08/2010 12:29 ...	Executable Jar File	owlapi-3.1.0 (C:\U...)
owlapi-bin.jar	20/08/2010 12:29 ...	Executable Jar File	owlapi-3.1.0 (C:\U...)
owllink-src.jar	18/06/2010 2:12 PM	Executable Jar File	owllinkapi-1.0.2 (...)
owllink-javadoc.jar	18/06/2010 2:12 PM	Executable Jar File	owllinkapi-1.0.2 (...)
owllink-bin.jar	18/06/2010 2:12 PM	Executable Jar File	owllinkapi-1.0.2 (...)
owlapi-bin-2.jar	18/06/2010 2:12 PM	Executable Jar File	lib (C:\Users\Kam...)
owlapi-bin.jar	18/06/2010 2:12 PM	Executable Jar File	lib (C:\Users\Kam...)
org.mortbay.jmx.jar	18/06/2010 2:12 PM	Executable Jar File	lib (C:\Users\Kam...)
org.mortbay.jetty.jar	18/06/2010 2:12 PM	Executable Jar File	lib (C:\Users\Kam...)
junit.jar	18/06/2010 2:12 PM	Executable Jar File	lib (C:\Users\Kam...)
javax.servlet.jar	18/06/2010 2:12 PM	Executable Jar File	lib (C:\Users\Kam...)
commons-logging-1.1.1.jar	18/06/2010 2:12 PM	Executable Jar File	lib (C:\Users\Kam...)
org.semanticweb.owllink.protege.jar	18/06/2010 2:12 PM	Executable Jar File	owllinkapi-1.0.2 (...)
Hermit.jar	28/04/2010 2:05 PM	Executable Jar File	lib (C:\Users\Kam...)

We removed unnecessary Jar files from the list such as the JavaDoc and Source Jars (we are only interested in binary versions) and others by using JarAnalyzer to understand dependencies. Some dependencies (Jar files) are not part of the list shown so we will have to build them or find a copy of their already built version. We will do that at a later stage when building the package as some of the dependencies will be resolved by the IKVM package libraries. We used Jar2IKVMC (<http://code.google.com/p/jar2ikvmc/>), which also depends on JarAnalyzer to generate a map for IKVMC, a tool part of IKVM that compiles Java classes and libraries into a .NET assembly (DLL). Jar2IKVMC generates a dependency matrix and an ordered set of IKVMC commands that can be run to compile the needed Jars into their corresponding DLL versions.

In our case, running “jar2ikvmc \dev\ map.bat” on the command line generates the following IKVMC commands based on the previous list of required files. The “-target:library” option generates a .dll of the same name as the Jar library name while the “-r:” option defines specific dependencies other than default IKVM package libraries:

```
ikvmc owlapi-bin2.jar -target:library
ikvmc owlapi-bin.jar -target:library
ikvmc org.semanticweb.owlapi.jar -target:library
ikvmc mail.jar -target:library
ikvmc junit.jar -target:library
ikvmc javax.servlet.jar -target:library
ikvmc javax.jms.jar -target:library
ikvmc velocity-dep-1.jar -target:library -r:javax.servlet.dll -r:javax.jms.dll
ikvmc log4j-1.2.16.jar -target:library -r:javax.jms.dll -r:mail.dll
ikvmc org.semanticweb.owlapi.apibinding.jar -target:library -r:mail.dll -
    r:log4j-1.2.16.dll -r:velocity-dep-1.dll -r:javax.servlet.dll
ikvmc commons-logging-1.1.1.jar -target:library -r:log4j-1.2.16.dll -r:velocity-
    dep-1.dll -r:javax.servlet.dll
ikvmc org.mortbay.jetty.jar -target:library -r:commons-logging-1.1.1.dll -
    r:javax.servlet.dll
ikvmc org.semanticweb.owlapi.owllink.jar -target:library -
    r:org.semanticweb.owlapi.apibinding.dll -r:org.mortbay.jetty.dll -
    r:org.semanticweb.owlapi.dll
ikvmc unittest.jar -target:library -r:org.semanticweb.owlapi.apibinding.dll -
    r:org.semanticweb.owlapi.owllink.dll -r:junit.dll
ikvmc org.semanticweb.owllink.protege.jar -target:library -
    r:org.semanticweb.owlapi.owllink.dll -
    r:org.semanticweb.owlapi.apibinding.dll
ikvmc org.mortbay.jmx.jar -target:library -r:org.mortbay.jetty.dll -
    r:commons-logging-1.1.1.dll
```

Running “map.bat” under the IKVM environment will generate all corresponding .Net DLL files for use part of .Net applications and run under the IKVM virtual machine. However, we encountered some problems due to some missing Jar dependencies. We, however, found all required files in binary form on the internet. Reflector 6 (<http://www.red-gate.com/products/dotnet-development/reflector/>) free edition was used to identify those dependencies in resulting DLLs and for debugging purposes.