2007

Microstructural effects on the mechanical properties of carburized low-alloy steels

Erin Boyle
University of Windsor

Using Sensor Ontologies to create Reasoning-Ready Sensor Data for Real-time Hazard Monitoring in a Spatial Decision Support System

by

James Dwight McCarthy

A Thesis
Submitted to the Faculty of Graduate Studies
through the Department of Earth and Environmental Sciences
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2007

Your file Votre référence
ISBN: 978-0-494-34970-0
Our file Notre référence
ISBN: 978-0-494-34970-0

Abstract

In order to protect at-risk communities and critical infrastructure, hazard managers use sensor networks to monitor the landscapes and phenomena associated with potential hazards. This strategy can produce large amounts of data, but when investigating an often unstructured problem such as hazard detection it can be beneficial to apply automated analysis routines and artificial intelligence techniques such as reasoning. Current sensor web infrastructure, however, is not designed to support this information-centric monitoring perspective. A generalized methodology to transform typical sensor data representations into a form that enables these analysis techniques has been created and is demonstrated through an implementation that bridges geospatial standards for sensor data and descriptions with an ontology-based monitoring environment. An ontology that describes sensors and measurements so they may be understood by an SDSS has also been developed. These tools have been integrated into a monitoring environment, allowing the hazard manager to thoroughly investigate potential hazards.

Dedication

To the family, friends, and teachers who have supported my life as a student for the last twenty years, thank you.

I wish to dedicate this work to my parents, Jim and Wendy. Their unending and unconditional love and encouragement have contributed more to this work than they will ever know, and for that I am truly grateful.

Acknowledgements

While this work may have a single name as an author, it is the support of many people that has made it possible. First, I would like to thank Dr. Phil Graniero, who gave me the opportunity to embark on this challenge and supported me every step of the way. His passion for what we do is infectious and pushed me when I needed it most. I wish to thank my fellow researchers in the Geotechnical In-Situ Sensor Technology Network as well as the Sensor Web Automation Network for their support. To my friends in the Multi-purpose Environmental Modelling Facility, especially Steve Rozic, Nafaâ Jabeur, Dan D'Alimonte, and Xitao Xing, thank you for your support and for reminding me that I'm not the only person in the world who finds this stuff interesting. I would like to thank Dr. Alan Trenhaile and Dr. Edwin Tam for serving on my thesis committee, Dr. Aaron Fisk for chairing my defense, as well as the staff and faculty of the Department of Earth and Environmental Sciences for their support. A special thank you also goes to all of the teachers and classmates I've had over the years as well as the students and professors I've had the pleasure of working with. I would also like to acknowledge the generous financial support of GEOIDE and OCE.

Finally, I would like to extend a heartfelt thank you to my family and friends. To my parents Jim and Wendy for their unwavering and eternal support, to my brothers Scott and Joe for always being there for me and for being great role models to aspire to, and to the rest of my family, your support has not gone unnoticed and has made this work possible. To my friends, especially Matt, Jay, and Dani, we've been through many adventures together, and this is one adventure I would not have completed had it not been for all of you.

Table of Contents

List of Figures

List of Code Listings

List of Abbreviations

AJAX: Asynchronous JavaScript and XML
CLIPS: C Language Integrated Production System
COOL: CLIPS Object Oriented Language
CSV: Comma Separated Values
CVS: Concurrent Versions System
DSS: Decision Support System
ECO-COSM: Extensible Component Objects for Constructing Observable Simulation Models
ESRI: Environmental Systems Research Institute
GIS: Geographic Information System
GIST: Geotechnical In-Situ Sensor Technology
GML: Geography Markup Language
GRASS: Geographic Resources Analysis Support System
IEEE: Institute of Electrical and Electronics Engineers
MEMF: Multipurpose Environmental Modelling Facility
O&M: Observations and Measurements
OGC: Open Geospatial Consortium
OWL: Web Ontology Language
RDF: Resource Description Framework
REASON: Real-time Evaluation Applying Sensor Ontologies
SAR: Synthetic Aperture Radar
SCADA: Supervisory Control and Data Acquisition
SensorML: Sensor Model Language
SDL: Semantic Data Language
SDSS: Spatial Decision Support System
SOS: Sensor Observation Service
SQL: Structured Query Language
SUMO: Suggested Upper Merged Ontology
SWE: Sensor Web Enablement
UML: Unified Modelling Language
VBA: Visual Basic for Applications
W3C: World Wide Web Consortium
XHTML: Extensible HyperText Markup Language
XML: Extensible Markup Language
XSLT: Extensible Stylesheet Language Transformation
XST: Extensible Set Theory

1. Introduction

1.1. Problem

The identification of impending natural hazards is a worthwhile research objective for many reasons. The early detection of a hazard has the potential to protect physical infrastructure, conserve natural resources, and save lives. Methods to detect hazards vary depending on the type of hazard, but are often built around the concept of using expert criteria for the identification of hazards or the determination of hazard potential based on analyzing field data collected from site(s) of interest. If it is determined from this analysis that hazards are likely, then it is up to administrators and decision makers to plan the next course of action based on this information. For example, when a forest is dry, the potential for wildfires to occur increases, and appropriate precautions can be taken to avoid the creation of sparks that may trigger a fire. The hazard manager is able to decide on an appropriate course of action (such as fire remediation), based on the most recent information (moisture levels). These determinations are made using a combination of accurate information and expert knowledge. The crux of any hazard monitoring problem, then, is to gather and analyze relevant information in a timely fashion using domain expertise. This presents two distinct, but related problems with their own unique challenges: the collection of data and the analysis of those data.

Before examining these problems, a distinction must be made between two terms that are often used interchangeably: data and information. Data are some observations or facts without context, whereas information is a collection of data organized in some logical manner that is relevant to a problem. The difference between these two states of knowledge is subtle, but when dealing with knowledge representation it is fundamental to know exactly what the intended use of the knowledge is so that it may be structured correctly. The tools and methodologies to make the transition from data to information in an intelligent, automated fashion form a large part of this thesis.

The collection of relevant data can be done using various methods such as manual collection using probes and in-situ sensors, remote sensing, field observation, and the use of automated sensor networks. Manual collection of data through sampling, probing, or other methods may be useful for the analysis of a very specific problem or when semi-quantitative or qualitative information is needed. The use of sensors to collect quantitative data that are relevant to a problem is a more fruitful endeavour, but retrieving data from the sensors can be costly, dangerous, and time-consuming if the sensors are

placed in a remote or hazardous area. Further, when decision makers wish to do hazard monitoring by detecting small changes over long periods of time, manual sampling is inefficient when compared to automated sampling methods. Automated methods provide the long-term collection capabilities that are needed by decision makers to monitor hazards, and to support simulation as a method of problem exploration. Automated methods also reduce the number of field excursions needed by making use of automated data collection and dissemination methods. The use of automated collection methods does not, however, eliminate the need for validation of the collected data. Employing strategies to verify the incoming data as correct is perhaps more important when using automated collection methods, as there is no first-hand verification of the data as there would be with a manual collection routine.

Analyzing relevant information is what allows decision makers to draw conclusions about the state of the system under consideration, and ultimately make decisions. Using problem-specific knowledge as context, expert users can look at the information presented to them and make informed decisions. This is what occurred in the wildfire scenario above. The domain knowledge can be considered as an ontology, or a model of the important concepts and relationships in that domain, which is discussed in Section 4. Any problem we wish to automatically monitor must be understood in some measurable, quantifiable manner, or in a way that can be qualitatively modeled using symbols. We must be able to model our problem space accurately enough that we can feed numerical measurements to our representative model and get reliable and usable results. The problem of hazard monitoring is often somewhat unstructured and relies on more symbolic or qualitative modeling and heuristic reasoning. This approach mimics what the expert user does; only in this case the role of domain expert is supported by software which has domain knowledge encoded in its knowledge base.

Relevant data are not always readily available, and can often be buried within massive data stores, so locating and identifying relevant data can be an issue. There is also a gap between the collection of the data and their usage, as there must be some infrastructure in place that takes the data in their collected form and delivers it as useful information to the decision maker. Domain experts must be able to retrieve the information in a form that is useful for problem solving, not just simple number crunching, allowing the decision maker to apply the information to their problem assessment. In all of these cases, the expert's ontology that models the problem domain is a key element in finding and filtering the data and transforming them into relevant information. The same data problems must be addressed in automated reasoning. Therefore an ontology-based decision support system and data

infrastructure can help support automated reasoning and hazard identification. These are some of the major issues which must be addressed to create a knowledge-based approach to hazard monitoring, all of which are addressed in some way in this thesis. The remainder of this chapter will demonstrate the rationale behind solving these problems as well as a generalized architecture used to build and integrate these solutions.

1.2. Thesis Objectives

The main objective of this thesis is to use sensor data collected by sensor networks for real-time hazard monitoring in a spatial decision support system (SDSS), and to investigate the use of ontologies as a means of increasing the usable information content of the data so an SDSS can be a more effective automated reasoning tool. The objective is approached with the operative hypothesis that: i) a machine-usable representation of the ontologies that model both the sensor and problem domains can be used to automate the data-to-information transformation and the reasoning process; and ii) the sensor ontology can be automatically produced from sensor descriptions in existing sensor web and geospatial infrastructure standards. To do this we must provide problem domain-specific context for sensor data and present that information in a form that is machine readable and understandable. This requires a series of sub-objectives to be met. First, the ontological needs of the sensor network domain must be investigated. This involves a thorough review of the sensor domain to find all of the key concepts that can be used to provide context to sensor data. From this information, an ontology must be created and made accessible to the systems using the sensor data. Various ontology development methodologies must be explored to ensure that the end result is a useful one. Tools must then be developed to take sensor data from typical geospatial storage infrastructure and convert it into an ontological form. This needs to be done in an automated way so that existing monitoring workflows are minimally affected. This information must then be integrated with an ontology-based spatial decision support system. This will allow the information to be analyzed within the context of the problem using advanced problem solving methods. The work will extend the capabilities of the REASON spatial decision support framework (Rozic, 2006), a system that provides expert users the ability to create a spatial decision support system that uses spatial- and knowledge-based analysis to aid expert users in analyzing problems. The extensions of REASON will allow it to automatically discover and use sensor data found through the sensor web infrastructure in a way that enables these types of analysis. All of these tools must be integrated and the entire workflow must be tested to ensure that the results of the

analysis based on the new ontology are reliable. Once this workflow is created, analysis routines can be created that will make use of the ontological information. The system must demonstrate usability in a hazard monitoring and detection application and show the benefits of ontology usage for hazard detection. It must also demonstrate usability and interoperability with distributed spatial databases and the standards that make up the current sensor web and geospatial infrastructure.

1.3. Significance of Research

This research is aimed at filling a gap that exists between advanced problem solving methods and the data which are currently used to feed more traditional analysis methods. Artificial intelligence techniques used to analyze and solve problems are continually advancing, and while they are powerful ways to solve problems, they typically need to use specialized information structures to represent the inputs to their problem. Through the use customized software tools that exploit common, open data standards, this work will show that it is possible to automate much of the work that needs to be done to take data that are more typically suited to conventional data-centric analysis and transform them so that they can be used in more information-centric analysis problems. It will show the benefits of using ontologies to structure knowledge and how the use of different levels of knowledge representation for different tasks can help the organizational aspect of hazard monitoring. Creating a specific transformation engine to bridge this gap would be feasible, but by generalizing some of the various levels of knowledge representation and moving between them in an automated way it becomes more feasible to apply these advanced methods to more traditional problem spaces, regardless of the specific architecture of a given monitoring workflow. The software produced in this thesis consists of a spatial decision support framework, a data transformation engine, and connections to the standard geospatial infrastructure. This software and demonstration application provide the foundation to build live monitoring systems in the future that use this new information-centric approach for the detection of hazards.

1.4. Chapter Outline

The remainder of this thesis is organized as follows. Chapter 2 introduces the key domains that apply to this project, specifically by examining the literature and some of the relevant technologies associated with geotechnical hazard monitoring, spatial decisions support systems, sensor webs, data encodings for knowledge representation, and ontologies. Chapter 3 explains how data encodings are

applied in this work. It also presents a classification scheme for these and other encodings based on their purpose in a typical monitoring workflow. It details the creation of a transformation engine based on this encoding classification, and explains how the transformation can be generalized so it can be applied to other monitoring workflows. Chapter 4 presents a general methodology for the development of an ontology. This methodology is illustrated through the development of a sensor ontology to be used in our reasoning engine. Chapter 5 explores our monitoring environment, and the additions that have been made to it over the course of this project. It explains the purpose and creation of the software tools that were created to enhance the capabilities of the monitoring environment and how the various tools have been integrated. It finishes by exploring the entire monitoring workflow from data to information to analysis. Chapter 6 presents a case study of how this system works in a geotechnical domain with the use of realistic geotechnical models for slope failure. Chapter 7 contains some concluding remarks about the research as well as some future directions that could be pursued.

2. Background

2.1. Geotechnical Hazard Monitoring and Spatial Decision Support Systems

To manage hazard risk, we must consider two important aspects of the problem of hazard assessment. The first is identifying what the likelihood of the hazard is, and the second is identifying who and/or what is vulnerable to the hazard. A large population centre with no corresponding likelihood of the hazard occurring does not pose a risk. Likewise, high likelihood of a hazardous event in completely unpopulated areas does not pose a risk. Only when there is a likelihood of a hazard occurring in areas where there are people and/or infrastructure that are vulnerable to that hazard does the hazard warrant monitoring. Using this principle we can direct our monitoring efforts and resources to the areas that are the most susceptible to hazard risks.

It is important, then, to be able to detect the potential for a hazard to occur based on available data so that the responsible parties can then plan response, mitigation and/or recovery strategies based on their evaluation of the potential effect of the hazard. Various studies have examined how this may be achieved by examining specific sites that have experienced slope failure of some form and trying to learn what the cause of the slope failure was and how it could be detected on other similar sites. As an example, Polemio and Petrucci (2001) investigated a mudslide that occurred in southern Italy and used the results from that investigation to determine the best course of action to evaluate landslide hazard potential based on remotely sensed data. Similarly, Zourmpakis *et al.* (2006) studied a site in southeastern England that was primarily composed of loess by subjecting it to controlled flooding and surface pressures. The site was monitored both geotechnically and geophysically, and the results of the geophysical monitoring were used to reinforce and verify the results of the geotechnical monitoring. Studies such as this serve not only to find better ways to combine instruments for a given monitoring task, but also act as a case study for similar sites so that the properties of hazard features can be understood. In an information-centric monitoring system these case studies can be described in ways that are understandable by a software system, which makes it possible for that system to compare the status of a monitored slope to a library of case studies that can be used to support or refute a hypothesis generated by the system regarding the hazard (Aamodt and Plaza, 1994). Crosta *et al.* (2006) provide another detailed study of landslide potential. In this case they did not examine a landslide that had occurred, but rather an area that is known to be susceptible to landslides. They integrated data from geological surveys and other field work, the study of triggering mechanisms, and

the results of other nearby landslide events to generate a risk factor to various areas with conditions analogous to those observed in the field and monitored with instrumentation.

Having this type of expertise available in a machine-usable form serves to reinforce any conclusions that a monitoring system may draw. Many natural and anthropogenic hazards can be detected through simple arithmetic methods. For example, rising water levels indicate that flooding may be occurring (Wassmann *et al.*, 2004), and the sudden motion of a slope can signify a slope failure (Hutchinson *et al.*, 2004). Being able to perform this kind of detection is important so that we may better understand a hazard, however by the time these simple conditions are seen it is often too late to enact adequate counter-measures or take the necessary actions to warn populations and protect infrastructure.

For these reasons an intelligent hazard monitoring system (or any intelligent monitoring system) should use its available knowledge to try and detect the *potential* for hazardous conditions, and any precursors to these conditions that may be identified by a domain expert. In other words, an intelligent flood monitoring system must monitor both rainfall and water levels so that rainfall can be used to predict future water levels or focus the monitoring efforts on certain locations. An intelligent slope monitoring system must monitor not only slope movement, but also the behaviour of the water table relative to the active areas of a slope. Monitoring of the water table and its proximity to landslide-prone areas can help predict when a slope is at risk of failure (Iverson, 2000). Like any data-driven problem, the quality of the analysis is directly related to the quality and quantity of the data being analyzed. The determination of what data to use and how confident one can be in the results is ultimately the responsibility of the expert user, however by looking for precursor phenomena and conditions that are known to trigger hazardous events, valuable time can be gained before the onset of a hazard, leaving the hazard manager with more preparation time.

A spatial decision support system (SDSS) is a typical way that this type of monitoring is done. A decision support system (DSS) can be defined as a computer-based information system that combines models and data in an attempt to solve unstructured or semi-structured problems with extensive user involvement through a friendly user interface (Turban *et al.*, 2005). An SDSS takes the DSS concept and applies it in a spatial context. An SDSS can take either the data-centric or the information-centric approach to hazard monitoring, depending on the intended purpose of the monitoring system. A system built to look for simple conditions and provide an alert when those conditions are violated can

be very effective data-centric systems. SCADA (Supervisory Control and Data Acquisition) systems take this approach and are used with great effect in the several industries, such as manufacturing, facility management, and infrastructure management (Kokai *et al.*, 1997). If a system is to be used to model a problem, run simulations of an environment, interact with and adjust the instruments providing the measurements, integrate multiple data sources, and for thorough problem investigation, then it should be built in an information-centric manner.

Yu *et al.* (2007) give an example of a system that uses data-centric methods to perform geotechnical hazard monitoring with positive results. It is a web-based system that uses real-time monitoring of rainfall combined with expert knowledge of where torrential rains are most likely to occur and the location of unstable slopes. The system is a tool for monitoring and problem exploration to determine where landslides and debris flow into creeks is most likely to occur on the main island of Taiwan. Their system integrates data from rain gauges, estimations from radar, water levels, and hazards information. In a browser-based GIS various risk parameters can be mapped, including rainfall levels, and landslide and debris flow potential estimations. Cheng *et al.* (2002) describe the development and use of a computer-based decision support system to monitor construction sites in which geotechnical stability is a concern. In this setting activities such as excavation, dredging, and other small- or large- scale construction activities are represented as GIS data within an integrated relational database. This information is passed through some analysis algorithms that apply fuzzy set theory in order to identify the possible cause of unexpected behaviour detected by geotechnical instrumentation. Harris *et al.* (2001) investigates the use of permafrost monitoring combined with climate monitoring stations to aid in learning more about how permafrost affects the movement of potentially unstable slopes and how permafrost behaviour can be used to help predict the motion of a slope.

These various examples show that a key factor when performing geotechnical hazard monitoring is to make use of, and integrate, data from several sources. The approach this project takes to this part of the problem is addressed in Chapter 3. The source of data that is most effective for our purposes of real-time or near-real-time evaluation and monitoring are field instruments that automatically measure areas of interest for the phenomena we wish to detect. This idea is embodied in the concept of sensor webs.

2.1.1 REASON

The REASON Spatial Decision Support Framework (Rozic, 2006) will provide the backbone of the demonstration system described later in the thesis. REASON can be used to develop rule-driven spatial decision support systems using an ontology based approach. It was developed using the ArcAgents extension (Ball and Harrap, described in Rozic 2006) to ArcGIS which embeds a CLIPS engine into ArcGIS. REASON builds on this connection by providing a CLIPS expert system that makes use of the spatial analysis capability of ArcGIS. The REASON architecture can be seen in Figure 1.

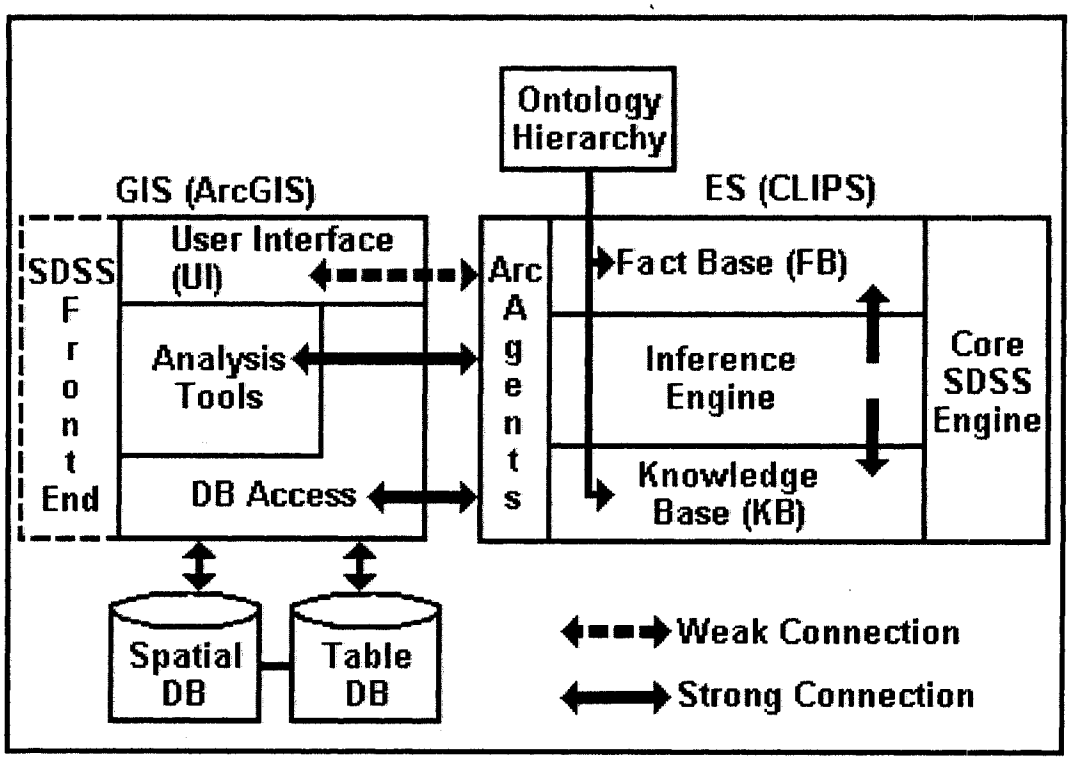

Figure 1 - REASON Expert System Architecture (Rozic, 2006)

Expert knowledge is partitioned into a series of ontologies to allow for customization across fields while maintaining portability of the source code, limiting changes to only application-specific concepts. The CLIPS reasoning engine provides facilities to reason on the knowledge stored in the system and ArcGIS provides the spatial analysis capability, making systems built on this framework powerful information-centric monitoring systems. REASON's abstracted data source mechanism allows it to make use of many different types of data; however the data binding is somewhat rigid in the sense that data

coming into the system must be structured in an expected way so that an appropriate data source handler can be created.

This work will improve on this aspect of the system by providing a sensor ontology that stores all of the measurement data in the context of the sensors that made the measurements. Software tools will be created that can transform the typically rigid data structures into more dynamic ontological data. This will in turn improve the analysis capabilities of the decision support system by providing the context for the incoming data.

2.2. Sensor Webs

There are many perspectives regarding sensor webs, typically focusing on a particular aspect of the technology that corresponds to a given research group's area of expertise. For example, NASA's Jet Propulsion Laboratory[1] defines a sensor web as "a new type of Geographical Information System (GIS) that can be embedded into an environment to monitor and control it. It is a spatially distributed, synchronous instrument that can react and adapt to changing environmental conditions." Their research tends to focus on environmental monitoring applications, and their definition reflects that focus.

Sensor webs are tools used for automated collection and storage of sensor observations, either integrating data from several separate sensor networks or acting in a coordinated fashion to generate aggregate or 'macro' observations. More generally, they are structures which move measurement data through a structured network from the sensors which collect the data to the applications which use them. They facilitate the collection, distribution, and dissemination of large amounts of spatially significant data, turning the Earth's surface, subsurface, oceans, and atmosphere into sensible entities (Gross, 1999). Initial sensor web implementations began as very basic networks, with a few sensors hard-wired together.

Quite often when a sensor web is deployed, having a traditional wired network is simply not possible. In recent years, the advent of mobile computing (Datta *et al.*, 1999) and low-power wireless communication technologies (Cetintemel *et al.*, 2003); (Kim and Yoo, 2005) has allowed sensor webs to be feasible in more realistic field settings. The methods used to transmit the data may vary, though

[1] http://www.jpl.nasa.gov/ Last Accessed August 20, 2007

most implementations tend to go through a data aggregation process which involves collecting several measurements and compacting them into messages so they can be sent more efficiently (Royer and Chai-Keong, 1999); (Madden *et al.*, 2002); (Heidemann *et al.*, 2001); (Krishnamachari *et al.*, 2004). This is especially important when using low-power wireless networks since capacity is always an issue (Gupta and Kumar, 2000); (Li *et al.*, 2001). All collected data must be sent completely and reliably, and in as small a message as possible so that the real-time arrival of data at the repository can be preserved.

With the continual advancement of sensor technology and wireless technology (Cetintemel, 2003), as well as growing support from the developer community, sensor webs have become a very useful and practical mechanism for automated data collection. As would be expected, this process can result in the collection of large amounts of data which can be used to feed analysis within a specified problem domain. Since the aim for any information-driven decision support system is to provide an expert user with relevant information that helps them to make informed decisions, sensor webs prove extremely valuable in providing data that may be transformed into timely information that is relevant to the problem.

Sensor networks can intelligently monitor their surroundings and are growing to be more responsive to external commands and control. Recent works have pushed to increase the intelligence in the way sensor networks are controlled. Jabeur and Graniero (Submitted) proposed a virtual layer of software agents that would sit on top of the sensor layer to manage much of the control and management of the network, taking advantage of higher processing capabilities to control functionality in a more intelligent manner while also extending the life of the power cells used by the sensors. Jabeur *et al.* (Submitted) shows how evaluation of incoming sensor data can be used to adjust the sampling behaviour of individual sensors to increase the relevance of the data. Stavroulaki *et al.* (2006) have worked to make sensor web services reconfigurable on the fly through the use of an overarching framework for distributed systems. They aim to take the many software tools that have been developed for sensor control and configuration and organize them into a high-level architecture to support the integration of these various tools. Fukatsu *et al.* (2006) describe an agent-based system to operate sensor nodes in the field via a web-based interface. The concept, design, and implementation of their system are discussed, as is the idea of how management of these types of web-based systems should be approached. A unique feature of their system is that the agents themselves generate web pages that

can be used to view the data collected by the sensors they are associated with, so not only do the agents control the behaviour of the system but they also help with data dissemination and presentation.

Measurement of slope motion and hazard potential can be accomplished in several ways, including the use of remote sensing and interferometry, and sensor networks. Colesanti and Wasowski (2006) explore how Synthetic Aperture Radar (SAR) can be used to monitor the potential size of slides, identify the amount of vegetation on the surface of a slope, measure the inclination of the slope, measure the movement of a slope, and determine the velocity and displacement of a slide event. Bovenga *et al.* (2006) use multi-temporal differential interferometry (an SAR technique) to investigate slope instability, (Meisina *et al.*, 2006) uses a similar SAR technique to analyze ground deformation. Terzis *et al.* (2006) apply wireless sensor networks to the prediction of landslides. By monitoring displacements of the sensor nodes, they calculate an estimated slip plane. This slip plane is used to feed an analysis routine that predicts the likelihood of a landslide. Sheth *et al.* (2007) have constructed SenSlide, a distributed sensor system used to predict landslides. Their focus was on making the system robust enough to withstand a slope failure while handling the typical wireless sensor network issues of connectivity and environmental variability.

Sensor webs and SDSS can be used not only for hazard detection, but also to plan sensor network deployment and manage the sensor network. The decision of which sites to monitor and what monitoring strategies to use are generally specific to the hazard being investigated, but there must be some suspicion that a hazard may occur before a monitoring strategy is employed. This suspicion may be based on the opinion of a domain expert, the results of simulation models, or the analysis of similar sites that have experienced a hazardous event.

2.3. Data Encodings and Representations

All of the studies noted in the last section rely on getting sensor data from the field to some remote analysis routine in a usable form. Finding effective methods to encode sensor data for different purposes is critical to providing a usable infrastructure for hazard monitoring. There are two approaches that may be taken: either a single encoding for all purposes, or some combination of encodings. If the single encoding approach is used, the encoding must be robust enough to account for all states of knowledge the system may encounter. Finding such a targeted encoding can be difficult, but if possible can decrease the complexity of the system. In many cases, the use of a single encoding is

not feasible, so a multiple encoding strategy must be used (such as in Bonnet *et al.*, 2004). When this is the case then there is often a need to convert between the different encodings. This section will examine some of the relevant encodings for hazard monitoring.

Knowledge representation methods span a wide range of complexity. Some are tailored to specific tasks while others are of a more generalized form. Semantic networks (Lehmann, 1992) can be used for knowledge representation when complex networks of concepts need to be represented. These networks are represented as nodes with relationships, similar to an ontology (see Section 2.4). Semantic networks were initially developed to aid in transforming human-readable information into machine-readable information. This idea has grown over decades of refinement into a practical method of both storing and presenting information for retrieval by computer programs. The use of semantic networks has grown to the point that researchers are now trying to enable the Semantic Web (Berners-Lee *et al.*, 2001). Semantic Web development aims to take the human-readable information content presented by the Internet and make it machine-readable. This would enable software programs to begin to understand the information stored on the World Wide Web, making it possible for agents or other programs to make intelligent use of the information.

Frames (Minsky, 1974) are another common way of representing knowledge. They take the approach that a frame can be used to represent an entity, and that the entity is described using attributes called slots. This again is a style that has been mimicked by the development of many ontological representations of knowledge. No matter the conceptual architecture used to structure the information, it must be rendered in some machine-readable format to be useful to software tools.

XML[2] has emerged as one of the standard and most common ways of cataloguing and exchanging information via the Internet. Its extensibility and scalability have made it a popular choice to represent everything from database records to web pages. Because XML is simply a specification of how data should be encoded, and not an implementation standard, its standardized way of encoding data can be applied to any representation task. Spatial information, for example, can be represented using the Geography Markup Language (GML) (Cox *et al.*, 2004). Its extensibility can be seen through the recent development of CityGML (Gröger *et al.*, 2006), an extension on GML used specifically to represent information about the objects and features found in cities and other urban areas. This demonstrates

[2] http://www.w3.org/XML/ Last Accessed August 20, 2007

one of the major advantages of using XML. If an existing encoding is close to what is needed for a given application, it can be extended for that application. This allows users to build on a given knowledge base while applying their own additions to it. Reuse is encouraged as opposed to having every encoding constructed from scratch. It is also easy to ignore information stored in an XML document, allowing the consumer of the document to target only the information they need. Since the format of the document is standardized, there exist many parsers that make extracting information simple (e.g., Expat[3] and Xerces[4]). Most programming languages have either a built in XML parser, or one which can be added on via external libraries, so interoperability between programming languages and software tools is rarely an issue when working with XML.

One disadvantage of XML is the size of a typical document. The documents are often verbose because all of the data elements are structured as a tree, and the beginning and ending of each node of the tree and the relationships between nodes are explicitly defined using tags. Because of this, XML documents often take up quite a bit of space when compared with database records or other proprietary formats. In modern server and desktop computing environments where high-bandwidth connections are normal, this is rarely an issue. For example, the recent development of more dynamic web pages and web based applications has been driven by the use of AJAX (Asynchronous JavaScript and XML) technology (Garrett, 2005); showing how XML can be used in a high-bandwidth environment with great effect. However, for sensors with limited processing power and communication bandwidth, large file sizes are a problem. Sending an entire XML document over a low-bandwidth modem would cause massive backlogs of information due to the time it would take to send each document. To avoid this problem, XML can be generated by the server after it has received the data in a more compact format like a database record, or it can be generated after a request from a client that wishes to have the data returned as XML. The latter is the approach that 52° North[5] takes in generating Observations and Measurements (Cox, 2006) documents in its Sensor Observation Service (SOS) (Na and Priest, 2006) software. Measurement records are stored to a database with a schema based on the O&M specification. When a request is made of that database from a client who needs the data, the SOS server pulls the relevant record(s) from the database and automatically generates the O&M document in XML format based on a template that is stored on the server. This way the sensors which publish the

[3] http://expat.sourceforge.net/ Last Accessed August 21, 2007
[4] http://xerces.apache.org/ Last Accessed August 21, 2007
[5] http://52north.org/ Last Accessed August 23, 2007

measurements can do so without the extra overhead that XML carries, while the client can make use of the extended descriptions that O&M provides.

Wagner *et al.* (2005) use an irregular wavelet transform for transmitting sensor data within and from a sensor network. It adapts to the hierarchy of the existing sensor network implementation rather than imposing its own structure so that it may best route its data. It is also possible to use lessons learned from other fields with similar constraints and problems. As an example, the field of genetics must deal with storing and representing extremely large amounts of data. Ontologies and databases are paired together (Stoeckert Jr. *et al.*, 2002) for use in this field. They apply microarray databases to store the massive amounts data that DNA and RNA stores, and recognize that there is a need for data management and transfer systems to make using these data possible.

Using the growing number of knowledge representation methods and tools to support decision support efforts, it is becoming simpler to integrate intelligent analysis methods into monitoring applications. Choosing the proper encoding(s) is a vital step in setting up any knowledge-driven system, but it is highly problem-dependent. The constraints and needs of the problem must be thoroughly explored before a decision can be made about how knowledge can best be represented.

2.4. Ontologies

Often when people with different areas of expertise attempt to collaborate, one of the biggest stumbling blocks is the vocabulary they use. An expert in one field may use jargon that is common knowledge to colleagues, but is unfamiliar to those who work outside that field. A similar problem also occurs when the same term has different meanings in different contexts, or when different fields use different terms to describe the same concept (the problem of semantic heterogeneity). However, if common terminology is agreed upon and understood by all parties, collaboration can continue with better understanding. This is, in a sense, the purpose of an ontology. An ontology can be considered broadly as a "specification of a conceptualization" (Gruber, 1993), in that the knowledge of a given domain or area of interest is explicitly defined and expressed in an organized way. An ontology specifies the important concepts in a domain and how they are related, giving structure to the knowledge about a certain domain. And in much the same way as a domain expert explaining terms can allow another person to understand the domain they are discussing, an ontology which is properly created can allow a piece of software to "understand" the domain it describes. A detailed discussion on the nature of the

term ontology and its roots can be found in (Øhrstrøm *et al.*, 2005). In this paper they discuss the origins of the term in philosophy to describe the study of being, and how it has since been applied in computer science. Figure 2 shows an example of how an ontological representation can be used to model concepts and relationships in a way that illustrates the utility of this approach. Figure 2a shows the data-centric approach to the representation of a sensor observation. The measurement (0.014) has attributes that are implicitly related to it, and by looking at the structure and content of these attributes one can interpret what they represent (a unit of measure, a time stamp, and a position). However a reasoning system does not have the tacit knowledge to draw on that an expert user has, and as such may not be able to interpret these attributes correctly. Figure 2b shows the application of an ontology to the representation of the sensor observation in an information-centric way. In this case the sensor observation (0.014) is related to its measurement attributes (such as the measurement location and time), the sensor that made the measurement (sensor 1), the attributes of that sensor (type, phenomenon measured, accuracy), and the other sensors that are connected to the network (sensors 4, 8, 15, 16, 23, and 42), through explicitly defined relationships that connect the concepts that are important to the domain. Information regarding the details of the other sensors can also be stored, as can further detail about seemingly simple attributes such as the measurement location. When this model is explicitly encoded in a machine-readable form, these concepts and relationships can be analyzed by appropriate software in order to better interpret the information stored within it.

Ontologies can, and have been, applied to many problems that deal with the management and representation of knowledge. Wang *et al.* (2007) use an ontology to verify feature models for use in domain engineering. Features of a given domain are represented as classes in the ontology and valid relationships between those features are also represented. Verification can then be done through reasoning or other processes to identify if a given configuration of features is valid or not according to the rules of the domain, or if inconsistencies in the configuration exist. Dietrich and Elgar (2007) make use of ontologies to detect and analyze design patterns within Java programs. Their ideal is to enable a web of design patterns and refactoring so that design improvements could automatically be applied to a system based on successful pattern usage in other similar cases. Chau (2007) uses an ontology-based knowledge management system to model the flow and quality of water. The study uses an ontology structure similar to the one used in our decision support engine, incorporating a domain ontology and an information ontology to partition its knowledge. It is used as part of a knowledge-based system that

incorporates Artificial Intelligence (AI) technology to assist with model selection and knowledge acquisition.

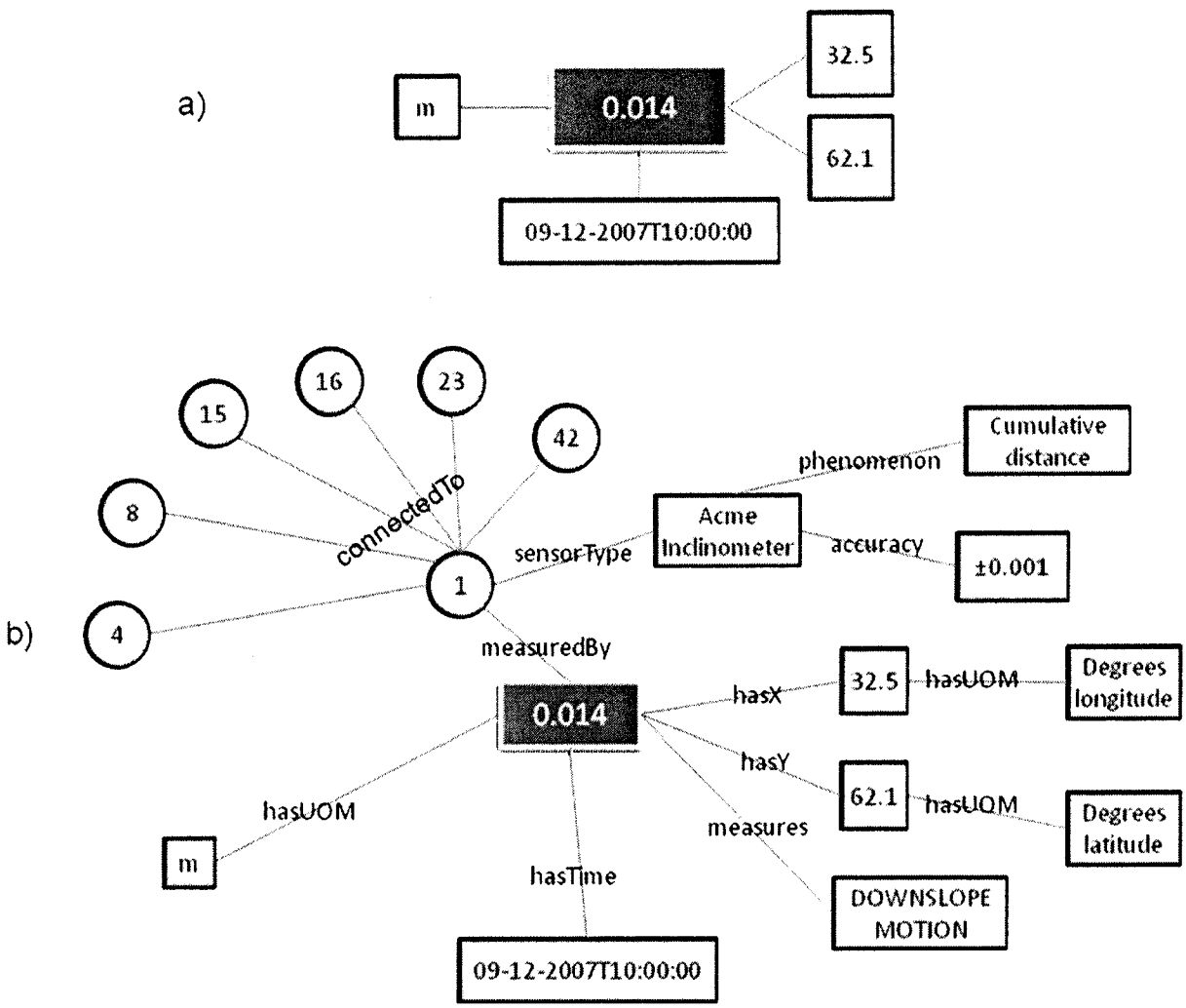

Figure 2 – a) Data-centric representation of a sensor observation; b) Ontological (information-centric) representation of a sensor observation

Some examples of how ontologies are used in a monitoring framework include the work of Pundt and Bishr (2002). They make ontologies available that describe their data in the environmental monitoring domain, making it possible for those not directly associated with the collection of the data to use the data in a proper manner. Their goal is to enable data sharing to support environmental monitoring in the hope that ontologies will provide the bridge needed to overcome the problem of semantic interoperability. In a much different domain, Zimmerman *et al.* (2005) describe the use of ontologies and agents to monitor supply chains. Ontologies provide a common language that enables

the agents to communicate in order to improve efficiency when tracking orders and handling inventory. Athanasiadis and Mitkas (2004) describe an agent-based system that uses ontologies for environmental monitoring. Their system uses meteorological data to monitor air quality and communicate the results to those in need of the information. An ontology is used to model the domain as well as provide guidelines for agent communications.

In many of these approaches, ontologies are either used in support of or as the backbone of the monitoring system. To apply ontologies to our monitoring problem, we can make use of sensor ontologies. A sensor ontology is a description of the domain of sensors. It explains what sensors are, the different types of sensors, how they operate, what their properties are, and how they are related. It allows specific sensors to be defined in the context of their domain and in a machine-readable format. Since the main goal of a sensor is to make measurements, then the concepts surrounding measurement processes and results should also be taken into account.

One example of a sensor ontology is OntoSensor (Russomanno *et al.*, 2005), a sensor ontology which makes use of both the Open Geospatial Consortium (OGC) and International Standards Organization (ISO) models for sensors. This is an approach that bridges these models for the purposes of embedding sensor knowledge into applications which need to reason on this information. Other ontologies used with sensor networks include Avancha *et al.* (2004) where ontologies are used to aid adaptation in the behaviour of sensor networks that are responsive to external phenomena. The ontology ensures that any adaptations made are valid and will not interrupt the monitoring tasks of the sensor network. In Santanche *et al.* (2006), Microsoft's SenseWeb project is introduced including the use of sensor ontologies for data discovery, fusion, and visualization. The ontology helps client applications properly interpret the data gathered from sensor networks.

The use of ontologies to represent knowledge has grown over the last decade in part because of a parallel growth in artificial intelligence research and also because of the increased storage and processing power of modern computing systems. However, since ontologies are still a single view of a complex world, they are only as expressive as their creators allow them to be. There are efforts to push towards more generalized high-level ontologies under which more targeted ontologies should fall, including IEEE's Suggested Upper Merged Ontology (SUMO) (Niles and Pease, 2001) and Cyc (Lenat, 1995), however, managing the cataloguing of thousands of domains leads to many problems such as

conflicting viewpoints, semantic heterogeneity, and logistical concerns. To achieve a vast, multi-domain, interconnected knowledge base is a massive undertaking that will take time to evolve, but as more projects begin to apply ontologies to the representation of knowledge in their domain there will be a larger set of examples and starting points to draw from when the time comes to merge these resources.

3. Classification and Transformation of Data Encodings

Sensor webs have the potential to collect very large amounts of data about a geographic region, and as the number of sensor web installations grow so will the amount of data available for problem solving. However organizing, discovering, and exchanging such large amounts of data can be a challenging task. It is this problem that has led many organizations, such as the Open Geospatial Consortium (OGC)[6] and World Wide Web Consortium (W3C)[7] to develop standards and encodings to represent various types of data. The aim of these initiatives is to enable the discovery and interchange of data so that people can find relevant data and apply them to their problem.

Data representation tends to be focused on representing individual data elements, such as a number or a name, in a structured way. This style of representation is worthwhile when trying to answer simple questions, but to answer complex questions we need more information about our data. Expressing the meaning of the data we are using and how they relate to other data, and doing so in an organized way can lead to more useful data. We could express the relationships by using symbolic links within our data that point at related data, or we could make use of metadata to express some of the meaning of the data, but the use of ontologies integrates all of our concepts into a single realm of knowledge. Using an ontology that defines all of the important concepts and relationships in our monitoring domain means software applications can be built which 'understand' it (Guarino, 1998). This subsequently results in an increased ability for the application to derive knowledge from sensor data as the ontology can provide context for the measurements that typically doesn't exist when using raw numerical values. This leads to applications that can use not just the data but the meaning of those data.

Nonetheless, many styles of representation exist for different purposes. The encodings relevant to the problem at hand can be broken down into three categories: data description languages, conceptual ontologies, and operational ontologies. This classification scheme is based on the typical use of these representations as well as how they would be applied to a monitoring problem.

[6] http://www.opengeospatial.org/ Last Accessed August 20, 2007
[7] http://www.w3.org/ Last Accessed August 20, 2007

3.1. Data Description Languages

Data description languages are those languages and encodings used to provide a defined representation for a given entity, but do not provide any explicit information regarding the relationships that the entities have, or how they relate to other entities (Wuwongse *et al.*, 2001). This is considered a 'data-centric' viewpoint on data storage. Database systems tend to take this data-centric approach. Relationships between entities can be implied by the structure of the records, or inferred from the structure of the database, but are not explicitly defined in a way that is fully understood by a software tool. Other examples include SensorML (Botts, 2005) and Observations and Measurements (Cox, 2006), as well as GIS metadata standards such as those supported by the International Standards Organization (ISO, 2003) and the Federal Geographic Data Committee's (FGDC)[8] core metadata framework.

In a typical monitoring system, these languages can serve several purposes. They can be used for archival of specification documents, measurements, and other pertinent information. These encodings also work well for data exchange and discovery since they provide information in a more human-readable format making them useful to catalogue data to be browsed by others. It is also good for efficient data mining since the data are not typically cluttered with extraneous information. Finding a specific piece of information in a structured way is typically quite simple and efficient since the queries are generally targeted directly to the language and there is very little ambiguity in the information declared in the document.

The lack of additional information about the meaning of the data does pose problems when the data are to be used in a reasoning-based system. The lack of explicitly defined relationships between concepts results in reasoning tools having to infer relationships from the structure of the documents. This can be problematic as it relies on the creator of the documents to structure their data a certain way, and that the structure is consistent with the way others do it. This also limits the interchange of data as the level of implied knowledge may not be the same by different data producers. This can lead to problems of misinterpretation as well as semantic heterogeneity. Also, without explicitly defining the relationships the implied relationships may be leaving out some tacit knowledge that may be obvious to a domain expert but not to a software tool. Both of these situations open up the possibility that a reasoning system could misinterpret the data. For example, a geologist would recognize that the

[8] http://www.fgdc.gov/ Last Accessed August 23, 2007

adjacency of rock units implies something about the relative ages of the units. However a reasoning system that understood the topological relationship between the units would not be able to infer that there is also a temporal relationship. This is why reasoning systems function best when relationships are explicitly defined.

3.2. Conceptual Ontologies

Conceptual ontologies are representations that are used to provide an 'information-centric' viewpoint of the entities which they describe, allowing for the description of concepts associated with a certain domain of interest. They express not only data, but the characteristics associated with those data, as well as the relationships between the concepts described within. Examples of languages that can be used to represent Conceptual Ontologies include the Web Ontology Language (OWL) and the Semantic Data Language (D.S. Mackay, unpubl. software)[9].

Benefits of these encodings in a monitoring environment center on the expressivity of the information. Relationships between concepts are explicitly defined and do not need to be inferred. They also provide a starting point from which others can build their own reasoning workflows. Expressing the concepts and relationships in a structured format allows others to take that information and permute it in a way that makes the most sense for their purposes. The specific instances of the concept within a data set must still be linked to the concept, at which point all the relationships associated with the concept become associated with the instances.

The main issues with these encoding types are that the amount of information encoded within is much larger than that of the Data Description Languages so finding a specific piece of information can sometimes be difficult for a human. The information is not as human-readable in its raw form, though with additional applications (such as Protégé[10]) can be made more human-friendly. The conceptual ontology is meant to be a reference model and while it can be more machine-readable than the Data Description Languages it can also be harder to build a monitoring application around a conceptual ontology, depending on the choice of representation and the tools available.

[9] http://water.geog.buffalo.edu/mackay/ Last Accessed August 22, 2007
[10] http://protege.stanford.edu Last Accessed August 21, 2007

3.3. Operational Ontologies

The role of an operational ontology is to represent an information-centric view in a format typically used by a reasoning engine, such as those written in programming languages like CLIPS[11]. The information in this representation closely resembles that which is found in Conceptual Ontologies. Often the information is identical, and only the syntax changes. It is represented in a manner more suited to reasoning engines, typically as source code in a programming language.

These ontologies have the same benefits as the Conceptual Ontologies with the added benefit that it is targeted to a specific application. This does, however, reduce the potential of reusing the information since those wishing to use the Operational Ontology for their reasoning purposes must build their system to work with whatever encoding is used in the target reasoning system. This is why the Conceptual Ontology proves useful as a starting point for the information-centric view, allowing others to 'spin-off' their own versions of the ontology encoded in a way that enables their monitoring system while still maintaining consistency with the information presented in the ontology. This is a highly desirable outcome that provides a common understanding between completely different reasoning systems.

It should be noted that a given ontology or encoding may in some instances act as both the Conceptual and Operational Ontology depending on the role it plays in the monitoring workflow. For example, an OWL ontology used for archival and reference purposes would be a Conceptual Ontology, but if that same ontology is paired with an application capable of reasoning on OWL documents, such as Pellet (Sirin *et al.*, in press) or FaCT++ (Tsarkov and Horrocks, 2006), and a monitoring system is built around that reasoning engine then it also functions as an Operational Ontology. This would of course still allow the spin-off concept to work as others could build on the OWL Conceptual Ontology in order to create an Operational Ontology that works for their environment.

3.4. Transformation of Encoded Knowledge

One of the aims of classifying the various encodings is to create a method to move between them in some organized manner. By grouping individual representation styles into categories, the chain of transformation steps can be abstracted. While this does not provide information on how to transform a

[11] http:// www.ghg.net/clips/CLIPS.html Last Accessed August 23, 2007

specific encoding to another specific encoding, it does provide some general insight into how such a transformation could be achieved. For example, to transform from a Data Description Language to a Conceptual Ontology will typically require a change in the structure and organization of the data, and potentially a change in syntax as well. A conversion from a Conceptual Ontology to an Operation Ontology will typically only require a change in syntax as the organization of the information should remain the same with such a conversion. So while this abstraction is not necessary to create a transformation chain it is certainly useful in guiding its development. Varro and Pataricza (2003) demonstrate how this type of abstraction can be useful in not only guiding transformation routine development, but also how the transformation routines can be automatically generated through the use of very high levels of abstraction. While this thesis does not strive to achieve this level of automation, it does demonstrate that automation of conversion between representations without the loss of semantics is possible, and that the semantics of the information can actually be reinforced and even increased (see Section 3.11).

3.5. Other Components

While not actually data encodings, there are other pieces of the transformation chain which must be considered. These are Data Sources, Applications, and Services. Data sources are the suppliers of the data that feed the transformation engine. In the case of a monitoring environment they may feed live or archived sensor data, results from simulations, landscape models, GIS data, or any other source of data that can be used as input to the analysis routines. They supply data that represent some real or simulated condition or phenomenon in the field. Applications are the target software tools that will make use of the data after they are converted. This is typically a decision support system of some form built to use ontological data. They have facilities to load and use the information stored in the Operational Ontology and often manage the various transformation steps as well. Services are the tools which assist in moving the data from one stage of the chain to the next. They typically take the form of either a conversion tool or a data access tool. For example, a conversion tool may take database records (a Data Description Language) and convert them into OWL instances that conform to a Conceptual Ontology (Zellwegger, 2005). A data access tool would be used to retrieve the database records and pass them on to the conversion tool (likely through the use of SQL or some other database access tool).

3.6. Target Architecture

A system built to monitor sensor data must have, at a minimum, access to sensor data and some analysis capability. These analysis methods may be simple data-driven algorithms or more knowledge-based approaches. The data that drive the analysis may be stored locally, however this limits the reuse of the data and the long-term archival possibilities. Database systems are typically used to store these types of data, and typical desktop computing environments used for analysis are rarely equipped to handle anything beyond small- to medium-scale databases efficiently. When dealing with anywhere from tens of thousands to even millions of records, multiple concurrent connections, and potentially sensitive data that should be stored in a secure manner, a large-scale distributed database system should be used. A database server (a computer or group of computers dedicated solely to storing and serving the contents of the database) is an ideal choice for this situation. In our case a central server to handle all of the sensor measurements, making them accessible to client applications such as expert systems, is what is needed. Several commercial and free solutions exist to serve database records. The most well known include Microsoft Access[12], Oracle[13], MySQL[14], IBM's DB2[15], and PostgreSQL[16]. In choosing a database implementation, we must also consider that the data being served are spatial in nature, and that this poses special challenges. Databases such as Oracle, PostgreSQL, and MySQL have realized this and developed spatial additions to their database offerings. As an added benefit PostgreSQL, some versions of MySQL, and their spatial extensions are freely available. The decision was made to use PostgreSQL as the database implementation for a few reasons, mostly dealing with ease of setup and configuration of an SOS server to serve the sensor observations (see Chapter 5 for more information on this). A Linux distribution called HostGIS Linux[17] is available that comes preconfigured with Apache and Apache Tomcat for web services, PostgreSQL with the PostGIS[18] spatial extension for building spatial databases, as well as other common spatial technologies such as MapServer[19] and GRASS[20]. All of the applications chosen for the server side are free as well as cross-platform, meaning they could be used in any Linux distribution as well as on Windows to create a free SOS server. Many

[12] http://office.microsoft.com/access Last Accessed August 20, 2007
[13] http://www.oracle.com/ Last Accessed August 20, 2007
[14] http://www.mysql.com/ Last Accessed August 20, 2007
[15] http://www.ibm.com/db2 Last Accessed August 20, 2007
[16] http://www.postgresql.org Last Accessed August 20, 2007
[17] http://www.hostgis.com/linux Last Accessed August 23, 2007
[18] http://postgis.refractions.net Last Accessed August 20, 2007
[19] http://mapserver.gis.umn.edu Last Accessed August 20, 2007
[20] http://grass.itc.it Last Accessed August 20, 2007

are also open-source, meaning they may be customized as needed through modification of their source code.

The intended general architecture surrounding the database server is shown in Figure 3. To enhance usability, several helper applications may exist on the client side and the server side to aid in the discovery, access, and manipulation of the sensor data for various purposes. For our purposes, the helper tools have been limited to the client side to keep the server as strictly a means to publish and retrieve sensor data and descriptions, allowing those with no need for the extra helper tools to bypass those steps.

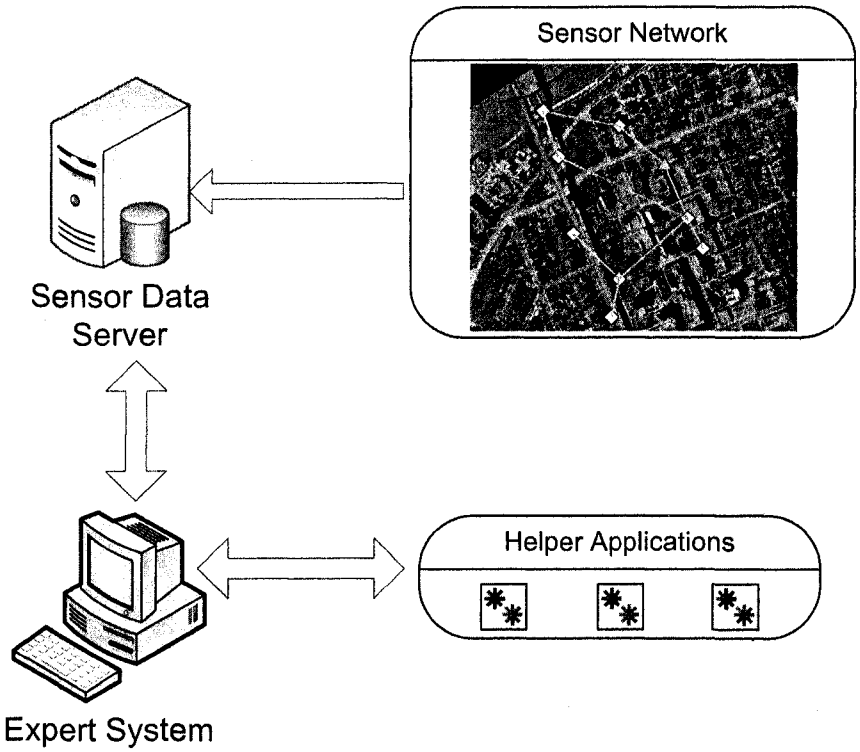

Figure 3 - General Monitoring Environment Architecture

Based on the general architecture, specific choices needed to be made about implementation standards and software for various components. In making these decisions, two design goals were followed. First, existing monitoring workflows should be minimally affected. It is necessary to meld this work into existing monitoring infrastructures, allowing these infrastructures to continue to be used in their traditional manner. This is done using an integration of tools and standards that are commonly

applied to sensor web implementations and infrastructures. Second, automation should occur for as many procedures as possible. This is because those working with the data will typically be well informed in the field being monitored, and are not likely to be well versed in knowledge representation and transformation routines. The time of the domain expert should be spent exploring problems and working on other monitoring tasks, not doing document conversion and validation. The automated conversion frees the expert user to spend their time analyzing the problem at hand.

3.7. Classification Rationale and Generalized Transformation

The classification described was created based on a pattern that emerged when examining various architectural options for the improved monitoring environment. The initial alternatives can be seen in Figure 4. As the alternatives were evaluated, an implicit classification scheme started to appear. It became clear that regardless of the implementation standards, the number of transformations and the style of transformations would stay the same. The general transformation chain can be seen in Figure 5 where the arrows represent the services used to move between steps. By chaining the various levels of representation together using transformations we can move from the data-centric to the information-centric viewpoint on using these data. This chain is intended to be independent of the languages or representations chosen for each phase. This classification can be seen implicitly in several cases, and most standards and encodings can be classified according to this scheme.

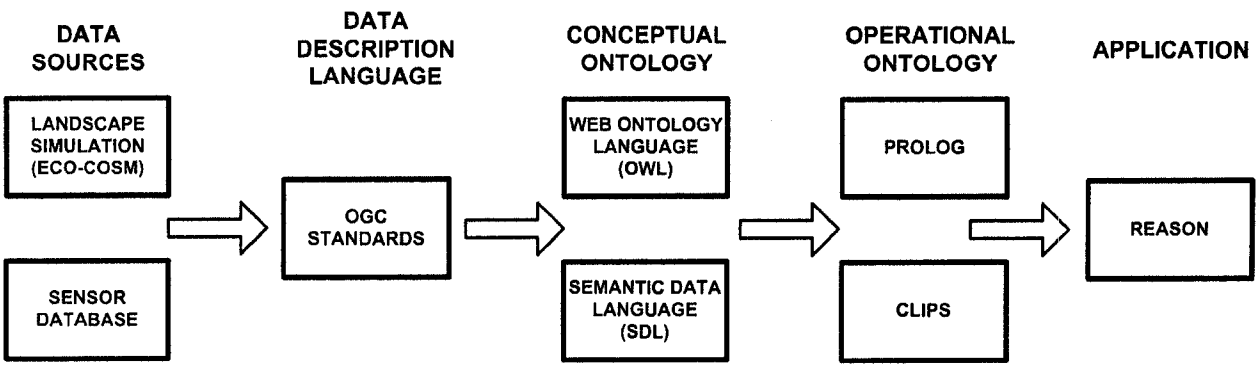

Figure 4 - Initial Monitoring System Design Alternatives

Figure 5 - Generalized Transformation Steps to Move from data to information

The ArcHydro system (Maidment, 2002) is an example that fits this classification system. ArcHydro is a combination of a data model and an associated extension for ESRI's ArcGIS. The extension provides a suite of tools that can be used to investigate and work with hydrologic data. The data that are used with these tools must be structured in a certain way to allow the tools to 'understand' the structure and content of the data. To help achieve this ArcHydro also supplies tools to impose this structure onto existing hydrologic databases. The typical workflow for an ArcHydro project is to take an existing database of hydrologic features, apply the data model, and then do analysis. This is analogous to the transformation chain presented in this thesis. The initial database plays the role of the Data Source, while the database schema acts as the Data Description Language, supplying the individual features that will be used for analysis purposes. Once the ArcHydro tools (a Service) have been used to apply the ArcHydro data model to the database we now have a database that contains domain-specific information and relationships, making the jump from a Data Description Language (the relational database model, the database schema, and any associated GIS metadata) to an Ontology (the newly formatted database). The ArcHydro data model acts as the Conceptual Ontology while the database itself is the Operational Ontology that allows the domain-specific analysis tools to be used on the information within them.

As shown in Figures 4 and 5, a general transformation process is the target with replaceable processes to target specific representations at each stage. For the purpose of demonstration in the scope of this thesis, one representation was chosen for each stage. To determine the appropriate path for the transformations to take, it was necessary to see how each transformation would be achieved and what the benefits and detriments of each step would be. Once that information was recorded, the appropriate transformation path would be clearer. The first decision that had to be made was the target application that would be used for the monitoring environment, as the representations chosen must ultimately lead to an encoding that is compatible with this environment. The REASON engine (Rozic, 2006) has proved successful in the past for the types of monitoring problems that this engine wished to examine (Hutchinson *et al.*, 2007; McCarthy *et al.*, 2007). REASON requires that CLIPS is used

for representation of the final (operational) ontology and sensor data, and since CLIPS is not a well-known language, this may become a barrier for some uses of the system. Alternatively, Prolog is a better known rule-based programming environment, and is more likely to be familiar to a developer. The disadvantage to Prolog in this scenario, however, was the lack of an existing decision support engine to feed the information to. Ultimately the familiarity of Prolog was sacrificed in order to reuse an existing decision support engine that had proved useful in the past. This meant that CLIPS would be used as our operational ontology representation, and that REASON would be used as the application. The service for bringing the CLIPS data into REASON is ArcAgents (Ball and Harrap, described in Rozic 2006).

Once CLIPS was chosen as the operational ontology representation, the next step was to determine if OWL or SDL (Semantic Data Language) would be used as the conceptual ontology representation. Here, the advantages of OWL clearly outweighed those of SDL. OWL is a well-known ontology implementation language, meaning that those who wished to use the data in their own monitoring application that is not based on CLIPS would have an easier time integrating the data into their workflow using a reasoner such as Pellet or FaCT++. It would be easier for them to find support tools and conversion tools to enable this transformation, and if a tool didn't exist it would be easier to develop since OWL is XML-based and there are a plethora of XML parsers in existence. SDL was developed more specifically for landscapes and spatial data, so in that regard it is slightly better suited, but its lack of exposure makes it less likely that someone will want to use the data. Also, SDL works with a concept compiler that takes the SDL statements and produces corresponding Prolog code. Since CLIPS was our target operational ontology language a new concept compiler would need to be written to output CLIPS code. Alternatively, the open-source Protégé ontology editor has the ability to work with both CLIPS code and OWL documents, so it was an easier task to automate this conversion using Protégé libraries than to build a concept compiler from scratch.

There were no data description language alternatives to the OGC encodings that were strongly considered since these encodings represented exactly what was needed from a data description language. The OGC Sensor Web Enablement (SWE) suite of standards is quickly becoming the de facto standard for geospatial and sensor web data, making it a good target for wide use. Further, they are geared towards the representation of sensor data and sensor descriptions, something that was very

desirable since those are precisely the data we wished to represent. The transformation from the OGC encodings into OWL is an XML to XML transformation, a transformation paradigm that is well supported.

Since either a landscape simulation or a sensor database could be used to generate the OGC documents, the choice of which to use was made based on ease of implementation. Since an existing SOS Server implementation is freely available from 52° North, it was used to implement the sensor database. It would also be possible to generate the SensorML and/or O&M documents from the ECO-COSM (Graniero and Robinson, 2006) simulation framework; however this would have been more time-consuming since appropriate data-drivers would need to be written, and example simulations models created. It would be very beneficial for this functionality to be added to ECO-COSM as it would allow landscape simulations to be tied directly into this workflow with almost no modification upstream.

3.8. Document Conversion Chain Implementation

To execute this chain of transformations, a series of services must be used to move between the steps. The services are indicated in Figure 6 as callout boxes. The retrieval and storage of the sensor data and sensor descriptions are done through a Sensor Observation Service database utilizing the predefined messages that an SOS provides to store and retrieve sensor descriptions and observations. The perspectives on how an SOS functions from both the sensor data producer and consumer perspective are shown in Figure 7. The sensor data producer must register its sensors with an SOS and make their details available. It can then insert observations into the SOS that correspond to the sensors that have been registered. The sensor data consumer uses whatever methods are made available by the SOS to discover the data stored within, and based on that it can retrieve sensor metadata and observations and use them as needed.

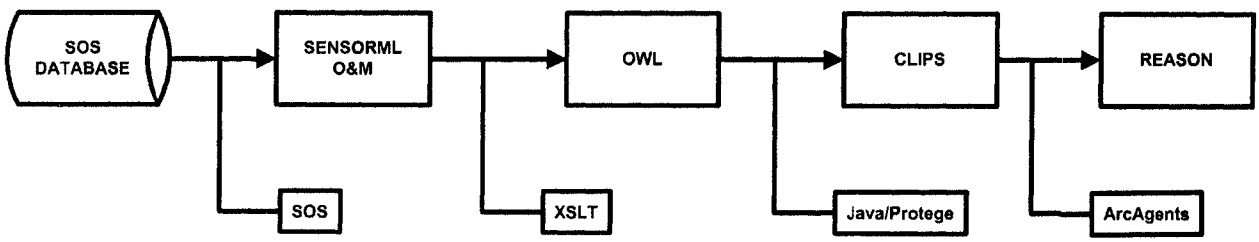

Figure 6 - Transformation steps to move from SOS data to CLIPS data

This architecture is used to provide the SensorML and O&M documents from the SOS to begin the conversion process. These documents are then converted into OWL and aligned to the ontology developed specifically for this engine (see Chapter 4). When the data are aligned with the ontology it can then be understood by a reasoning system. At first glance, it appears that the use of OWL as an intermediate step is excessive and without merit. However, to move from a basic data encoding such as those provided by the OGC into an ontological structure which is ready to be reasoned upon requires a great deal of change to both the structure and the content of the documents. Converting the documents to OWL first allows our documents to take on an ontological structure while still remaining in an XML-based syntax. Since most (but not all) concepts in OWL have an equivalent in CLIPS, the task of converting from one syntax (XML) to another (a CLIPS knowledge base) is much simpler since the restructuring has already been handled by the first conversion. There are some concepts in OWL that the version of CLIPS ArcAgents is based on does not support, such as constraints on the cardinality and ranges of relationships. The use of OWL validation lets us know that our CLIPS code will conform to our ontology even though the constraints cannot be explicitly enforced in CLIPS.

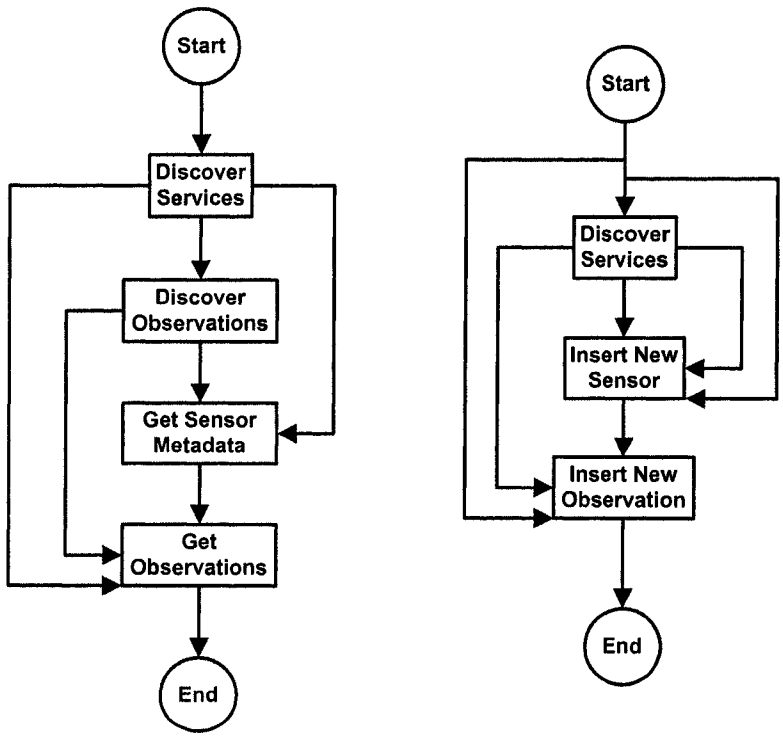

Figure 7 - SOS Operations Perspective for Sensor Data Consumer (Left) and Producer (Right) – Reproduced from Na and Priest (2006)

This two-step process gives more opportunity in a generalized workflow to move between a Data Description Language and an Operational Ontology that have considerably more difficult translations than from an SOS to OWL and then CLIPS.

The specifics of how each Service is used to perform each conversion are detailed in the following sections. The conversions were implemented as separate tools, all of which can be controlled and chained together from the monitoring environment. This automated transformation ontologizes the base OGC data and enables reasoning to be done on them.

3.9. SensorML/O&M to OWL

To move from strictly data-centric encodings such as SensorML and O&M into an information-centric encoding such as OWL, we must map the concepts in our data-centric encodings to concepts in our ontology. Once the mapping from one encoding to another has been conceptualized, it can then be formalized through the development of a conversion tool.

When moving from one XML-based encoding to another, there are typically two technologies which are used. The first is XQuery[21], which is an SQL-like language used primarily to navigate an XML document and extracting pertinent information. The second is XSLT (eXtensible Stylesheet Language Transformation)[22]. XSLT was created initially to change XML into XHTML, though it has since expanded its usage to any general XML-to-XML conversions. Both technologies were considered, however XSLT was deemed more appropriate for this task since it is better suited to convert entire XML documents.

Conversion of documents using XSLT is based on the use of templates, an idea which draws on the roots of XSLT as a language to render XML data in a format suitable for web browsers. Templates are matched against the various elements of a source document and, depending on what those elements are, appropriate action is taken. In the case of converting the OGC-based encodings to OWL, templates were created to match the various concepts defined in the specifications of the OGC encodings regarding sensors and their measurements that output the OWL code which corresponded to those concepts. In general, XSLT templates take the following form:

[21] http://www.w3.org/TR/xquery/ Last Accessed August 20, 2007
[22] http://www.w3.org/TR/xslt/ Last Accessed August 20, 2007

```
<xsl:template name="templateName" match="xpath-expression">
  ...output...
</xsl:template>
```

Listing 1 - XSLT Template Structure

Each template has associated with it a name attribute and/or a match attribute. The name attribute allows other templates to call the template by name. The match attribute takes an XPath[23] expression, which is a standard way of navigating the various nodes of an XML document. Templates which make use of the match attribute will search the source document for nodes which match the expression given. If a matching node is found, the template is called and executed. If not, then the template is ignored. The use of the name attribute is useful when we have a clearly defined set of steps which must be followed in the stylesheet, allowing these templates to be called under specific circumstances, much the same way that methods and functions are used in typical programming languages. However, the problem of converting these sensor descriptions and observations into OWL is much more dynamic, and since the way in which the document is handled depends largely on its contents, we must make use of the matching functionality provided by XSLT.

Listing 2 shows an example template that takes what is typically the root element of a SensorML document (System) and converts it into a concept in the ontology. Most directives in the template are prefixed with a namespace. This namespace prefix indicates what namespace the directive comes from, and thus how it should be interpreted. In this template, lines prefixed with *xsl:* come from the XSLT namespace and tell the XSLT processor, Saxon[24], to treat the directive as an XSLT command. The *memf:* prefix is used to indicate the statements that will be printed to the output OWL document as all statements in the OWL ontology are in the *memf* namespace, representing those created by MEMF Lab, our research group. The *SensML:* prefix is used to match against the input document and to find SensorML concepts in that document.

[23] http://www.w3.org/TR/xpath/ Last Accessed August 20, 2007
[24] http://saxon.sourceforge.net/ Last Accessed August 27, 2007

```
<xsl:template name="System" match="SensML:System">
    <xsl:choose>
        <xsl:when test="current()//SensML:positions">
            <memf:Station>
                <xsl:for-each select="current()/*">
                    <xsl:apply-templates select="current()"/>
                </xsl:for-each>
            </memf:Station>
        </xsl:when>
        <xsl:otherwise>
            <memf:Sensor>
                <xsl:for-each select="current()/*">
                    <xsl:apply-templates select="current()"/>
                </xsl:for-each>
            </memf:Sensor>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Listing 2 - Sample Conversion Template

The first line of the template starts the template and defines the two ways in which a template may be called: by name and by match pattern. The name is used when other templates wish to call this template like a function. The template is also called when a node in the input document matches the match pattern. In this case the template will be called when a *<System>* tag in the *SensML* namespace is encountered in the input document. The next line is an *<xsl:choose>* statement, similar to a switch or case statement in common programming languages. This statement directs the XSLT processor to look at the choices it presents and evaluate them in order. If the first choice evaluates to being true, it is executed and the remaining statements in the choose block are ignored. If the first statement is not true then the rest of the statements are evaluated the same way until a true statement is found. In XSLT, this is done using the *<xsl:when>* directive. The <xsl:*when*> directive shown tells the XSLT processor to execute its containing code if a given test (*current()//SensML:positions*) is passed. In this case the test is to see if the current node in the input document (the *<System>* node) has a child node of type *<positions>*. If it does, the code in the *xsl:when* block is executed. In the SensorML specification, a System can represent a sensor or an actual measuring station installed in the field. In the sensor ontology that was developed, these are different concepts, and need to be represented as such. They are differentiated by the template by looking for a position. If no position is given in the input document, then it is interpreted as a Sensor, whereas if a position is given it is represented as a Station, meaning that the sensor is installed at a given location in the field. So when the SensorML document

contains a statement that gives the position of the System, the code within the *when* directive is executed. First the statement *<memf:Station>* is printed to the output document, then the XSLT processor is told that for each child node of the *<System>* node (the current node), apply the templates to that node, and if a match is found the template will be executed. The results of the execution will be output within the *<memf:Station>* block. This process continues recursively until there are no more nodes left to match on. At that point, the *</memf:Station>* statement is printed to the OWL document to signify the end of that particular Station. Because nodes are addressed in the order they appear in the input document, any nodes that are children to the *<System>* node will be handled and have their output printed within the *<memf:Station>* block.

If a situation arises where the *<System>* node does not contain a child node describing its position, then the *<xsl:when>* statement fails and the next alternative is tested. In this case there are no *<xsl:when>* statements left to test. When this happens, two choices are possible. The first is that the *<xsl:choose>* statement is terminated. The second occurs if an *<xsl:otherwise>* element has been defined at the end of the *<xsl:choose>* statement. The *<xsl:otherwise>* element defines what actions should be taken if none of the *<xsl:when>* tests evaluate to true. In this template, if the single *<xsl:when>* statement fails, then the *<xsl:otherwise>* statement directs the XSLT processor to print *<memf:Sensor>* to the output OWL document and continue searching the child nodes in the same manner as the *<xsl:when>* block would. What this ultimately means is that the template examined the contents of the input document and, based on the contents of that document (the existence of a *<System>* tag and the possible existence of a *<positions>* tag), took an action (printing a statement to the output document). This is the structure that the vast majority of the templates took, though many are simpler than this as there is no decision logic involved. This particular template shows that not only is the information restructured as it is converted from the data-centric to the information-centric perspective, it can also be enriched and clarified so that the meaning associated with the data is clearer when they are consumed.

Initially, this conversion step was divided into two separate tools: a SensorML-to-OWL converter which would handle sensor descriptions, and an O&M-to-OWL converter to handle sensor observations. The concepts in SensorML and O&M were mapped to those found in the ontology, and templates were created which echoed this mapping. It was quickly discovered, however, that there were several concepts that were used by both SensorML and O&M. For example, both SensorML and O&M use the

Geography Markup Language (GML) (Cox *et al.*, 2004) to describe basic geographic concepts such as position and time. Also, both SensorML and O&M make use of common encodings for various data types which are managed by the OGC's Sensor Web Enablement Working Group (Botts *et al.*, 2006). Because of the potential for overlap and duplication of work, it was decided that instead of two conversion utilities, a single conversion utility which handled not only SensorML and O&M data but also many of the shared encodings such as GML and SWE was needed. This conversion tool was created using a series of XSLT stylesheets, each focusing on a given encoding. These stylesheets are then imported into a master stylesheet which is the starting point for any transformation.

The stylesheets for the different standards are implemented to varying degrees of detail. O&M, SensorML, and SWE are all relatively well detailed, the key concepts of space and time have been implemented from GML, and the XST (an XML schema that handles set theory) and SA (an XML schema that handles sampling) encodings have one or two common concepts implemented.

The conversion tool makes great use of the nested nature of XML in deciding how to proceed with its conversion task. It essentially walks the tree of the input document in a recursive manner, starting with the root node. This is made possible by the ability to match based on patterns. Almost the entire engine is written using these templates, with the exception of the master stylesheet (shown in Listing 3).

This stylesheet begins by defining its namespace prefixes and the output method to be used by the document (in this case, the stylesheet will output XML). Following that is a series of *<xsl:include>* statements which tells the XSLT processor to include all of the templates from the indicated stylesheets into this transformation, effectively creating one large stylesheet from several smaller ones. The benefit to this approach is that it is modular, and could be modified to only include the stylesheets that are needed for the conversion or to add additional stylesheets without interfering with the existing logic. After the include directives, a single template is used to launch the conversion. This template matches on the first node (the root node, indicated by the *"/"*) of the input document. It then writes several statements to the top of the new document. The first set indicate that this new document will be a Resource Description Framework (RDF) document (OWL is actually an extension of RDF) and specifies the resource that can be used to see what the document is intending to store. The second set indicates that the document is to import an OWL document located at the indicated path and that this should

serve as the base ontology for the resulting document. The imported document contains the ontology, and the result of the conversion will be instances of that ontology.

When combined this will represent the complete knowledge base with respect to the given sensor(s) or observation(s). The final directive tells the XSLT processor to begin matching templates by using the first node it sees (indicated by the *"*"* parameter), this node will typically be a *<System>* node for a SensorML document or some type of *Observation* or *Measurement*, or a collection of these in the case of an O&M document. Once the first node is matched, that node's template will match on its child nodes recursively until all nodes have been examined. The conversion results in an OWL document that contains ontology instances.

```
<xsl:stylesheet version="2.0" xmlns:SensML="http://www.opengis.net/sensorML"
xmlns:memf="http://matrix.memf.uwindsor.ca/ont/memf/" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:fn="http://www.w3.org/2005/xpath-functions"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:udt="http://matrix.memf.uwindsor.ca/ont/udt#">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

    <xsl:include href="sml2owl.xslt"/>
    <xsl:include href="swe2owl.xslt"/>
    <xsl:include href="om2owl.xslt"/>
    <xsl:include href="gml2owl.xslt"/>
    <xsl:include href="sa2owl.xslt"/>
    <xsl:include href="xst2owl.xslt"/>

    <xsl:template match="/">
        <rdf:RDF>
            <rdf:Description rdf:about="http://matrix.memf.uwindsor.ca/ont/SensorWeb.owl">
                <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Ontology"/>
            </rdf:Description>
            <owl:Ontology rdf:about="">
                <owl:imports rdf:resource="http://matrix.memf.uwindsor.ca/ont/SensorWeb.owl"/>
            </owl:Ontology>
            <xsl:apply-templates select="*"/>
        </rdf:RDF>
    </xsl:template>
</xsl:stylesheet>
```

Listing 3 - Master Stylesheet

3.10. OWL TO CLIPS

Once an OWL document is created and validated, it is possible to convert it into CLIPS code, making it ready for our reasoning system. This conversion was automated through the use of Protégé[25], an open source tool developed at Stanford University which is used for the development of ontologies. Protégé stores its ontologies in a file format modelled after CLIPS, though with a few small differences. Protégé also provides support for OWL ontologies, and can export them into its CLIPS-like format. Since the Protégé project is open source, a Java program was created for this thesis which would load an OWL document into Protégé and export it into the CLIPS-style format. Once the documents are created, some post-processing is applied that makes the documents conform to the CLIPS syntax, making it easier to load the files into our reasoning engine using ArcAgents, as well as to make them more easily readable.

The CLIPS Object Oriented Language (COOL) is used to store the ontology and its associated instances. The object-oriented approach blends well with the hierarchical approach used in the ontology. The concepts in the ontology are represented as classes, and the relationships are represented as slots of the classes. The facets of the slots are represented using CLIPS constructs which place restrictions on the slots. Listing 4 shows two classes from the ontology represented as CLIPS code. CLIPS represents all of its statements such as facts, rules, and classes within brackets. An opening parenthesis signifies the beginning of a statement, and a matching closing parenthesis signifies the end of that statement. Statements are typically nested within other statements. Both of the examples in Listing 4 are classes, as signified by the *defclass* keyword. Following that is the name of the class and a series of statements about that class. For example, the *memf:Sensor* class has three types of statements which describe it. The first statement is *(is-a memf:Instrument)*. This is how inheritance is specified in CLIPS, meaning that any member of the *memf:Sensor* class is also a member of the *memf:Instrument* class (defined elsewhere). Another way to describe this is to say that a *Sensor* "inherits" its attributes from an *Instrument*, meaning that a Sensor is a specific kind of Instrument that has all of the properties an Instrument has, plus some of its own properties. The *Station* class contains a similar statement, *(is-a memf:Sensor)*, stating that a *Station* inherits attributes from a *Sensor*, or that a Station is a specific type of Sensor. Inheritance propagates from one class to the next in CLIPS, so since a *Station* is a *Sensor*, and a *Sensor* is an *Instrument*, therefore a *Station* is also an *Instrument*.

[25] http://protege.stanford.edu

```
(defclass memf:Sensor
    (is-a memf:Instrument)
    (role concrete)
    (multislot rdf:type
        (type SYMBOL)
        (create-accessor read-write))
    (multislot memf:hasConnections
        (type INSTANCE)
        (allowed-classes memf:Connections)
        (create-accessor read-write))
    (multislot memf:hasClassification
        (type INSTANCE)
        (allowed-classes memf:Classification)
        (create-accessor read-write))
    (multislot memf:isFeatureOfInterestOf
        (type INSTANCE)
        (allowed-classes memf:Result)
        (create-accessor read-write))
    (multislot memf:hasProcesses
        (type INSTANCE)
        (allowed-classes memf:Processes)
        (create-accessor read-write)))

(defclass memf:Station
    (is-a memf:Sensor)
    (role concrete)
    (multislot rdf:type
        (type SYMBOL)
        (create-accessor read-write))
    (multislot memf:hasSensor
        (type INSTANCE)
        (allowed-classes memf:Sensor)
        (create-accessor read-write))
    (multislot memf:hasSamplePosition
        (type INSTANCE)
        (allowed-classes memf:SamplePosition
        (create-accessor read-write))
    (multislot memf:hostsProcedure
        (type INSTANCE)
        (allowed-classes memf:Procedure)
        (create-accessor read-write))
    (multislot memf:id
        (type STRING)
        (create-accessor read-write)))
```

Listing 4 - CLIPS representation of Sensor and Station classes

The next statement about both classes is *(role concrete)*. This specifies that the classes are concrete as opposed to abstract. A concrete class is one which is able to have instances (i.e. specific usable objects that conform to the class definition), while an abstract class cannot have instances. Aside from some high-level classes, all of the classes in the sensor ontology are concrete. The remaining statements that describe the classes take the form of multislots. In CLIPS, a slot is an attribute that a class can have. For instance, a class "Student" would have slots such as "Name", "Age", and "Grade". In the case of the sensor ontology, these slots represent the relationships that the class can have with other classes. For example, the *Station* class has the slot *hasSensor*. This is the name of a relationship defined in the sensor ontology which specifies that a Station has a Sensor associated with it. The term multislot simply means that the slot can hold more than one value of the specified type. This slot also contains statements that describe it. These are equivalent to the facets of the ontology. The *hasSensor* relationship has two facets. The first, *(type INSTANCE)*, specifies that the value of this slot must be an instance of some class as opposed to a primitive data type such as a string or an integer. The second, *(allowed-classes memf:Sensor)*, specifies that the instance must be of type *memf:Sensor*. This facet helps validate the contents of the ontology, ensuring that it is structured properly. If something other than a *Sensor* is placed in this slot the ontology will be marked as invalid upon loading. The final

assertion about the *hasSensor* slot is the statement *(create-accessor read-write)*. This statement tells CLIPS to define message-handlers that can set or retrieve the value of the slot during program execution. In this way all of the relationships and classes in the OWL ontology are defined and organized.

This class hierarchy definition is used to give structure to the instances which represent the actual objects of interest, in this case sensors and observations. It describes how sensors and observations are structured, what kind of properties they have, and what the relationships are between them. The instances of the hierarchy will represent individual sensors and observations that are governed by this hierarchy. Listing 5 shows an example of a Station instance as part of the CLIPS make-instance function.

```
(make-instance [@1179243727343_:A0] of  memf:Station

    (memf:hasClassification [@1179243727343_:A11])
    (memf:hasConnections [@1179243727343_:A73])
    (memf:hasIdentification [@1179243727343_:A1])
    (memf:hasInputs [@1179243727343_:A24])
    (memf:hasOutputs [@1179243727343_:A28])
    (memf:hasProcesses [@1179243727343_:A35])
    (memf:hasReferenceFrame [@1179243727343_:A21])
    (memf:hasSamplePosition [@1179243727343_:A71])
    (memf:hostsProcedure [@1179243727343_:A74])
    (rdf:type memf:Station))
```

Listing 5 - An instance of the Station class

When this function is loaded, the instance will be created and added to the knowledge base. This funciton will direct CLIPS to create an instance of the class memf:Station. The [@1179243727343_:A0] label is a unique name that is used internally to differentiate and keep track of individual instances. CLIPS provides functions to work with these data so that the user never has to handle these labels. The remaining lines specify the slots that are defined for this Station, each of which are filled with an instance of the appropriate class. For example, the (memf:hasSamplePosition [@1179243727343_:A71]) statement indicates that this station has a SamplePosition that is specified by the instance with the label [@1179243727343_:A71]. This instance is shown in Listing 6 with its own slots that define the coordinate frame, units of measurement, and the coordinates of the Station.

For a typical project, hundreds of instances of various classes are created to represent sensors and observations as well as their properties. The relationships between the various instances are

automatically generated and maintained because of the class-based ontology that provides the context for the information contained in the instances.

```
(make-instance [@1179243727343_:A71] of  memf:SamplePosition

    (memf:hasLocalCoordinateFrame [@1179243727343_:A72])
    (memf:x "934.538")
    (memf:XUOM "urn:ogc:def:uom:OGC:1.0.30:m")
    (memf:y "81.102")
    (memf:YUOM "urn:ogc:def:uom:OGC:1.0.30:m")
    (rdf:type memf:SamplePosition))
```

Listing 6 - SamplePosition CLIPS Instance

CLIPS provides several functions which are intended to work on COOL structures, which enables the navigation and usage of these data and ensures that the data will be usable by an expert system. The potential usage of the data is broad and is explained and illustrated throughout Chapters 5 and 6.

3.11. Semantic Repair

Not only does this ontological structure allow us to validate our results, it also allows us to perform some tests on the semantic validity of the data; we can then ensure that pertinent information is expressed in the data. This can be done in a step that detects "semantic errors" and repairs the semantics of the document, providing an extra layer of quality control on the document that simple validation does not. This repair step requires reasoning on the incoming data along with related data and the ontology's conceptual structure to try and detect any inconsistencies or problems that may arise. Whenever changes in representations like these are undertaken there is a possibility of data loss or inconsistency (Gannod and Cheng, 1999). To help detect when this has happened we can validate the data against the ontology, but we can also use the reasoning process to examine the data and look for semantic errors beyond those enforced by the ontology. For example, suppose we have an O&M document representing a set of observations made by a sensor as a time series, and that one of the measurements is missing a time stamp. This may be legal according to the ontology (which specifies that a measurement *can* have a time, not that it *must* have a time), but not having a time associated with the measurement makes the measurement far less useful to a real-time monitoring problem. However, with the use of reasoning tools this problem can be ameliorated. A reasoning tool can be used to flag these kinds of errors and force the user to deal with them or at least inform them of the

problem. Depending on the severity of the error the reasoner could decide whether the appropriate course of action is to flag the error, hold it in a cache until it has been dealt with, or try to fix it based on other available data. Using the time series example, the reasoner may be able to infer what the missing time stamp is based on the interval of other similar measurements. It may look at recent measurements from that sensor and see what the measurement interval has been and make an estimate based on any gaps in the measurement record. Alternatively it could simply look at the sequence of the measurements and note the time stamps of the measurements made before and after the affected record and then mark that the measurement was made between those two times. This can be done either by setting a specific value, or instance the midpoint of the time range, or by setting a range bounded by the times on the measurements found before and after. In this way reasoning can be used to repair and enrich the content of the data by detecting errors of omission and commission as well as other more complex and application-specific problems.

3.12. Summary

This chapter has shown how the use of targeted knowledge representation standards for specific tasks can ensure that the meaning of information can be captured in a form that enables reasoning and knowledge-based analysis of sensor data. This is achieved through a carefully constructed set of transformations that take data-centric representations of data such as basic XML structures and database records and transform them into an ontological structure for use within a knowledge-oriented analysis system. The next chapter will explain how the ontology used to provide the context for the information was developed and presents a methodology that can be used for other similar attempts to build an ontology.

4. Ontology Development

4.1. Methodology

Development of a thorough sensor ontology to be used within REASON was fundamental in increasing the reasoning capabilities of the system. The sensor ontology built in the initial system expressed the most basic concepts that were necessary to operate and demonstrate the system. The new sensor ontology expresses these concepts along with much more contextual information about how a sensor behaves, the various properties it has, how it makes measurements, what types of measurements it makes, and how all of these concepts are related. This allows the reasoning system (or an expert user) to have much more knowledge about how their particular sensor network is operating, providing context to the measurements it produces.

Many methodologies for the development of domain-specific ontologies exist (e.g. Sure *et al.*, 2002; Lambrix *et al.*, 2003). There are also methodologies that have been developed that are less domain-specific (e.g. Lopez *et al.*, 1999). Based on a hybrid approach from Mizen *et al.* (2005) (Steps 1, 2, 3, 5, 9, and 10 as shown below) and Noy and McGuinness (2001) (Steps 1, 2, 4, 6, 8, 11 as shown below), as well as some additional steps, the following methodology was developed to create the sensor ontology. Phase 1 details creation of the "pencil and paper" version of the ontology, and Phase 2 details the formal computer implementation of that ontology.

Phase 1 of Ontology Development Process

1) *Create a set of competency questions.* These questions place demands on the ontology. They represent the requirements that the ontology needs to satisfy. These requirements are represented as questions that can, and likely will, be asked of the ontology. (Gruninger and Fox, 1995)
2) *Scope the ontology.* Determine the domain, purpose, and potential users of the ontology. This scope should be kept in mind during the entire creation process.
3) *Collect data about the domain.* Identify any documentation that captures the knowledge that needs to be in the ontology (keeping in mind the scope).
4) *Enumerate important terms.* Compile a list of all terms and sentences we would like to either make statements about or explain to a user. For all terms consider what properties they have and what needs to be said about them.

5) *Populate a knowledge glossary.* Using the list of important terms and the semi-structured sentences, build a knowledge glossary. The knowledge glossary should capture the reasoning behind the selection of the various terms, the properties of those terms, and any description of the terms or assumptions that are made about them.

6) *Examine existing ontologies.* Explore other ontologies in the domain to see if they meet the requirements defined in the previous steps, or if they can be extended to do so. If so, decide if it is possible to reuse and/or extend the existing ontologies.

7) *Choose language(s) of implementation.* Note any restrictions the chosen language(s) may place on the ontology's design. While the ontology is initially created using a "pencil and paper" style, the representation used to implement the operational ontology should be decided on before the creation of the actual ontology. Ensure that any restrictions placed on the ontology by the chosen language are acceptable both now and in the future of the ontology, and insure that it can represent any information the ontology may need.

8) *Build the ontology.*

- Define classes and class hierarchy: use the knowledge glossary to find terms which describe objects having independent existence rather than terms that describe these objects. These terms become classes in the ontology and will become anchors in the class hierarchy. Organize the classes into a hierarchical taxonomy by asking if, by being an instance of one class, the object will necessarily be an instance of some other class.
- Define the properties of the classes (slots): keep in mind that there are different types of properties, such as intrinsic properties, extrinsic properties, parts, and relationships to other individuals. The terms remaining in the knowledge glossary after the previous step (defining classes) are likely to be properties.
- Define the facets of the slots: Slots can have different facets describing the value type, allowed values, the number of the values (cardinality), and other features of or restrictions on a slot's value.

9) *Evaluate the conceptual ontology.* Check whether all information captured in the glossary has been captured in the ontology. Check the ontology for: logical consistency (cycles, repetition, omission); conceptual accuracy (with respect to the domain); minimal ontological commitment (ontology has been limited to the original scope); information loss being recorded; and acceptable answers to competency questions.

10) *Document the conceptual model.* Conceptual ontology documentation must include the knowledge glossary (from Step 5), the concept and relationship networks (from Step 8), recorded information loss, and any defined rules and assumptions made throughout the modelling process.

11) *Create instances to test the ontology.* Instances should be created in order to test the ontology and its effectiveness. This will ensure that all information that was intended for representation can actually be represented.

Phase 2 of Ontology Development Process

12) *Implement the ontology in the chosen language.* The ontology should be implemented using the language(s) identified in Step 7 to assure that the design does indeed work for the given language.

13) *Use instances to test the implemented ontology.* Ensure that the implemented ontology can handle the instances created in Step 11. If changes are needed to the ontology, they should be performed iteratively. Evaluation should occur after each iteration, until the ontology satisfies all requirements laid out in previous steps. This step is only finished when these requirements are met or a decision is made to ignore those requirements.

14) *Document the ontology.* Ensure documentation is written completely and clearly to enable reuse. Perform a final evaluation cycle on the implemented ontology and assure that the results match those from Step 11. Document any changes in scope, requirements, or any other pertinent information.

4.2. Development

This section will explain how each of the steps from Section 4.1 was carried out during the development of the sensor ontology. It also details what the results of the steps were, any problems that were encountered, and any other information that may be relevant to those who want to apply this methodology to their own ontology development problem. For the purposes of brevity, the complete results of each step have been made available on the companion CD for this thesis.

4.2.1 Competency Questions

A set of competency questions was created which would be used to evaluate the ontology at the end of the design cycle. These questions are essentially tests that the ontology must pass in order to be

considered useful for our purposes. The questions are broken down into two groups, those regarding the evaluation of the ontology and its expressiveness, and those related to more domain-specific needs.

Ontology Evaluation Questions

- Is there a strong distinction between sensors and objects which are not sensors?
- Is there a distinction between sensors and transducers?
- Are the terms used for classes and properties the same as those that would be used by experts in the field?
- Do the relationship names accurately describe the relationships between objects?
- Does the ontology sufficiently describe the domain?
- Can any redundancy be aggregated?

Domain-specific Questions

- Can a sensor's suitability for a certain purpose be determined with the ontology?
- Can improper use of a sensor be detected?
- Can any sensor be represented as an instance of the ontology without losing any key information?

4.2.2 Scope

The scope of the ontology encompasses four factors: the purpose of the ontology, a definition of what an ontology is so that it can be used to guide development, the domain the ontology will be used in, and an idea of who will use and maintain the ontology. The intended purpose of this ontology is to provide the information necessary for high-level reasoning on geotechnical (and other) sensor data to be performed by a spatial decision support system. The intended domain of use is slope hazard monitoring with geotechnical sensors, but should be broad enough to encompass all in situ sensors. The ontology will be used by geotechnical engineers who wish to monitor slopes and maintained by domain experts and those well versed in knowledge representation. For the purposes of this development effort an ontology will be defined as *a specification of the important concepts and relationships between these concepts within a particular domain.*

4.2.3 Collect Data about Domain

The data collected about the domain were collected from the OGC's SensorML and Observations and Measurements specifications. As these specifications are intended to specify details of sensors and their measurements, they provided excellent operational descriptions of these topics. Dunnicliff (1993) was used to gather information on geotechnical instrumentation, including the concepts of how geotechnical instrumentation is utilized in typical monitoring scenarios, and the common types of geotechnical sensors and transducers.

4.2.4 Enumerate Important Terms

Based on the information collected in Step 3, a list of almost 600 terms was created. These terms were chosen because they represent the important concepts and relationships that needed to be represented in the ontology. The documentation resources were scoured for relevant terms and concepts, building a foundation for the knowledge that would need to be captured in the ontology. At this point in the development process it was important to capture all knowledge which may be relevant. The later steps in the ontology development process would be used to pare down the knowledge to the most important concepts and to eliminate any duplication in concepts.

4.2.5 Populate Knowledge Glossary

From the list generated in the previous step, a knowledge glossary was created. This glossary is used to enumerate and organize the terms from the list into a consistent format. The information recorded about each term was a list of synonyms that were found in the original list or any other synonyms that may have been missed; a natural language definition such as one that would be found in a dictionary; the part of speech of the term (noun, phrase, or verb); the anticipated usage in the ontology (concept, relationship, or characteristic); the importance of the term (core/secondary); characteristics that the term may possess (for core concepts only); the value and units associated with the term (if applicable); and any rules, constraints, or assumptions that needed to be recorded. This glossary formed the basis of the first draft of the ontology.

A subset of this glossary can be seen in Figure 8. The glossary was organized into a spreadsheet and was set up with filters to aid in rapid visualization of relevant data. This step may take several days to complete based on the size of the ontology and the level of detail required. This glossary will typically

contain more information than the final ontology will as similar concepts are merged and less important concepts are dropped. The glossary provides the first organized view of all of the collected information.

Term	Synonym Term	Natural Language Text Definition	Linguistic Term	Conceptual Ontology Term	Core/Secondary	Core Concept Characteristics	Value and Units	Rules, Constraints, and Assumptio
Sensor	Geotechnical instrument (for the purposes of the ontology; "sensor" is typically a more broad term)	A physical device capable of measuring a specific phenomenon, consists of a transducer, data acquisition system, and a communication system between the two	Noun	Concept	Core	is a device, has reliability, has accuracy, has type, has target phenomenon, has conformance, has precision, has measurement, has resolution, has sensitivity, has transducer, has input, has output, has linearity, has maximum error, has hysteresis, has suitability, has data acquisition system, has communication system	Specific to individual sensor types	
Precision	Reliability, Repeatability	The extent to which a measuring procedure yields the same results on repeated trials	Noun	Characteristic	Secondary		± units of measurment	
Accuracy	Error	The closeness of a measurement to the true value of the quantity measured	Noun	Characteristic	Secondary		± units of measurment	
Measurement	Observation	A value describing a certain phenomenon obtained through measuring methods	Noun	Concept	Core	has unit of measurement, has value, has location, has time	units and value vary based on sensor type	
Sensor Type		The class of the instrument, representing its intended use	Noun	Characteristic	Secondary			

Figure 8 - A subset of the knowledge glossary

4.2.6 Examine Existing Ontologies

Now that there is some idea of the information that needs to be represented in the ontology, a thorough examination of existing ontologies can be done. Before this step, exploring the existing ontologies could only have resulted in broad comparisons. Once the knowledge glossary is populated, though, a detailed comparison can be done to see if any existing ontologies meet the needs defined by the competency questions and the knowledge glossary. An existing ontology developed for sensor observations (Probst *et al.*, 2006) was used to provide a base for the new ontology. The decision was made to create the sensor portion of the ontology from scratch based on the SensorML specification and later merge it with the aforementioned sensor observation ontology that was based on Observations and Measurements. Other sensor ontologies were considered, such as OntoSensor

(Russomanno *et al.*, 2005), however merging two existing ontologies developed in a disjoint manner and for different purposes would have been difficult and much of the ontology would likely have been rewritten.

4.2.7 Choose Implementation Languages

For this ontology, several choices of ontology implementation languages were explored. Also, because of the need for multiple representation styles for various stages of knowledge (see Chapter 3), it was important to identify multiple possibilities for the various stages in the transformation. CLIPS was chosen as the operational ontology implementation language because the target application built in REASON expects CLIPS code. The conceptual ontology was built in OWL instead of the Semantic Data Language because OWL is a more common way of representing ontological information. Also, the conversion between OWL and CLIPS could be automated using Protégé, and since similar tools exist to convert OWL into other formats it became clear that OWL would be the best language to enable sharing of data. This also meant that only one conversion tool would need to be built from scratch. The species of OWL chosen was OWL-DL. OWL-DL is the middle-tier of the three OWL species, the other two are OWL Lite and OWL Full. OWL Lite is a very basic version of OWL that allows minimal expressiveness but also minimal computational complexity. OWL Full allows for maximum expressivity, however there is no guarantee that an OWL Full ontology will be computationally complete or decidable. This implies that queries on an OWL Full document may not return results in a reasonable amount of time, or at all. OWL-DL is a blend between the other two species. It guarantees that the ontology will be computationally complete and decidable, however it must place some minimal restrictions on the ontology to make this claim. In return there is more computational complexity in OWL-DL ontologies than in OWL Lite ontologies. In this case the restrictions would not limit the ontology in any way, and so OWL-DL was deemed the most appropriate species of OWL to use.

4.2.8 Build the Ontology

The ontology was built initially using pseudo-UML notation to graph the ontology in Microsoft Visio[26]. The initial version of the ontology focused largely on geotechnical sensor types. As other related work began to develop further, the focus shifted to a more general ontology with clear places to store domain specific knowledge.

[26] http://office.microsoft.com/visio/ Last Accessed August 20, 2007

4.2.9 Evaluate the Conceptual Ontology

After several iterations of refinement, the ontology was deemed to be conceptually accurate and logically consistent. This was based on the examination of the ontology and a thorough comparison with the original knowledge glossary. It was limited to the initial scope of geotechnical sensors, though this scope was later enlarged and generalized.

4.2.10 Document the Conceptual Ontology

All of the documents used in creating the ontology were examined for correctness and accuracy, ensuring they documented the ontology that was created and any deviations from the original design of the ontology. Since the ontology was built in Visio, it was largely self-documented.

4.2.11 Create Instances to Test Conceptual Ontology

Instances of the ontology were created that represented inclinometers and piezometers. The relevant information about these instances was laid out in advance and was then made to fit the ontology. All of the information that was considered relevant was able to be placed in the ontology, and so the tests were considered successful.

4.2.12 Implement the Ontology

The ontology was implemented using the Protégé ontology editor (Figure 9) in the OWL language. Protégé-OWL[27] is a graphical front-end to OWL ontology development that is available as an add-on to the Protégé Frames[28] editor. This editor makes the organization and management of ontologies, especially large ones, much simpler than writing XML. The final ontology had almost 200 classes and 150 relationships, which would have been very difficult to manage strictly using XML. Protégé enabled both ontology-wide refactoring and simple adjustments to be done with a few simple commands. Figure 9 shows Protégé being used to edit a relationship of the Sensor class. This illustrates the ease of modifying classes and relationships across various classes.

Since OWL is based on the Resource Description Framework (RDF), all of the information in the ontologies is represented as triples of subjects, predicates, and objects. The subject is the entity that is

[27] http://protege.stanford.edu/plugins/owl/ Last Accessed August 20, 2007
[28] http://protege.stanford.edu/overview/protege-frames.html Last Accessed August 20, 2007

being described, and the predicate expresses some characteristic that the subject has that is described by a relationship to the object. The OWL approach dictates that the subject is a class, and that predicates are represented as properties of the class. These properties take two forms: Object Properties and Datatype Properties. Object properties relate individuals of one class to individuals of another class. An example of this would be the relation that a member of the class "Teacher" would be related to a member of the class "Student" with the relationship "Teaches", so the triple would be "Teacher-Teaches-Student". The second type of property used in OWL relates an individual of a class to a typed value such as a string or a number. An example of this type of relationship would be that a member of the class "Student" is related to the number "24" using the relationship "Age", so the triple would be "Student-Age-24". The only real difference between Object properties and Datatype properties is the type of value in the object position.

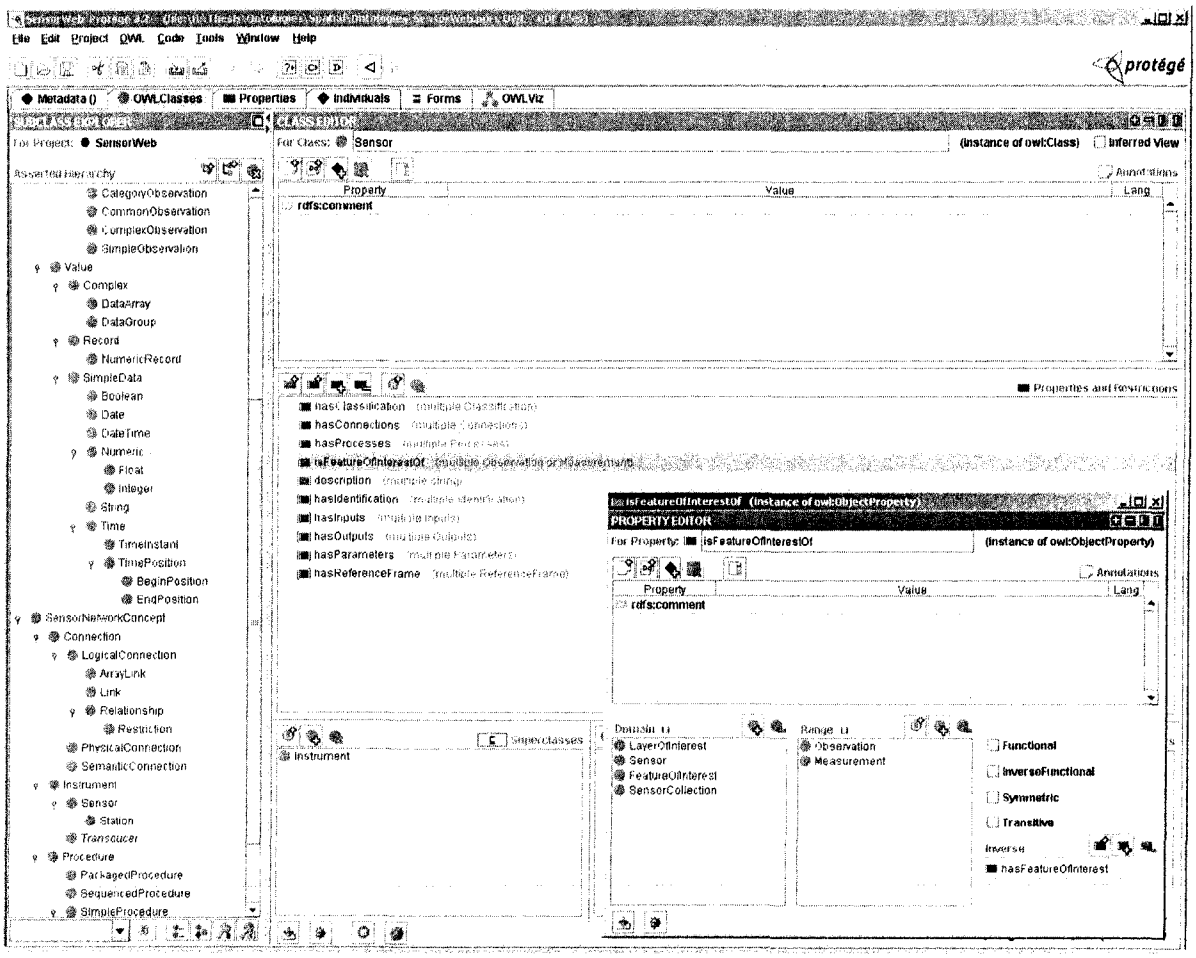

Figure 9 - The Protégé Ontology Editor

OWL properties are not limited to one-to-one relationships. Often a property will have multiple classes allowed in the subject and/or object positions. The terminology OWL uses for this is to say that a given property has a domain and a range. The domain of a property is the group of classes that are allowed to take on the subject position, while the range is the group of classes that may take the object position. This restriction on class types makes it possible to validate an ontology to ensure that the individuals used in the ontology are organized in a legal manner according to the ontology specifications.

4.2.13 Create Instances to Test Implemented Ontology

The instances created in Step 11 were recreated within the Protégé Ontology Editor. All of the important concepts and properties were able to be represented within the implemented ontology. The ontology was passed through an OWL validation application[29] that confirmed that the ontology was indeed a valid OWL document. Further, it confirmed that the ontology was of the OWL-DL species, meaning that it was computationally complete and decidable. This was necessary to prove that the information contained in the ontology could be used to perform reasoning in a practical way.

4.2.14 Document the Ontology

The final step of ontology development was to ensure that all documentation was completely written and that it documented all of the steps that were taken to complete the ontology.

4.3. SUMMARY

The sensor ontology does not focus on sensors from a single problem space. There is room for further domain-specific additions to be made to the ontology, however these are best left to experts in specific fields. The ontology was developed in the Web Ontology Language (OWL), using the Protégé Ontology Editor. The ontology underwent several iterations before being finalized and made web accessible. The management of these versions of the ontology was done using the Concurrent Versions System (CVS)[30]. CVS allows developers of computer code or other documents to track changes and development across multiple versions of their documents, giving them the option of rolling back to previous versions should the need arise. Initially the ontology was created as many pieces, each

[29] http://www.mygrid.org.uk/OWL/Validator Last Accessed August 20, 2007
[30] http://www.nongnu.org/cvs/ Last Accessed August 20, 2007

corresponding to the specifications from which the concepts were drawn. Eventually, these ontologies were aligned and duplicate concepts were merged for simplicity. Dozens of versions of the integrated ontology were created, so management of these versions was essential. The final version of the ontology used to test and integrate with the monitoring system contained 192 classes and 148 relationships.

5. Monitoring Suite and System Integration

A monitoring system that handles spatial data is an essential tool for hazard managers. Being able to analyze incoming real-time data is a must to monitor any hazard which does not provide much lead time. In these cases it is also a good idea to examine similar events that may have occurred and use them to forecast what may occur when familiar conditions are seen. For any system of this type, the more knowledge it can draw on, the more useful it is.

This work was developed to integrate with the REASON overall decision support system framework developed by our research group within the Geotechnical In-Situ Sensor Technology (GIST) Network, a GEOIDE-funded collaborative network. This framework is designed with geotechnical hazard monitoring in mind, but could be applied with only minor changes to monitoring problems in other domains as well. The overall structure of the framework can be seen in Figure 10. The framework provides a guide to the integration of not only the various parts of this system but how those parts will fit into the overall workflow of a geotechnical hazard monitoring problem. This work addresses several aspects of this framework.

The Sensor Observation Service Database provides the Spatial Data in the form of Observations and Measurements Documents and the Instrumentation Sources as SensorML Documents. The transformation engine is used to feed the data into the monitoring system as well as to manage those data. The domain-specific knowledge expressed by the ontology along with the associated measurements provides part of the knowledge base for use by any analysis methods. The GIS Interface and Rule Sets are provided by a Spatial Decision Support System built earlier in the GIST project.

The REASON Spatial Decision Support Framework (Rozic, 2006) is a tool which can be used to develop spatial decision support systems. It was developed using the ArcAgents tool (Ball and Harrap, described in Rozic, 2006) which bridges CLIPS, a programming language geared toward the development of expert systems, and ESRI's ArcGIS. REASON makes use of ontologies to partition the various knowledge it has about a given problem. Ontologies are often used where knowledge definition is a key component of the problem-solving process.

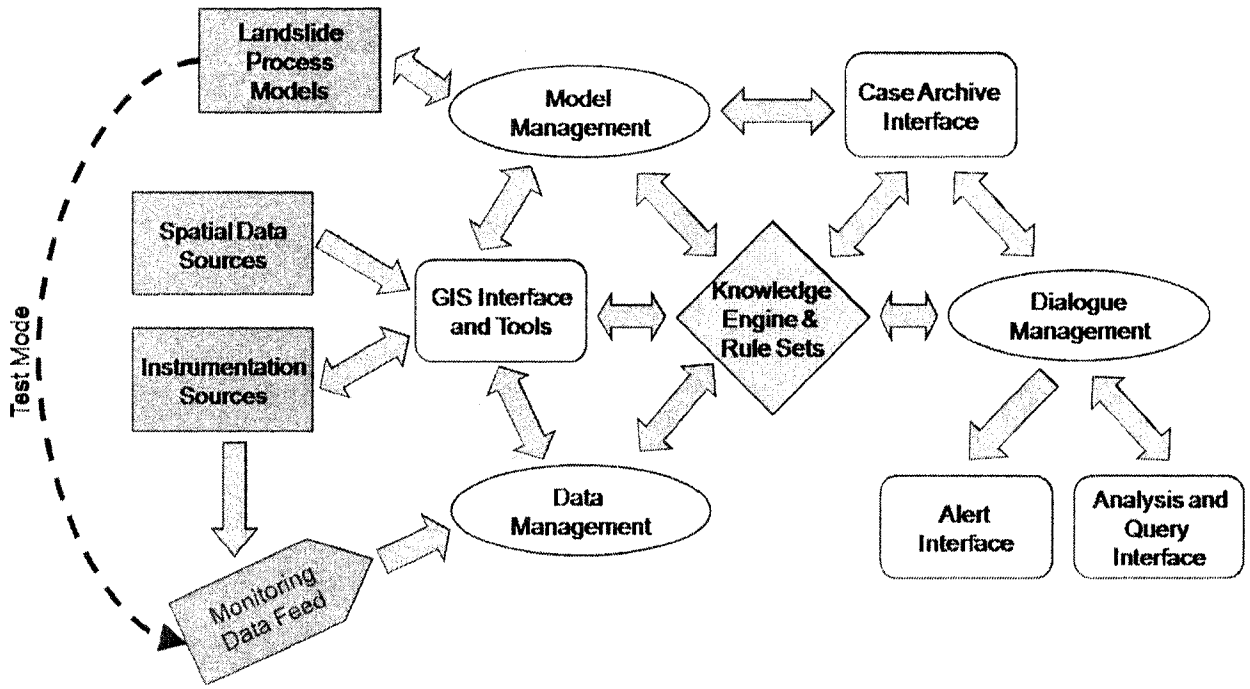

Figure 10 - GIST DSS Conceptual Framework (from Harrap *et al.*, 2006)

The ontological structure used is a variant of the hierarchy proposed by O'Brien and Gahegan (2004), in which there are four separate but related ontologies which are used to contain most of the knowledge required for the operation of the system (see Figure 11). These ontologies contain both facts which describe the various concepts and objects, and rules which govern their behaviour. The "Spatial-Temporal Ontology" is the top-level ontology used to define foundational concepts such as geometrical, topological, and temporal relationships. Two mid-level ontologies build on the concepts from this ontology: the "Domain Ontology" and the "Sensor Ontology". The domain ontology is used to describe the concepts related to the domain being observed, such as the hydrological or geotechnical domains. The sensor ontology describes the sensors which are used to perform the observation. Finally, the bottom level "Application Ontology" contains the concepts related to the execution and capabilities of our given monitoring application, such as flood monitoring or slope monitoring. This includes the decision trees which govern the analysis of incoming sensor data. The application ontology builds on the knowledge from the two mid-level ontologies, and thus from the spatial-temporal ontology as well.

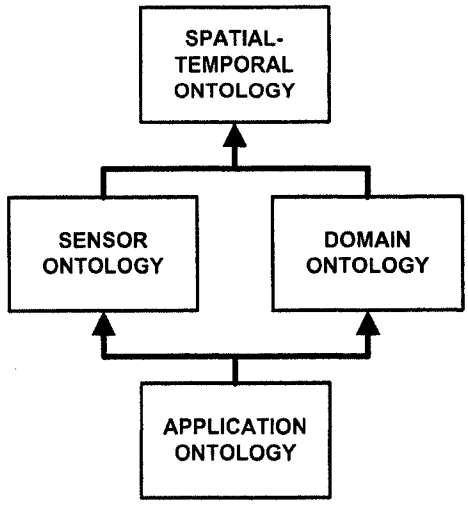

Figure 11 - REASON Ontology Hierarchy

When it comes to spatial data there are a myriad of possible data types, so it is important to be able to use as many as possible in a manner that is transparent to the analysis logic of the system. Some sensor data may be archived, while real-time data may be streamed directly to the system without an intermediate database. The decision of what data are archived and what data are streamed in real-time is made by the designer of the monitoring infrastructure based on the needs of the problem. Likely, the most pertinent data are streamed directly into the system (and also archived for future use) while less imperative data are archived until it is needed. Simulation data may take the form of Excel tables, CSV files, XML files, or some other implementation-specific format. The constraints of the monitoring problem may also dictate how the data are stored. For example, low-power sensor networks will require short messages with minimal transmitted information, whereas higher power, wired networks may be able to transmit messages with more complex structures. Supporting the many variants of spatial data infrastructure is a necessity as we move towards a more interoperable sensor web (Gorman *et al.*, 2005). Therefore, one of the key design features of REASON is that the mechanism to bring data into the system has been abstracted. This abstraction, along with the supporting ontologies, allows various types of data to be used with the system in a common way.

When an SDSS is built upon the REASON framework, it defines the data source(s) it will use and provides an implementation of the abstract DATA-SOURCE class for each type of data source. This defines how the SDSS should connect to and disconnect from the data source, as well as how the data are updated and how the next update cycle is handled. In this way, any data source can be used within

a REASON SDSS. Figure 12 shows the main workflow of the REASON SDSS engine. Once data sources are bound to the objects in our SDSS and to the GIS layers which will be used for any spatial analysis, the objects are updated with new values at every regular time step (or as otherwise defined by the data-source implementation). Evaluation is then carried out on the new values as defined in the application ontology. When evaluation is completed, new values are acquired from the data source and the process repeats itself until the system is told to release the data source resources and terminate.

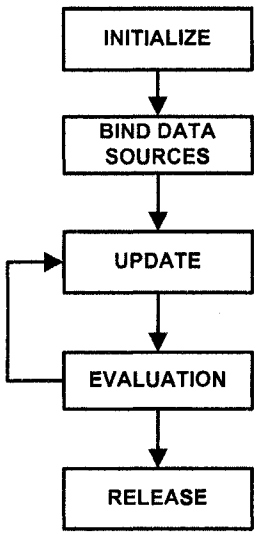

Figure 12 - REASON Evaluation Loop

Since the REASON data-source mechanism is abstracted, observations can be drawn from Excel or database tables to generate facts which correspond to a template-based ontology, or encodings from geospatial standards such as a Sensor Observation Service (discussed in Section 3) can be used to generate instances of an object-oriented ontology, with minimal changes to the actual decision-making logic. The knowledge of the domain is separated from the other knowledge in the system, so creating a monitoring system to work in a different domain only involves changing the domain ontology to one which describes our new domain of interest, and creating a new rule set which governs what we are interested in monitoring. Figure 13 shows the detailed methodology behind the updated version of the REASON slope monitoring system that interacts with the SOS server. It makes use of the abstracted data-source mechanism to connect to an SOS server to retrieve its values while making use of the ontology hierarchy to divide its knowledge.

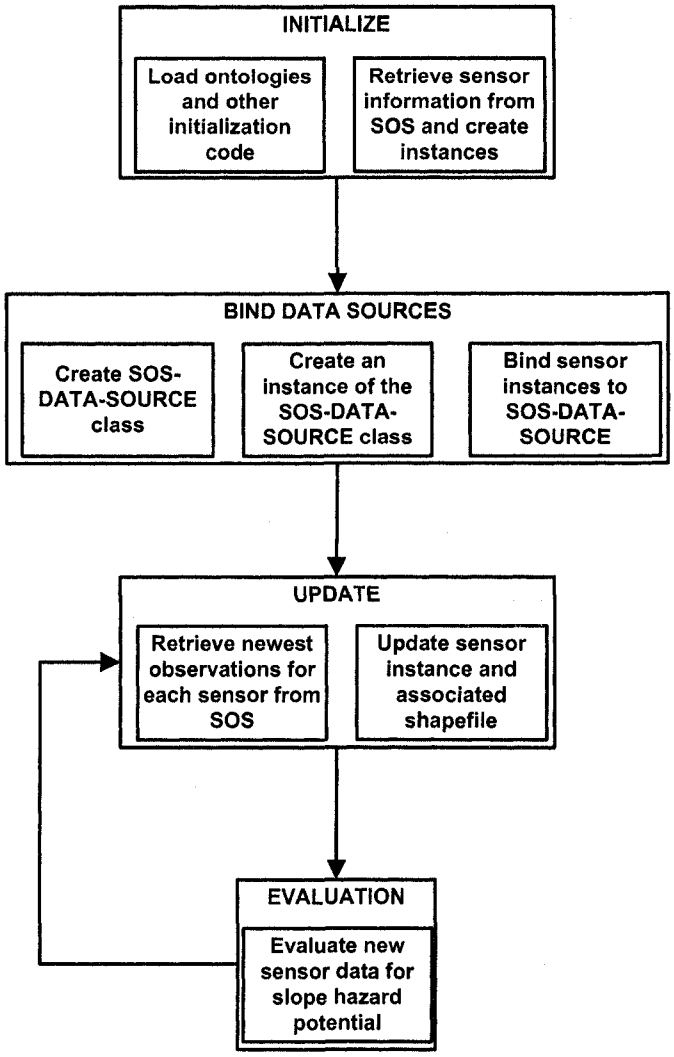

Figure 13 - Expanded version of the REASON evaluation loop

When the system is initialized, the ontologies are loaded into the CLIPS knowledge base. These ontologies contain the majority of the code that will be used to operate the system. A MAIN module is also loaded which initializes the SDSS and controls the evaluation loop. For details on how this is done, see (Rozic, 2006). The main changes to the system have been expansion of the sensor ontology and addition of a new data-source implementation that interacts with an SOS server. The new data source class (Beacon-SOS-DATA-SOURCE) is built dynamically during the binding process. An instance of this class is created that can be called on to retrieve data for any Beacon during the update cycle. To accomplish this, the data-source class actually interacts with an interface object exposed by ArcAgents for use in Visual Basic for Applications (VBA). This allows VBA code to handle the communication with

the SOS server, passing the results of these interactions back to the CLIPS knowledge base where they can be evaluated. Figure 14 shows a sequence diagram of a typical request for a sensor description. Requests for observations are identical in terms of the components and order of the messages, only the messages themselves differ.

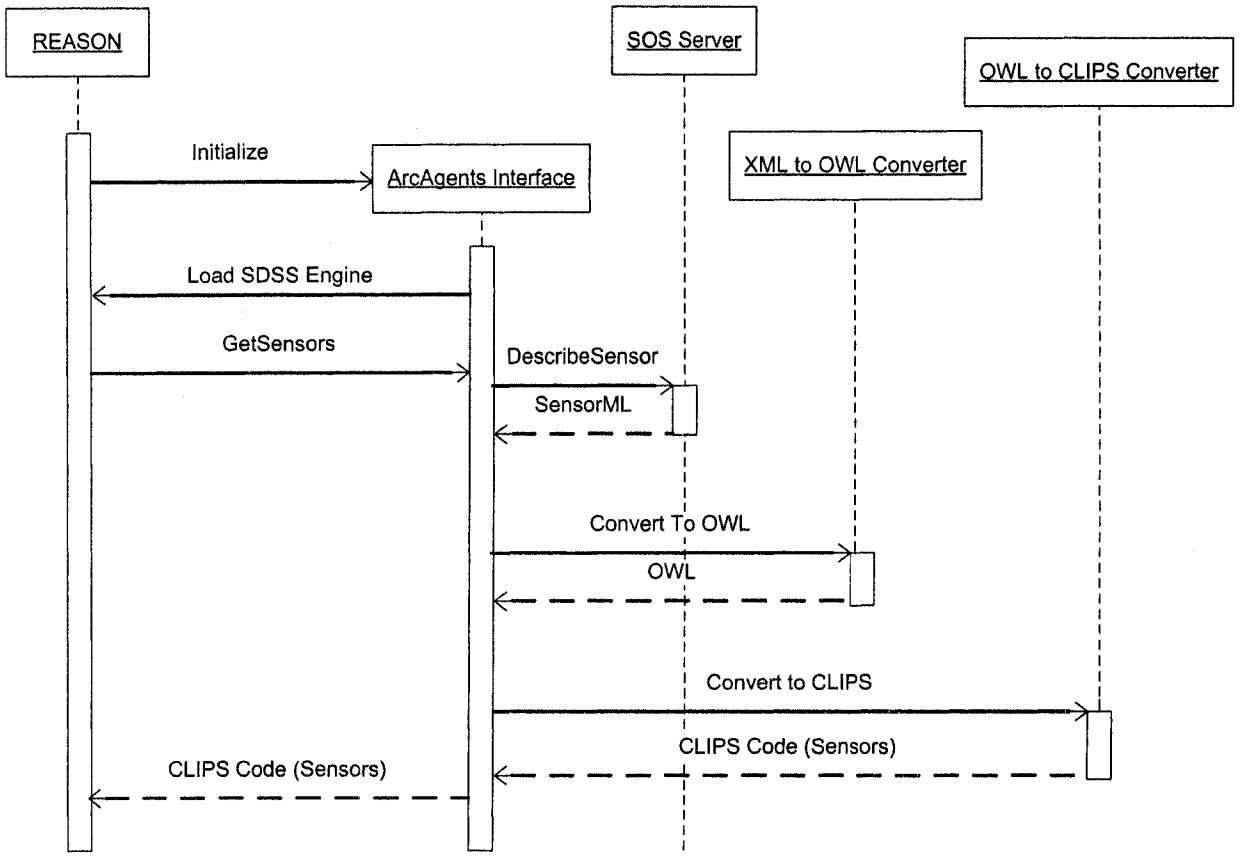

Figure 14 - Sequence diagram for a sensor request

The control of the overall system is coordinated by REASON. REASON initializes the ArcAgents Interface object when a document with the appropriate template is loaded. When the ArcAgents Interface has loaded successfully, it requests that REASON load its SDSS engine. Once the engine is loaded, it can make requests of the ArcAgents Interface object which is persistent as long as REASON is running. For example, it can send a request for sensor descriptions to the ArcAgents Interface that launches a chain of instructions that results in CLIPS code being returned to REASON that describes those sensors. The ArcAgents Interface retrieves the SensorML document from the SOS Server. It then passes that document to the XML to OWL Converter which returns an OWL document. The OWL

document is passed to the OWL to CLIPS converter that returns CLIPS code which is then returned to REASON. The specifics of these operations are described in Chapter 6.

Once the next set of observations is retrieved, they are used to update the GIS layers representing the observations as well as any of the facts in the CLIPS knowledge base that may be relevant. At this point evaluation can be performed on the new data as well as any historical data. Analysis methods are domain- and application- specific but will generally rely on the most recent data and possibly archived data to examine for some predefined conditions that have been determined to be of interest.

5.1. INTEGRATION

The integrated monitoring system closely resembles Figure 3 in terms of overall architecture. Figure 15 shows the complete architecture of the system, broken into three main layers: the servers, the helper tools, and the expert system. The sensor network components were left out of the diagram as they are not the focus of this thesis. Testing was done using simulated sensor data based on a slope failure scenario model (Hutchinson *et al.*, 2007). The sensor data were extracted from Excel spreadsheets and inserted into an SOS server that was built using an implementation developed by the 52° North Initiative[31]. This same server implementation was tested with live hydrologic and temperature sensors, with the results being transparent from the sensor data consumer's perspective. This has enabled our reasoning system to be hooked up to live sensor data. Figure 14 shows a UML sequence diagram of how the various components interact during a given request. These requests are initiated in the SOS-DATA-SOURCE implementation for a given sensor. The abstract data-source mechanism created in the initial system forces any implementation to define three methods: get-data, next-cycle, and close-data-source. The get-data and close-data-source methods are unchanged from their initial implementations; it is only the next-cycle method which needed to be rewritten to work with an SOS server. The next-cycle handler is responsible for fetching the newest set of sensor observations from the data source, in this case a sensor data server. To accomplish this, the handler interacts with an ArcAgents interface object defined in an ArcMap template using Visual Basic for Applications (VBA). The interface is capable of watching certain aspects of the CLIPS engine's functionality and taking action when certain events occur. The key functionality that was used was the interface's ability to listen for facts to be posted and take action when they meet certain criteria. In this case, whenever a fact with a

[31] http://52north.org Last Accessed August 20, 2007

specific name was posted, action could be taken. A set of facts were created which could be used to launch external commands simply by being asserted. The facts were treated as functions, where the name of the fact would act as the function name, and the value(s) of the fact would be the arguments. These "fact-functions" ranged in use from simply dealing with time stamps to initiating database transactions. The simplest of the functions are:

- **(print-text <text>):** prints the contents of <text> to a console, this is used to print important notifications to a separate console from the simple diagnostic and procedural information
- **(first-cycle-time <date>):** sets the timestamp for the first cycle to the date and time specified by <date>, this allows for data that are not real-time in nature to be used
- **(cycle-time-interval <time>):** sets the time interval that the current time would be incremented each cycle

A more complex function was needed to handle the database queries. The (send-SOS-request) fact was created for this purpose. A summary of the fact's functionality is given below. The complete syntax and list of options for this function are given in Appendix B. The format of the fact is **(send-SOS-request <operation> <options>)** where **<operation>** is one of *"obs", "cap", "des", "mro", "sen"*. These are abbreviations for GetObservation, GetCapabilities, DescribeSensor, Most Recent Observation, and Sensors. The first three correspond directly to SOS operations that retrieve observations, SOS server capabilities, and sensor descriptions respectively. The **<options>** section takes different parameters based on the type of operation. For the "obs" function, these options specify the offering, procedure (actually a sensor name), and observed property (phenomenon) that the user wishes to retrieve observations for. These observations can also be filtered with spatial and temporal filters (such as bounding boxes and time ranges), as well as numeric filters on the observations themselves. The "cap" function takes options which specify the specific sections that are needed from the SOS server's capabilities document. The "des" function takes a procedure (actually a sensor name) and returns the SensorML document which describes that procedure. The "mro" function makes a GetObservation request with the time values tailored specifically to retrieving the most recent observation for each sensor. The "sen" function makes a GetCapabilities request which returns only sensor descriptions and details. This information is used to create an ESRI shapefile that contains spatial features representing the sensors provided by a given offering.

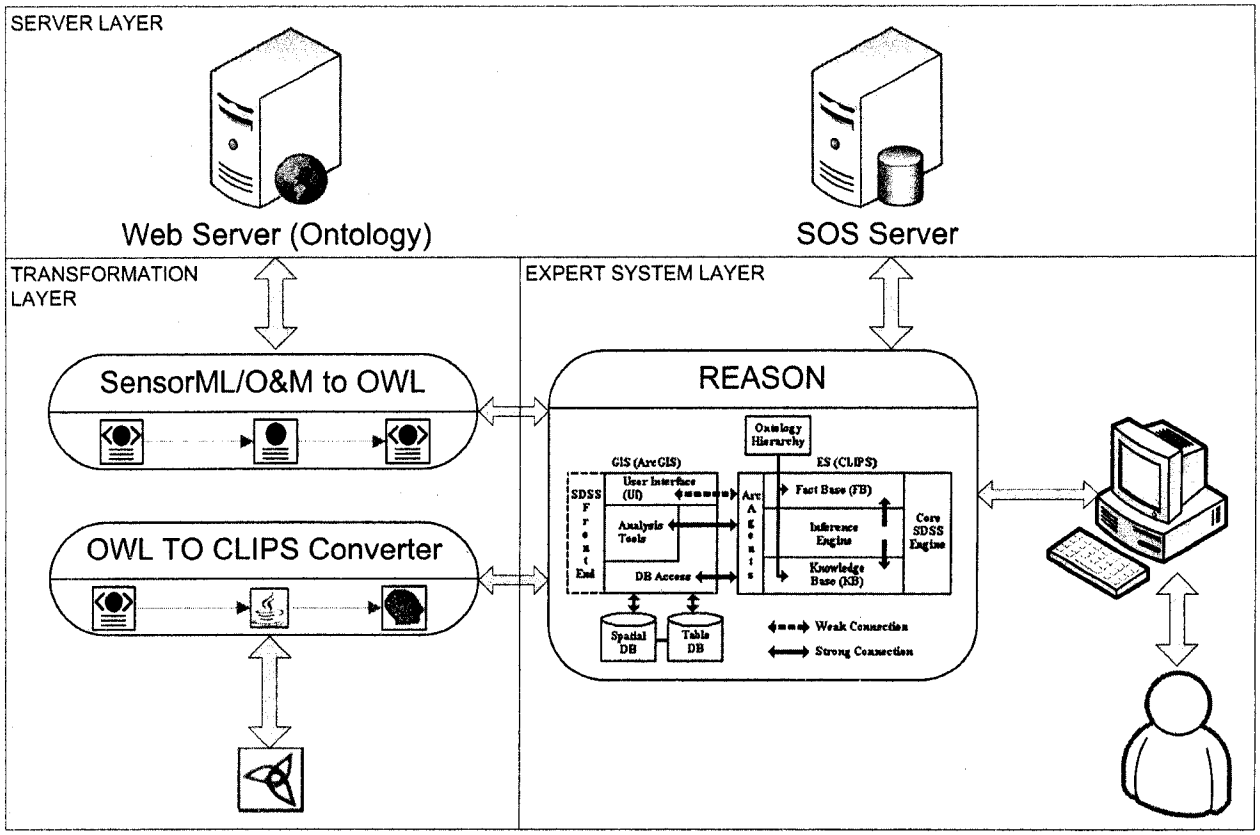

Figure 15 - Integrated Monitoring System Architecture

For all of these operations, the ArcAgents interface object parses the fact string and generates the XML required to make the request of the SOS server. It then makes the request and gets the return value as either a SensorML or O&M document. Once the document is retrieved, it then passes it to the SensorML/O&M to OWL Conversion tool. This tool uses the methods described in Section 5.2 to generate an OWL document. This OWL document is read by the ArcAgents interface object and then passed to the OWL to CLIPS conversion tool. This tool uses the methods described in Section 5.3 to generate CLIPS code which contains the instances of the ontology that represent sensors or observations, depending on the operation requested.

Applying this layered framework to integrate the components required for hazard monitoring using sensor networks provides several benefits. Modularity is achieved through the separation of components based on intended usage, and communication interfaces between these components are structured in a way that would minimize the impact of changing components. For example, using a different SOS implementation will only affect the interaction with REASON (specifically the ArcAgents

Interface Object). Similarly, a different transformation engine could be used in place of the one that was implemented, as it would still have access to the OWL ontology from the web server and the ability to pass its results to the expert system. Having this framework in place before the design and implementation of individual components guides the creation of those components as well as the interfaces between them. This inter-component communication is quite often a difficult problem to overcome when collaboration between system objects is needed. This hurdle is best dealt with at the beginning of design so that the individual components are designed in a way that enables communication between them.

This thesis has so far shown how the various system components have been developed and integrated, and has detailed the impetus for doing so. The next chapter will examine how the system works in practice and how it can be used for geotechnical hazard monitoring on a simulated slope with realistic movement driven by the movement of an associated water table. It will show how the creation of the new components such as the transformation engine aid the decision making process. It is anticipated that the use of a concrete example will better illustrate how this system improves on the current state of hazard monitoring systems.

6. Case Study

This chapter will illustrate how the tools developed during the course of this thesis can be applied to a realistic hazard monitoring problem, as the tools are applied to the problem of slope monitoring. Slope monitoring was chosen as an example case because of the availability of a realistic slope model and associated data. None of the tools that are applied are specific to a given hazard, and as such can be used for any monitoring problem provided that appropriate analysis logic can be created and that data is available to drive the analysis. The problem of slope monitoring makes an interesting case because the onset of the hazard is rapid, and so it is important to detect the conditions that are likely to trigger the hazard rather than the hazard itself. The system directs the hazard manager to investigate areas of concern based on domain knowledge and context-enhanced information delivered by the transformation engine. This system may also be used outside of the hazard management domain. For example monitoring the climatic conditions of vineyards could be done through this infrastructure, and the use of reasoning software could enable analysis methods such as case-based comparisons to previous years' conditions and the use of climate models for forecasting future weather conditions. Another possibility would be to monitor the sediment flow and nutrient flushing in a watershed after a rainfall to explore how these parameters affect the health of the vegetation in the watershed.

The use of REASON as the basis for a spatial decision support system for geotechnical hazard monitoring was initially detailed in Rozic (2006). However additions have been made to the demonstration system to illustrate the enhancements developed over the course of this thesis. The additions related to interaction with the SOS Server, the integration of the new sensor and measurement ontology, and the new data source class to manage these have all been detailed in previous chapters. There have also been enhancements in the decision-making logic as well as the addition of water table data to provide a context for the behaviour of the slope. The original slope model can be seen in Figure 16, showing the positions of the seventy-two inclinometer sensors, the active portions of the slope (shear zones) and the material layers.

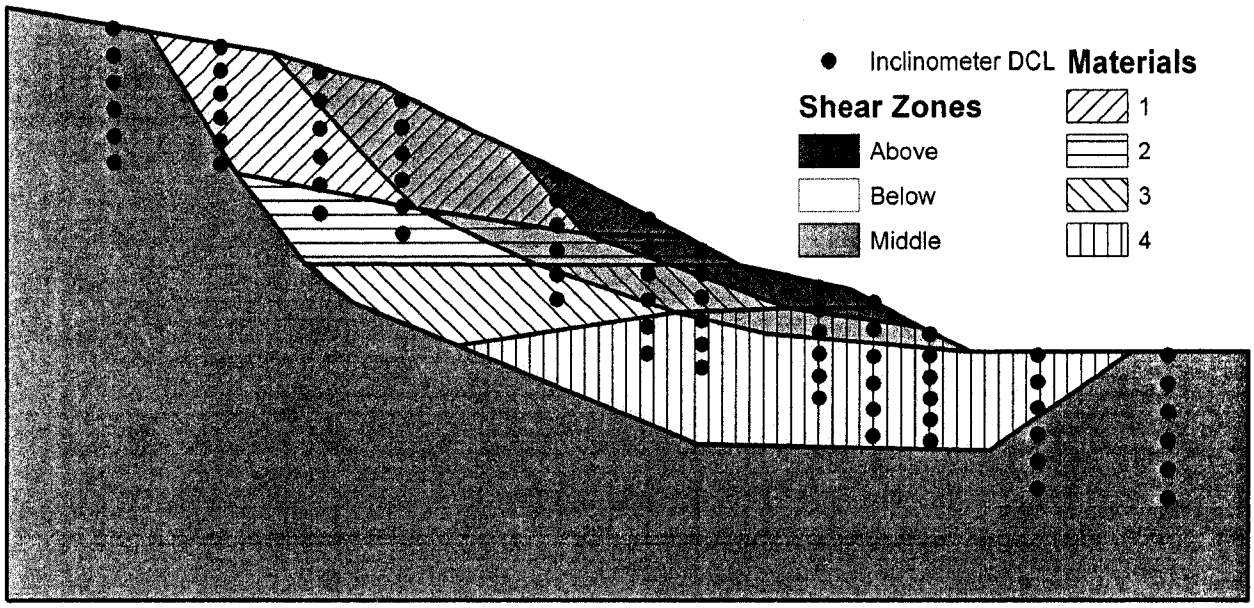

Figure 16 - Initial Slope Model

REASON uses a decision tree structure (Quinlan, 1990) to perform an evaluation at each cycle based on the newest data as well as the data from previous cycles. The decision tree that was used in the initial prototype system can be seen in Figure 17. In this system, the position of each Data Collection Location (or sensor) was used to determine various parameters for the motion of the slope. First, the boreholes are identified based on the horizontal coordinates of the sensor. The sensors with similar horizontal coordinates are grouped together into boreholes. Then each sensor is examined for activity according to a set of rules that define what is considered relevant motion (increasing in displacement with an increment greater than one percent of the current cumulative displacement). This ensures that even though a sensor moves it is not necessarily considered active, since all inclinometers are expected to exhibit some downslope motion. Active sensors are then categorized according to the rock mass to which they belong. Finally, active zones of the slope are determined based on whether or not the zone contains active sensors.

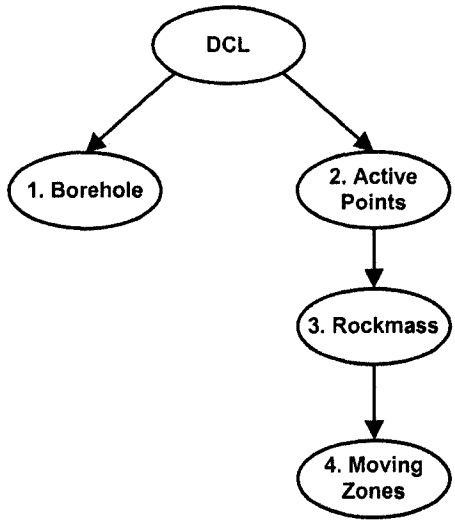

Figure 17 - Initial Geotechnical Monitoring Decision Tree

This simple decision tree illustrated REASON's ability to capture domain knowledge in an SDSS and to aid in the monitoring tasks of an expert user by alerting them to areas of interest. The decision tree is executed after each set of new measurements is retrieved from the SOS server, and the output is printed to a console so that the user may be notified of the results. The control of the system can be seen by examining the next-cycle handler of the new SOS-DATA-SOURCE class that was created for this system. The code for this message-handler is shown in Listing 8. Note that in the source code of the system, this code is written as part of the CLIPS "eval" function that takes a string as an argument and converts it into CLIPS code at runtime. This allows the code to be dynamic in nature so that certain parts of the source code are actually generated at runtime in order to match the parameters of the given monitoring task. The handler is called like a function during each cycle of execution, and is passed three arguments: a template name, a sensor type, and the current cycle. The template name is used to make an association between a template and a shapefile in the GIS environment. This association tells REASON which shapefile should be updated when a given template receives new values. The sensor type indicates what kinds of measurements are expected from the sensor: absolute, cumulative, or incremental. This information ensures that the sensor values are interpreted properly. The final argument tells the handler what cycle is currently executing. The REASON system is based on cycles rather than the system clock, so the actual time of the measurement is not specified here.

```
(defmessage-handler MAIN::Beacon-SOS-DATA-SOURCE next-cycle (?template-name ?sensor-type ?cycle)
        (if (not ?self:layer-logical-name)
         then (send ?self get-table ?template-name)
        )
        (bind ?busy (assert (busy-SOS true)))
        (assert (send-SOS-request "mro" "DOWNSLOPE_MOTION" "urn:ogc:phenomenon:distance"))
        (bind ?pairs (find-all-instances ((?proc-time-pair PROC-TIME-PAIR)) (= 1 1)))
        (acSelectByLayer ?self:layer-logical-name new)
        (bind ?numSelected (acGetNumSelectedByLayer ?self:layer-logical-name))
        (bind $?values (create$))
        (progn$ (?pair ?pairs)
                (bind ?procedure (send ?pair get--procedure))
                (bind ?time (send ?pair get-time))
                (bind ?msr_ins (nth$ 1 (find-instance ((?msr memf:Measurement))
                                                (and
                                                        (= (str-compare (nth$ 1 (send (nth$ 1 (send ?msr get-memf:hasProcedure)) get-
                                                        memf:hasURI)) ?procedure) 0)
                                                        (= (str-compare (nth$ 1 (send (nth$ 1 (send (nth$ 1 (send ?msr get-memf:hasTime)) get-
                                                        memf:hasTimePosition)) get-memf:timePosition)) ?time) 0)
                                                )
                                        )
                                )
                )
                (bind ?whereClause (str-cat Beacon_ID= """ ?procedure """))
                (acSelectByAttribute ?self:layer-logical-name ?whereClause New)
                (bind ?selectedFeature (acGetLayerSelection ?self:layer-logical-name))
                (bind ?feature-fid (nth$ 2 ?selectedFeature))
                (bind ?value (implode$ (create$ (nth$ 1 (explode$ (nth$ 1 (send (nth$ 1 (send ?msr_ins get-memf:hasResult)) get-memf:value)))))))
                (bind $?values $?values (implode$ (create$ ?procedure)) (implode$ (create$ ?feature-fid)) "0" ?value)
                (acClearSelection)
                (send ?pair delete)
                (bind $?values (eval "(" Absolute ?self:layer-logical-name (implode$ $?values) ")" ))

                (bind ?id-one (nth$ 1 $?values))
                (bind ?fid   (nth$ 2 $?values))
                (bind ?X (nth$ 3 $?values))
                (bind ?Y (nth$ 4 $?values))
                (bind ?M (nth$ 5 $?values))
                (bind ?R (nth$ 6 $?values))

                (if (eq ?id-one "done")
                then
                        (return close)
                )

                (bind $?slots (create$))

                (while (> (length$ $?values) 0)
                (send ?self put-values ?id-one ?fid (create$ ?X ?Y ?M ?R))

                (bind $?values (delete$ $?values 1 6))

                (bind ?id-one (nth$ 1 $?values))
                        (bind ?fid   (nth$ 2 $?values))
                        (bind ?X (nth$ 3 $?values))
                                (bind ?Y (nth$ 4 $?values))
                        (bind ?M (nth$ 5 $?values))
                        (bind ?R (nth$ 6 $?values))
                )
                (acClearSelection)
                (acRefreshMap)
                (acSelectByLayer ?self:layer-logical-name new)
                (return more)
                )
)
```

Listing 7 - SOS-DATA-SOURCE Message Handler

The message handler begins its execution by checking if the template has been bound to a layer in the GIS. If it has not, the handler will call a function *(get-table)* that will prompt the user to specify the

shapefile that will be updated. It then asserts the fact *(busy-SOS true)* that indicates to other parts of the system that the SOS is currently being accessed. It then asserts a 'fact-function' (see Chapter 5) that initiates an SOS request *(assert (send-SOS-request "mro" "DOWNSLOPE_MOTION" "urn:ogc:phenomenon:distance"))*. This fact requests the most recent observation from the SOS for the offering *DOWNSLOPE_MOTION* and the *distance* phenomenon. The VBA code in the ArcMap template parses this fact and makes the request of the SOS in the manner described in Chapter 5. The result of the request will be a set of CLIPS objects that are loaded into the knowledge base. These objects are instances of the sensor and measurement ontology, specifically an ObservationCollection instance with several Observation instances (Figure 18).

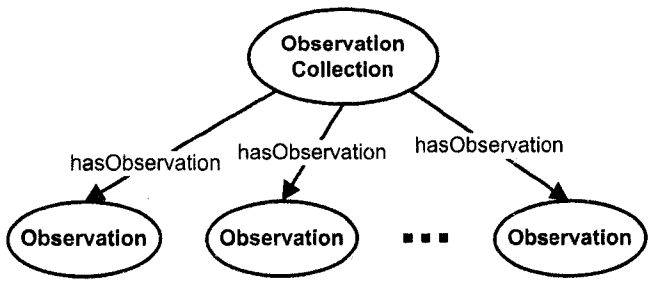

Figure 18 - Observation Collection with associated Observations

To rapidly retrieve this information, the ArcAgents interface object also creates a set of *PROC-TIME-PAIR* objects that pair a sensor name with the updated timestamp of the newest observations. Once the SOS operations are complete, control of the system is passed back to the message-handler which searches for all of these *PROC-TIME-PAIR* objects and stores them in a variable *(?pairs)*. The number of features in the layer is determined by selecting and counting them and an empty multifield value *($?values)* is created to store a list of the updated values that will be retrieved from the Observation instances. The message handler then iterates through the *PROC-TIME-PAIR* objects, retrieving the *Observation* object which has matching procedure and time values (Figure 19). Then, it must match that observation object against the appropriate feature in the associated layer by selecting the feature from that layer that has the same procedure value as the *Observation* currently being examined. Once this match has been made the new sensor value is retrieved from the *Observation* object (Figure 20) and added to the *$?values* variable.

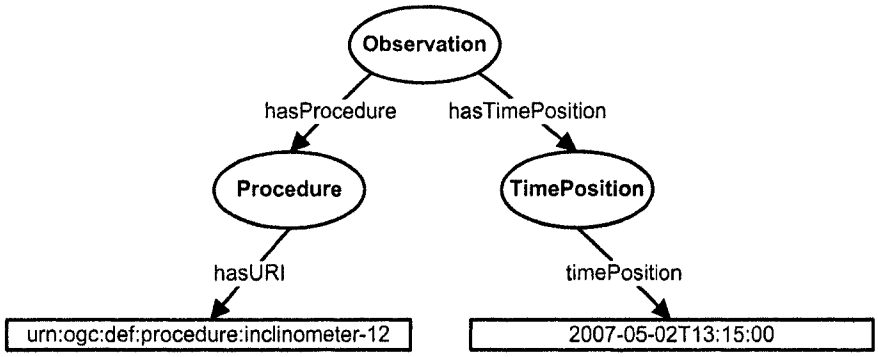

Figure 19 - Observation instance with matching procedure and timePosition

When all of the observations have been matched with their associated features in the layer, the $?values variable will contain a record of all of the updated values. This record is passed to a function that updates the records in the table based on the sensor type (absolute, incremental, cumulative). The data are then used to update various critical parameters of the fact that represents the sensor such as its position and ID values. Some housekeeping tasks are done to make sure all of the fields of the template are up to date, and finally a value of "more" is returned, indicating that this data source has not expired.

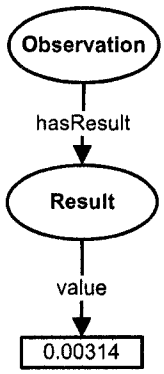

Figure 20 - Observation instance showing a result

The values retrieved from the SOS and updated using the ontology are used by the decision tree to evaluate the state of the slope at the given time step. Once the decision tree has executed the evaluation continues at the next time step and the process repeats itself as long as new data exist that can be accessed by the system.

The original decision tree identifies active areas of the slope at every time step, however it would be beneficial to explore the problem more thoroughly should there be data available to do this. Figure 21 shows an expanded version of the decision tree that was implemented as part of this current thesis to demonstrate the extended REASON system's ability to interact with an SOS server and make use of the expanded sensor ontology

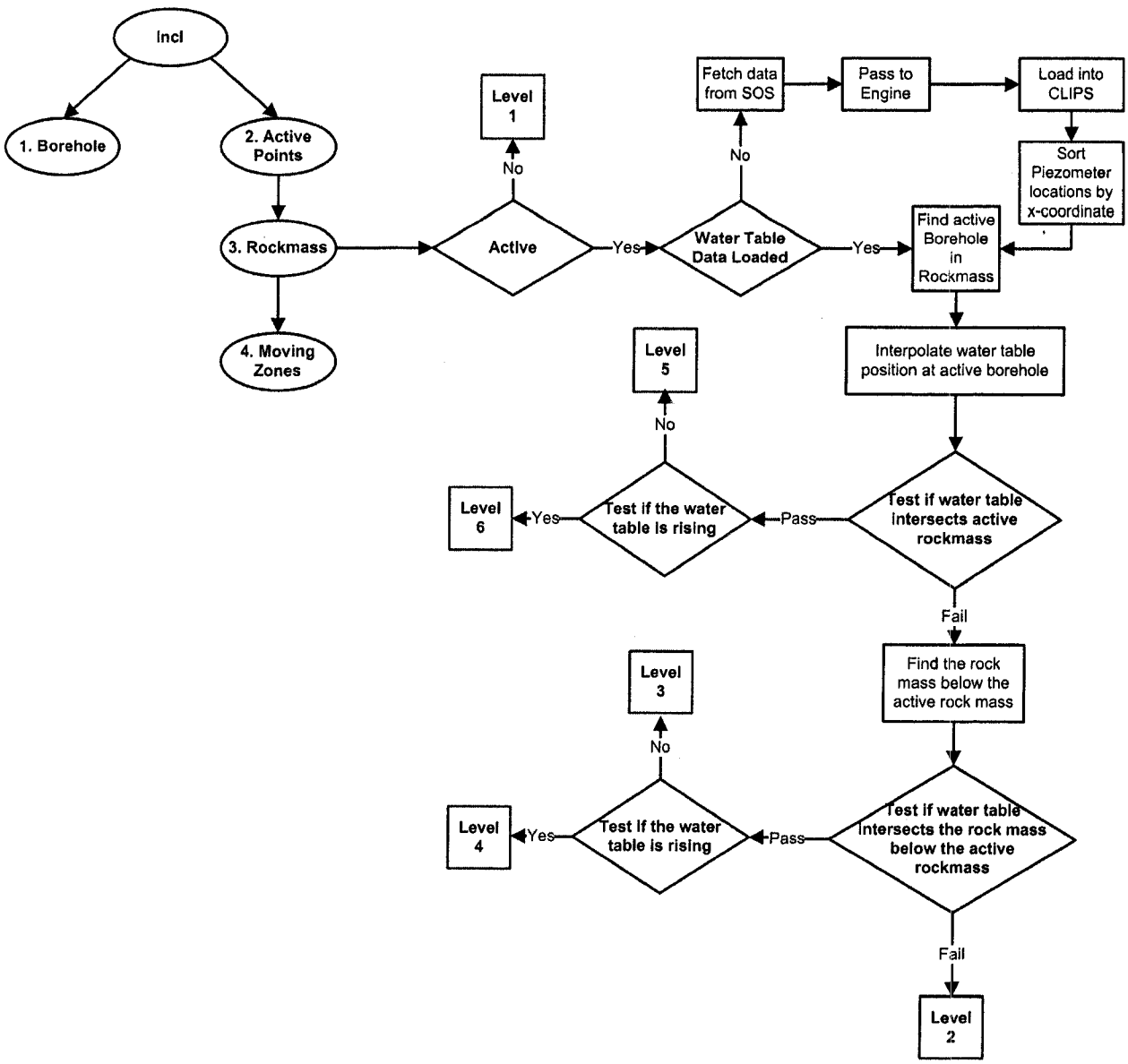

Figure 21 - Expanded REASON Decision Tree

The decision tree classifies all rock masses at every time step as having an alert level from one to six, with six being the most severe. The conditions for these alert levels are as follows:

Alert Level 6: The rock mass is active, the water table currently intersects the rock mass, and the water table is rising.

Alert Level 5: The rock mass is active, the water table currently intersects the rock mass, and the water table is falling.

Alert Level 4: The rock mass is active, the water table currently intersects the rock mass below the active rock mass, and the water table is rising.

Alert Level 3: The rock mass is active, the water table currently intersects the rock mass below the active rock mass, and the water table is falling.

Alert Level 2: The rock mass is active, and the water table does not intersect the active rock mass or the rock mass below.

Alert Level 1: The rock mass is not active.

The decision tree specifies the steps that are taken to identify alert levels based on the above criteria. These rules capture basic slope mechanics (Lee and Jones 2004), and were chosen based on the ability to rapidly implement them as well as the need to demonstrate the newest capabilities of the system. The system also checks the quality of incoming data by ensuring that the values used for analysis are reasonable. The standpipe measurements were simulated to include measurement errors such as missing values, and these are detected during the execution of the decision tree. When a missing value is detected, interpolation of the water table position is performed using the next closest standpipe reading, provided that it is reasonable. Figure 22 shows the result of a cycle of execution under the new decision tree. This shows how the addition of water level data from another sensor offering can enhance the information content delivered by the model, providing a visual representation of where the most hazardous areas of the slope are. The two sensor offerings do not need to come from the same data set or the same sensor network to be combined in a useful way. By relating the position of the water table with the activity of the individual rock masses the objective of integrating different data sources for the purposes of analyzing sensor data in a problem-specific manner is achieved.

It should be noted that the efficiency of the system is decreased when the extra decision logic is added. Each ArcAgents operation that occurs usually requires several internal steps, including inter-process and cross-system communication, so minimizing the number of these operations is the key to decreasing the operating time of each analysis cycle. The two main controls on this are the number of features that need to be updated each cycle, and the number of spatial operations that need to be performed. In this demonstration system, seventy-eight sensor locations (seventy-two inclinometer locations and six piezometer locations) are updated during each cycle, and several spatial queries are also needed each time the extended decision tree is executed (once per active rock mass). The system currently takes approximately three to four minutes to execute an analysis cycle.

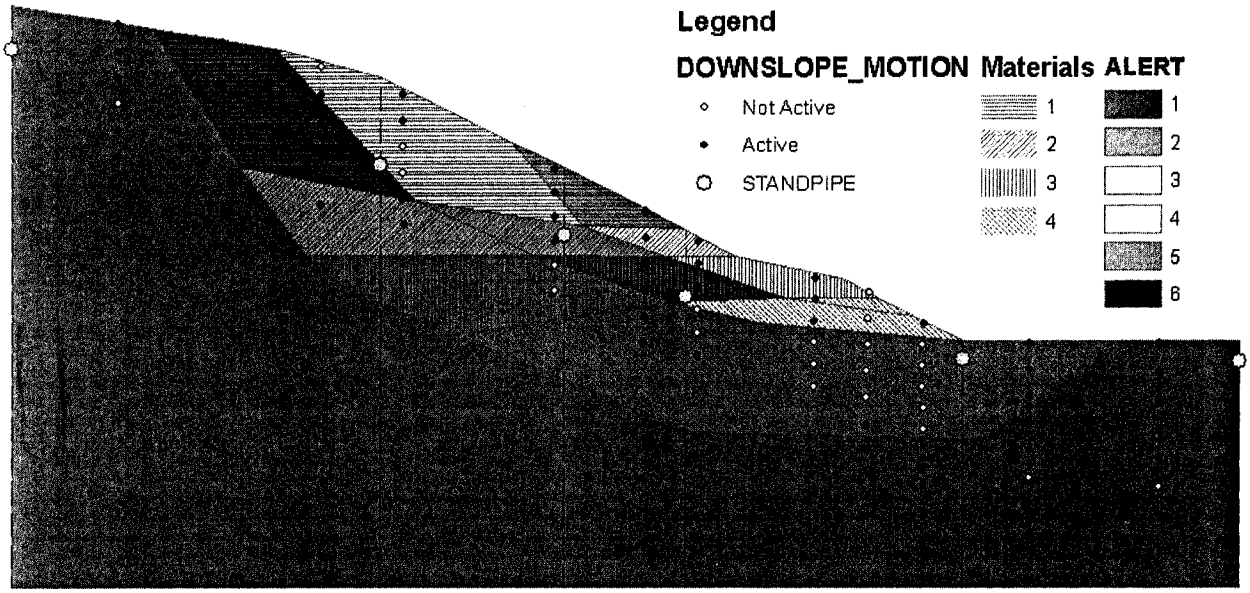

Figure 22 - Revised Slope Model

For the purposes of this demonstration, this was an acceptable timeframe in which to receive results, as the simulated measurements occur once per day. In a situation where new data are being fed to the system every few minutes, four minutes per analysis cycle may not be sufficient. However, the analysis time could be reduced by creating a simpler decision tree, or only focusing on specific sensors of interest in order to reduce the amount of data being handled during each cycle. Ultimately the suitability of this approach to any hazard monitoring problem would be largely dependent on the scope and complexity of the analysis to be performed, factors that must be considered at design time.

Other design approaches are being considered to connect rule-based analysis methods with ArcGIS, as well as alternate GIS packages to use instead of ArcGIS, in order to remove the performance bottleneck. This would potentially allow more complex decision trees with shorter evaluation times. Also, as ArcAgents continues to develop its own efficiency may be improved.

7. Conclusions

The growing number of sensor network installations has resulted in an increased amount of data for hazard managers to use when exploring a potentially dangerous site. These data can be useful for data-centric problem solving approaches, but to answer more complex problems richer more detailed information is needed, a viewpoint that motivated this thesis work. While much development has been done on sensor network infrastructure to support monitoring, it has been traditionally focused on the data-centric perspective. This is largely due to the constraints that exist in sensor hardware and typical messaging protocols, as well as by the types of problems that the data have been used to solve. Encoding geospatial information in a way that supports an information-centric perspective on sensor data within spatial decision support tools has been explored in previous work, however this is not the typical approach that sensor networks take when collecting and disseminating sensor data. Therefore a need exists to enable the information-centric perspective on sensor data while supporting the traditional data-centric perspective associated with current monitoring strategies. This thesis combines a spatial decision support system, common sensor web infrastructure, a sensor web ontology, and an engine for transforming geospatial information to achieve this goal.

Several different approaches to knowledge representation can be seen in various examples of hazard monitoring infrastructure. The examination of these different approaches has led to their classification based on characteristics such as the level of detail they contain, the structure used to model the data, and the functionality of the approach for use in a reasoning system. The integration of a sensor monitoring system with ontology-based knowledge representation and domain-specific knowledge encoding has resulted in a spatial decision support system that can be used to monitor sensors. By encoding problem-related knowledge into the system, expert users can ensure they are interpreting the information properly, and the software system can also derive meaning from the information. This domain knowledge can also be used to reinforce conclusions drawn by the software.

Hazard managers traditionally work with spatial data, not spatial information; however the use of spatial information enables more advanced problem investigation methods, including reasoning and other artificial intelligence techniques that are well suited to unstructured and semi-structured problems such as hazard monitoring. These methods are not commonly supported from the geospatial perspective, and when they are it is typically done in a very application-specific manner. The creation of a conceptual framework for the transformation of sensor data and sensor descriptions aims to bridge

this gap. The transformation chain is based on the classification of knowledge representation styles and is intended to serve as a guide to creating a transformation engine that minimizes the risk of information loss while enabling the enrichment of sensor data with contextual information. The generalized nature of the transformation engine allows the concepts to be applied to any monitoring problem, and an implementation of this engine that transforms common sensor data representations into CLIPS code that can be used within the ArcAgents environment to monitor hazards has been created. These supporting tools have been developed to automate the conversion of sensor data from common geospatial standards into an ontological format. This automation is important because the target user base of this system is not typically going to be well versed in knowledge representation. The user of the system should be focused on problem exploration, not knowledge representation. Beyond that, the automation ensures that the conversions are done in a consistent way, reducing the chances of ambiguity in the results of the conversion or inconsistencies between conversions.

The generalized design of the system allows multiple data sources to be integrated into the same representation style, ensuring that regardless of the type of data being used with the system it can feed knowledge-based analysis and drive simulation and hypothesis testing. The demonstration system integrates sensor data from two types of sensors measuring different phenomena. By analyzing this information the system is able to classify the hazard level at any given time by applying domain logic in much the same way that a domain expert would use their expertise. The target users for a system such as this are expert users and hazard managers who wish to monitor a sensor network installation or explore various 'what-if' scenarios by using simulation results in place of sensor data, as well as those who are planning sensor network installations and wish to model various configurations of sensor types, placements, triggers, and sampling rates to see how the different configurations affect the collection of relevant data.

The end result is a monitoring program that can be merged with existing monitoring infrastructure and workflows. Advanced reasoning techniques based on artificial intelligence methods are supported by a sensor and measurement ontology, and are automated using tools that make use of existing spatial data infrastructure. This enables expert users to apply their own expertise to their monitoring problem while making use of existing monitoring infrastructure. This system can also be tied to simulation engines and mock sensor data to perform hypothesis and scenario testing. Advanced domain ontologies can be plugged in to the system to enhance the conclusions that can be drawn, and knowledge reuse is

promoted by the use of ontologies to partition system knowledge. This thesis has demonstrated that existing sensor and monitoring infrastructures can be bridged in a way that supports information-centric analysis of a monitoring problem with the use of real-time sensor data. Approaching the problem in a generalized way has created the potential for this framework to be applied as a supplement to a hazard manager's current practices in order to improve the quality of their results and to allow deeper and more thorough problem investigation so that they may better protect people and infrastructure from potential hazards.

References

Aamodt, A., & Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications , 7* (1), 39-59.

Athanasiadis, I. N., & Mitkas, P. A. (2004). An agent-based intelligent environmental monitoring system. *Management of Environmental Quality , 15* (3), 238-250.

Avancha, S., Patel, C., & Joshi, A. (2004). Ontology-driven adaptive sensor networks. *MobiQuitous 2004, Proceedings* (pp. 194-202). Cambridge, MA, USA: IEEE Computer Society.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American , 284 (5)* , 28-37. CDS Communications Data Services.

Bonnet, P., Gehrke, J., & Seshadri, P. (2001). Towards sensor database systems. In K.-L. Tan, M. J. Franklin, & J. C.-S. Lui (Ed.), *Mobile Data Management: Second International Conference* (pp. 3-14). Hong Kong, China: Springer-Verlag.

Botts, M. (2005). OpenGIS Sensor Model Language (SensorML) Open Geospatial Consortium Best Practices Document. Open Geospatial Consortium.

Botts, M., Percivall, G., Reed, C., & Davidson, J. (2006). OGC Sensor Web Enablement: Overview And High Level Architecture. Open Geospatial Consortium Whitepaper. Open Geospatial Consortium.

Bovenga, F., Nutricato, R., Refice, A., & Wasowski, J. (2006). Application of multi-temporal differential interferometry to slope instability detection in urban/peri-urban areas. *Engineering Geology , 88* (3-4), 218-239.

Centintemel, U., Flinders, A., & Sun, Y. (2003). Power-efficient data dissemination in wireless sensor networks. *Proceedings of the 3rd ACM International Workshop on Data Engineering for Wireless and Mobile Access* (pp. 1-8). New York: ACM Press.

Chau, K. W. (2007). An ontology-based knowledge management system for flow and water quality modeling. *Advances in Engineering Software , 38* (3), 172-181.

Cheng, M.-Y., Chien-Ho, K., & Chang, C.-H. (2002). Computer-aided DSS for safety monitoring of geotechnical construction. *Automation in Construction , 11* (4), 375-390.

Colesanti, C., & Wasowski, J. (2006). Investigating landslides with space-borne synthetic aperture radar (SAR) interferometry. *Engineering Geology , 88* (3-4), 173-199.

Cox, S. (2006). Observations and measurements. OpenGIS Discussion Paper. Open Geospatial Consortium.

Cox, S., Daisey, P., Lake, R., Portele, C., & Whiteside, A. (2004). OpenGIS geography markup language (GML) implementation specification. OpenGIS Recommendation Paper. Open Geospatial Consortium.

Crosta, G. B., Chen, H., & Frattini, P. (2006). Forecasting hazard scenarios and implicaitons for the evaluation of countermeasure efficiency for large debris avalanches. *Engineering Geology , 83* (1-3), 236-253.

Datta, A., Vandermeer, D. E., Celik, A., & Kumar, V. (1999). Broadcast protocols to support efficient retrieval from databases by mobile users. *ACM Transactions on Database Systems , 24* (1), 1-79.

Dietrich, J., & Elgar, C. (2007). Towards a web of patterns. *Web Semantics: Science, Services, and Agents on the World Wide Web , 5*, 108-116.

Dunnicliff, J. (1993). *Geotechnical Instrumentation for Monitoring Field Performance.* New York: Wiley-Interscience.

Fukatsu, T., Hirafuji, M., & Kiura, T. (2006). An agent system for operating web-based sensor nodes via the internet. *Journal of Robotics and Mechatronics , 18* (2), 186-194.

Gannod, G. C., & Cheng, B. H. (1999). A framework for classifying and comparing software reverse engineering and design recovery techniques. *Sixth Working Conference on Reverse Engineering* (pp. 77-88). Washington, DC: IEEE Computer Society.

Garrett, J. J. (2005, February 18). *Ajax: a new approach to web applications*. Retrieved August 20, 2007, from Adaptive Path: http://www.adaptivepath.com/publications/essays/archives/000385.php

Gorman, B. L., Mallikarjun, S., & Smith, C. M. (2005, April 1). Advancing sensor web interoperability. *Sensors Magazine , 22 (4)* , 14-19.

Graniero, P. A., & Robinson, V. B. (2006). A probe mechanism to couple spatially explicit agents and landscape models in an integrated modelling framework. *International Journal of Geographical Information Science , 20* (9), 965-990.

Gröger, G., Kolbe, T., & Czerwinski, A. (2006, July 28). Candidate OpenGIS CityGML Implementation Specification. Open Geospatial Consortium.

Gross, N. (1999, August 30). 21 ideas for the 21st century: the Earth will don an electronic skin. *BusinessWeek Online* , 68-70. McGraw-Hill.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition , 5* (2), 199-220.

Gruninger, M., & Fox, M. S. (1995). Methodology for the design and evaluation of ontologies. *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95* (pp. 1-10). San Mateo, CA: Morgan Kaufmann Pub.

Guarino, N. (1998). *Formal Ontology in Information Systems: Proceedings of the 1st International Conference.* Amsterdam: IOS Press.

Gupta, P., & Kumar, P. R. (2000). The capacity of wireless sensor networks. *IEEE Transactions on Information Theory , 46* (2), 388-404.

Harrap, R. M., Hutchinson, D. J., Graniero, P., Diederichs, M., Moulin, B., & Martin, D. (2006). The GIST-II project - decision support strategies for natural hazards monitoring and analysis. *GEOIDE 8th Annual Scientific Conference, Poster Session* . Banff, Alberta, Canada.

Harris, C., Haeberli, W., Vonder Mühll, D., & King, L. (2001). Permafrost monitoring in the high mountains of Europe: the PACE project in its global context. *Permafrost and Periglacial Processes , 12* (1), 3-11.

Heidemann, J., Silva, F., Intanagonwiwat, C., Govindan, R., Estrin, D., & Ganesan, D. (2001). Building efficient wireless sensor networks with low-level naming. *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles* (pp. 146-159). New York: ACM Press.

Hutchinson, D. J., Diederichs, M. S., & Harrap, R. (2004). Landslide monitoring and analysis using GIS technology. *57th Canadian Geotechnical Conference, Proceedings*, (6 pp.).

Hutchinson, D. J., Diederichs, M. S., Carranza-Torres, C., Harrap, R., Rozic, S., & Graniero, P. (2007). Four dimensional considerations in forensic and predictive simulation of hazardous slope movement. In E. Eberhardt, D. Stead, & T. Morrison (Ed.), *Proceedings of the 1st Canada-US Rock Mechanics Symposium* (8 pp.). Taylor & Francis, Ltd.

ISO. (2003, May 8). ISO 19115 - An international metadata standard for geographic information. International Organization for Standardization.

Iverson, R. M. (2000). Landslide triggering by rain infiltration. *Water Resources Research , 36* (7), 1897-1910.

Jabeur, N., & Graniero, P. A. (Submitted). Creating an intelligent wireless sensor web from conventional sensor networks using a 'virtual wireless sensor network' concept. *International Journal of Systems Science* .

Jabeur, N., McCarthy, J. D., Xing, X., & Graniero, P. A. (Submitted). A knowledge-oriented meta-framework for integrating sensor network infrastructures. *Computers and Geosciences* .

Kim, K. J., & Yoo, S. J. (2005). Power-efficient reliable routing protocol for mobile ad-hoc networks. *IEICE Transactions on Communications , E88B* (12), 4588-4597.

Kokai, Y., Masuda, F., Horiike, S., & Sekine, Y. (1997). Recent development in open systems for EMS/SCADA. *International Journal of Electrical Power & Energy Systems , 20* (2), 111-123.

Krishnamachari, B., & Estrin, D. W. (2004). The impact of data aggregation in wireless sensor networks. *Proceedings of the 22nd International Conference on Distributed Computing Systems* (pp. 457-459). Washington, DC: IEEE Computer Society.

Lambrix, P., Habbouche, M., & Pérez, M. (2003). Evaluation of ontology development tools for bioinformatics. *Bioinformatics , 19* (12), 1564-1571.

Lee, E. M., & Jones, D. K. (2004). *Landslide risk assessment.* London: Thomas Telford.

Lehmann, F. (1992). Semantic networks. *Computers and Mathematics with Applications , 23* (2-5), 1-50.

Lenat, D. B. (1995). CYC: a large-scale investment in knowledge infrastructure. *Communications of the ACM , 38* (11), 33-38.

Li, J., Blake, C., de Couto, D. S., Lee, H. I., & Morris, R. (2001). Capacity of ad hoc wireless networks. *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking* (pp. 61-69). ACM Press: New York.

Lopez, M. F., Gomez-Perez, A., Sierra, J. P., & Sierra, A. P. (1999). Building a chemical ontology using Methontology and the OntologyDesign environment. *IEEE Intelligent Systems and Their Applications , 14* (1), 37-46.

Madden, S., Franklin, M. J., Hellerstein, J. M., & Hong, W. (2002). TAG: a tiny aggregation service for ad-hoc sensor networks. *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (pp. 131-146). New York: ACM Press.

Maidment, D. (2002). *Arc Hydro: GIS for Water Resources.* Redlands, California: ESRI Press.

McCarthy, J. D., Graniero, P. A., & Rozic, S. M. (2007). An Integrated GIS-Expert System Framework for Live Hazard Monitoring and Detection. *Proceedings of the Joint CIG/ISPRS Conference on Geomatics for Disaster and Risk Management* (11 pp.). CD-ROM.

Meisina, C., Zucca, F., Fossati, D., Ceriani, M., & Allievi, J. (2006). Ground deformation monitoring by using the permanent scatterers technique: the example of the Oltrepo Pavese (Lombardia, Italy). *Engineering Geology , 88* (3-4), 240-249.

Minsky, M. (1974). *A framework for representing knowledge.* Cambridge, MA, USA: Massachusetts Institute of Technology.

Mizen, H., Dolbear, C., & Hart, G. (2005). Ontology ontogeny: understanding how an ontology is created and developed. *Geospatial Semantics, Proceedings. Lecture Notes in Computer Science, 3799,* pp. 15-29.

Na, A., & Priest, M. (2006). OpenGIS Sensor Observation Service Implementation Specification. Open Geospatial Consortium.

Niles, I., & Pease, A. (2001). Towards a standard upper ontology. *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems* (pp. 2-9). New York: ACM Press.

Noy, N. F., & McGuinness, D. L. (2001). *Ontology development 101: a guide to creating your first ontology.* Stanford, CA: Knoledge Systems Laboratory.

O'Brien, J., & Gahegan, M. (2004). A knowledge framework for representing, manipulating and reasoning with geographic semantics. In P. Fisher (Ed.), *Developments in Spatial Data Handling: 11th International Symposium on Spatial Data Handling* (pp. 585-603). Berlin: Springer.

Øhrstrøm, P., Andersen, J., & Schärfe, H. (2005). What has happened to ontology? *Lecture Notes in Artificial Intelligence , 3596,* 425-438.

Polemio, M., & Petrucci, O. (2001). Hydrogeological monitoring and image analysis of a mudslide in southern Italy. *Physics and Chemistry of the Earth Part C-Solar-Terrestrial and Planetary Science , 26* (9), 689-695.

Probst, F., Gordon, A., & Dornelas, I. (2004). OGC discussion paper: ontology-based representation of the OGC observations and measurements model. Open Geospatial Consortium.

Pundt, H., & Bishr, Y. (2002). Domain ontologies for data sharing - an example from environmental monitoring using field GIS. *Computers and Geosciences , 28,* 95-102.

Quinlan, J. R. (1990). Decision trees and decision-making. *IEEE Transactions on Systems, Man and Cybernetics , 20* (2), 339-346.

Royer, E. M., & Chai-Keong, T. (1999). A review of current routing protocols for ad hoc mobile wireless networks. *Personal Communications, IEEE , 6,* 46-55.

Rozic, S. M. (2006). Representing spatial and domain knowledge within a spatial decision support framework. M.Sc. Thesis, University of Windsor.

Russomanno, D. J., Kothari, C., & Thomas, O. (2005). Building a sensor ontology: a practical approach leveraging ISO and OGC models. In H. R. Arabnia, & R. Joshua (Ed.), *The 2005 International Conference on Artificial Intelligence, Proceedings* (pp. 637-643). CSREA Press.

Santanche, A., Nath, S., Liu, J., Priyantha, B., & Zhao, F. (2006). *SenseWeb: browsing the physical world in real time.* Redmond, WA: Microsoft Research Report.

Sheth, A., Thekkath, C. A., Mehta, P., Tejaswi, K., Parekh, C., Singh, T. N., et al. (2007). Senslide: a distributed landslide predicition system. *ACM SIGOPS Operating Systems Review , 41* (2), 75-87.

Sirin, E., Parisa, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (In Press). Pellet: a practical OWL-DL reasoner. *Journal of Web Semantics* .

Stavroulaki, V., Demestichas, K., Adamopoulou, E., & Demesticahas, P. (2006). Distributed web-based management framework for ambient reconfigurable services in the intelligent environment. *Mobile Networks and Applications , 11*, 889-900.

Stoeckert Jr., C. J., Causton, H. C., & Ball, C. A. (2002). Microarray databases: standards and ontologies. *Nature Genetics , 32*, 469-473.

Sure, Y., Erdmann, M., Angele, J., Staab, S., Studer, R., & Wenke, D. (2002). OntoEdit: collaborative ontology development for the semantic web. *First International Semantic Web Conference. 2342*, pp. 221-235. Berlin: Springer.

Terzis, A., Anandaraja, A., Moore, K., & Wang, I.-J. (2006). Slip surface localization in wireless sensor networks for landslide prediction. *Proceedings of the fifth international conference on Information processing in sensor networks* (pp. 109-116). New York: ACM Press.

Tsarkov, D., & Horrocks, I. (2006). FaCT++ description logic reasoner: system description. In U. Furbach, & N. Shankar (Ed.), *Proceedings of the 3rd International Joint Conference on Automated Reasoning* (pp. 292-297). Berlin: Springer.

Turban, E., Aronson, J. E., & Liang, T.-P. (2005). *Decision Support Systems and Intelligent Systems* (7th ed.). Upper Saddle River, New Jersey: Pearson Education, Inc.

Varro, A., & Pataricza, A. (2003). UML action semantics for model transformation systems. *Periodica Polytechnica Electrical Engineering , 47* (3/4), 167-186.

Wagner, R., Sarvotham, S., & Baraniuk, R. (2005). A multiscale data representation for distributed sensor networks. *IEEE Conference on Acoustics, Speech, and Signal Processing, 2005. Proceedings. 4*, pp. 549-552. IEEE.

Wang, H. H., Li, Y. F., Sun, J., Zhang, H., & Pan, J. (2007). Verifying feature models using OWL. *Web Semantics: Science, Services, and Agents on the World Wide Web , 5*, 117-129.

Wassmann, R., Hien, N. X., Hoanh, C. T., & Tuong, T. P. (2004). Sea level rise affecting the Vietnamese Mekong delta: water elevation in the flood season and implications for rice production. *Climatic Change , 66* (1), 89-107.

Wuwongse, V., Anutariya, C., Akama, K., & Nantajeewarawat, E. (2001). XML declarative description: a language for the Semantic Web. *Intelligent Systems , 16* (3), 54-65.

Yu, F. C., Chen, C. Y., Lin, S. C., Lin, Y. C., Wu, S. Y., & Cheung, K. W. (2007). A web-based decision support system for slopeland hazard warning. *Environmental Monitoring Assessment , 127* (1-3), 419-428.

Zellweger, P. (2005). A database taxonomy based on data-driven knowledge modeling. In C. Thompson, & H. Hexmoor (Ed.), *International Conference on Integration of Knowledge Intensive Multi-Agent Systems* (pp. 469-474). IEEE.

Zimmerman, R., Kas, S., Butscher, R., & Bodendorf, F. (2005). An ontology for agent-based monitoring of fulfillment processes. In *Whitestein Series in Software Agent Technologies and Autonomic Computing* (pp. 323-345). Basel, Switzerland: Birkhauser.

Zourmpakis, A., Boardman, D. I., Rogers, C. D., Jefferson, I., Gunn, D. A., Jackson, P. D., et al. (2006). Case Study of a Loess Collapse Field Trial in Kent, SE England. *Quarterly Journal of Engineering Geology and Hydrogeology , 39*, 131-150.

Appendix A: Ontology Specification

This appendix details the new sensor ontology. Figures A-1 to A-5 show the hierarchical relationship of all concepts in the sensor ontology. Table A-1 shows all relationships within the ontology as triples.

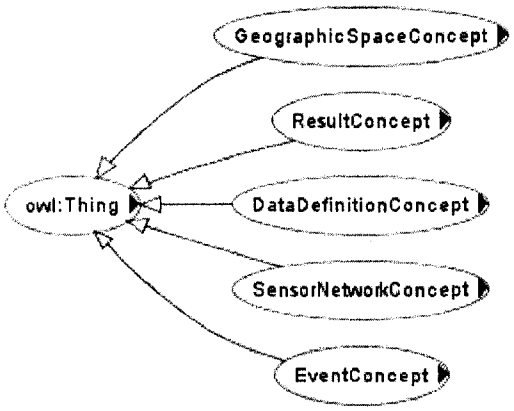

Figure A - 1: High level ontology concepts

Figure A - 2: Geographic space ontology concepts

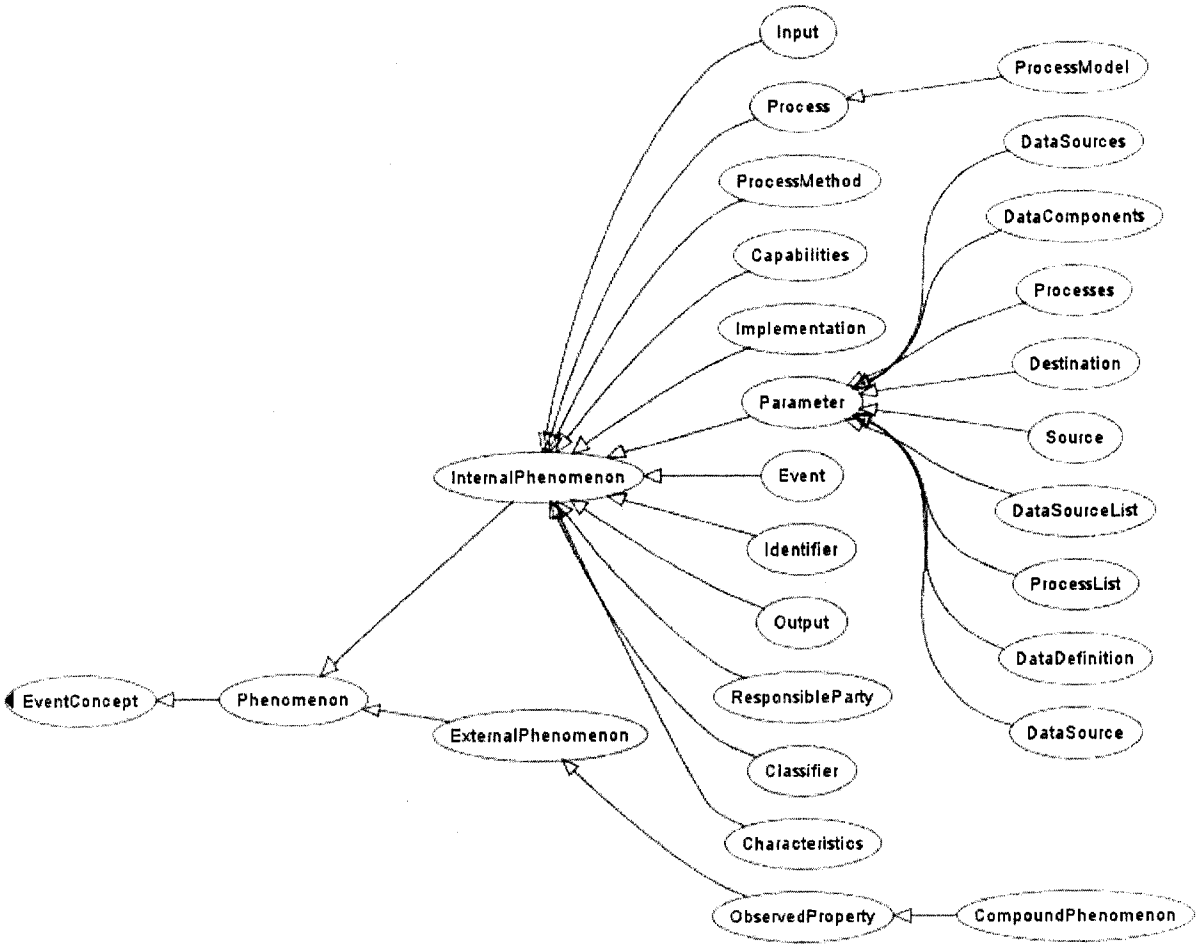

Figure A - 3: Event ontology concepts

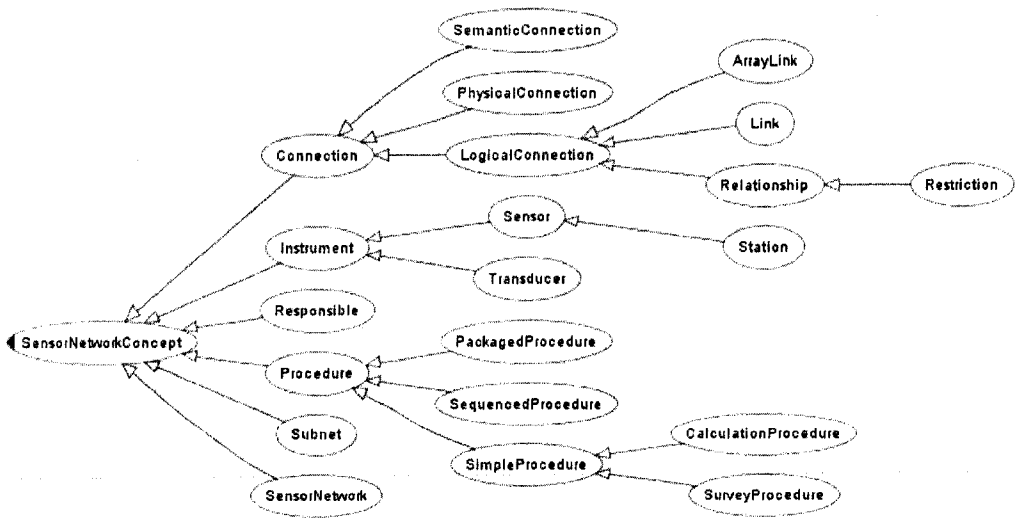

Figure A - 4: Sensor Network ontology concepts

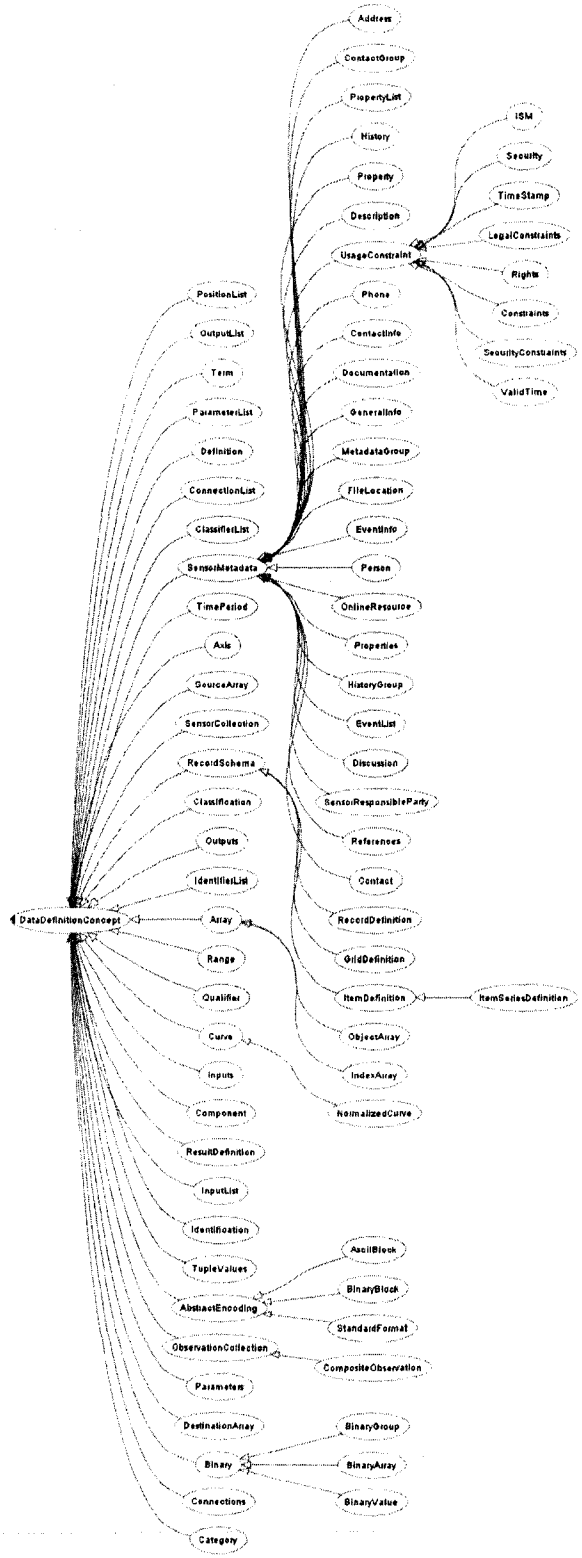

Figure A - 5: Data Definition ontology concepts

Table A - 1 - Ontology property descriptions

Datatype (D) or Object (O) Property	Property Name	Domain	Range
D	SRSName	LocalCoordinateFrame OR Envelope	string
D	UOM	Area OR Value OR Component OR Axis	string
D	XUOM	SamplePosition	string
D	YUOM	SamplePosition	string
D	anchorPoint	Datum	string
D	area	Polygon	float
O	arrayMap	owl:Thing	Array
O	axis	Coordinates	Axis
D	base	Value OR Restriction	string
D	beginPosition	BeginPosition	string
D	boolean	Boolean	boolean
D	booleanData	owl:Thing	boolean
O	boundedBy	LayerOfInterest OR Result	Polygon
D	classification	SimpleData	string
D	codeSpace	Identifier OR Category OR Value	string
D	date	Date	date
D	dateTime	DateTime	dateTime
D	definition	Category OR Value OR FeatureOfInterest OR ResultDefinition OR Component OR Axis	string
D	description	ObservationCollection OR Instrument OR Solid OR Value	string
D	doubleData	owl:Thing	float
D	duration	owl:Thing	string
D	endPosition	EndPosition	string

D	float	Float	float
D	frame	SimpleData	string
O	generates	Procedure	Result
O	hasAddress	owl:Thing	Address
D	hasAdministrativeArea	owl:Thing	string
D	hasAffiliation	owl:Thing	string
D	hasAlgorithm	owl:Thing	string
D	hasAnchorPoint	owl:Thing	string
O	hasArea	LayerOfInterest	Area
O	hasArrayComponent	owl:Thing	BinaryArray
D	hasArraySize	Curve OR DataArray	int
O	hasAsciiBlock	owl:Thing	AsciiBlock
D	hasAxis	owl:Thing	string
D	hasAxisName	owl:Thing	string
D	hasAxisType	owl:Thing	string
O	hasBeginPosition	TimePeriod	BeginPosition
D	hasBitLength	owl:Thing	int
D	hasBitOffset	owl:Thing	int
D	hasByteEncoding	owl:Thing	string
D	hasByteLength	owl:Thing	int
D	hasByteOffset	owl:Thing	int
D	hasByteOrder	owl:Thing	string
D	hasCS	owl:Thing	string
O	hasCapabilities	owl:Thing	Capabilities
O	hasCategory	DataComponents	Category
O	hasCharacteristics	owl:Thing	Characteristics
D	hasCity	owl:Thing	string
O	hasClassification	Sensor	Classification
O	hasClassifier	ClassifierList	Classifier

O	hasClassifierList	Classification	ClassifierList
O	hasComponent	DataGroup OR DataArray	Component
O	hasComponentPhenomena	ItemDefinition	RecordSchema
D	hasCompression	owl:Thing	string
O	hasConnection	ConnectionList	Connection
O	hasConnections	Sensor	Connections
O	hasConnectionList	Connections	ConnectionList
O	hasConstraints	owl:Thing	Constraints
O	hasContact	owl:Thing	Contact
O	hasContactGroup	owl:Thing	ContactGroup
O	hasContactInfo	owl:Thing	ContactInfo
D	hasContactInstructions	owl:Thing	string
O	hasCoordinateReferenceSystem	owl:Thing	CoordinateReferenceSystem
O	hasCoordinates	Definition	Coordinates
D	hasCopyRights	owl:Thing	boolean
D	hasCountry	owl:Thing	string
O	hasCurve	NormalizedCurve	Curve
O	hasDataArray	owl:Thing	DataArray
O	hasDataComponents	DataDefinition	DataComponents
O	hasDataDefinition	owl:Thing	DataDefinition
O	hasDataGroup	owl:Thing	DataGroup
O	hasDataSource	owl:Thing	DataSource
O	hasDataSourceList	owl:Thing	DataSourceList
O	hasDataSources	owl:Thing	DataSource OR DataSources
D	hasDataType	owl:Thing	string
O	hasDatum	owl:Thing	Datum
D	hasDatumName	owl:Thing	string
D	hasDecimalSeparator	AsciiBlock	string
O	hasDefinition	Curve	Definition

D	hasDeliveryPoint	owl:Thing	string
O	hasDescription	owl:Thing	Description
O	hasDestination	Link	Destination
O	hasDestinationArray	owl:Thing	DestinationArray
D	hasDimension	CompoundPhenomenon	string
O	hasDiscussion	owl:Thing	Discussion
O	hasDocumentation	owl:Thing	Documentation
D	hasDoubleList	owl:Thing	string
D	hasElectronicMailAddress	owl:Thing	string
O	hasElement	PackagedProcedure	Procedure
D	hasEmail	owl:Thing	string
O	hasEncoding	owl:Thing	AbstractEncoding
D	hasEncryption	owl:Thing	string
O	hasEndPosition	TimePeriod	EndPosition
O	hasEvent	owl:Thing	EventInfo
O	hasEventList	owl:Thing	EventList
D	hasFacsimile	owl:Thing	string
O	hasFeatureOfInterest	Observation OR Measurement	LayerOfInterst OR Sensor OR FeatureOfInterest OR SensorCollection
O	hasFileLocation	owl:Thing	FileLocation
O	hasFollowingEvent	Event	Event
D	hasFormat	owl:Thing	string
O	hasFunction	owl:Thing	Curve
O	hasGeneralInfo	owl:Thing	GeneralInfo
O	hasGroupComponent	owl:Thing	BinaryGroup
O	hasHistory	owl:Thing	History
O	hasHistoryGroup	owl:Thing	HistoryGroup
D	hasHoursOfService	owl:Thing	string

O	hasISM	owl:Thing	ISM
O	hasIdentification	Instrument	Identification
O	hasIdentifier	ObservedProperty OR IdentifierList	Identifier
O	hasIdentifierList	Identification	IdentifierList
O	hasImplementation	owl:Thing	Implementation
D	hasIndexDestination	owl:Thing	int
D	hasIndexSource	owl:Thing	int
D	hasIndividualName	owl:Thing	string
O	hasInput	InputList	Input
D	hasInputBias	NormalizedCurve	string
D	hasInputGain	NormalizedCurve	string
O	hasInputList	Inputs	InputList
O	hasInputs	Instrument	Inputs
D	hasIntegerList	owl:Thing	string
D	hasIntellectualPropertyRights	owl:Thing	boolean
O	hasItemDefinition	Component	ItemDefinition
O	hasLegalConstraints	owl:Thing	LegalConstraints
O	hasLink	owl:Thing	Link
D	hasLinkRef	owl:Thing	string
O	hasLocalCoordinateFrame	ReferenceFrame OR SamplePosition	LocalCoordinateFrame
O	hasLocation	Result OR Event	GeographicSpaceConcept
O	hasMetadata	owl:Thing	MetadataGroup
O	hasMethod	owl:Thing	ProcessMethod
D	hasMimeType	owl:Thing	string
D	hasName	owl:Thing	string
O	hasNormalizedCurve	owl:Thing	NormalizedCurve
O	hasObservation	ObservationCollection	Observation
O	hasObservedProperty	Observation OR Measurement	ObservedProperty
O	hasOnlineResource	owl:Thing	OnlineResource

D	hasOrganizationName	owl:Thing	string
D	hasOrientation	Datum	string
O	hasOutput	OutputList	Output
D	hasOutputBias	NormalizedCurve	string
D	hasOutputGain	NormalizedCurve	string
O	hasOutputList	Outputs	OutputList
O	hasOutputs	Instrument	Outputs
O	hasParameter	ParameterList	Parameter OR Curve OR NormalizedCurve
O	hasParameterList	Parameters	ParameterList
O	hasParameters	Instrument	Parameters
O	hasPerson	owl:Thing	Person
D	hasPhenomenonProperty	ItemDefinition	string
O	hasPhone	owl:Thing	Phone
D	hasPhoneNumber	owl:Thing	string
O	hasPoint	SamplePosition	Point
O	hasPositionElement	owl:Thing	Position
O	hasPositionList	owl:Thing	PositionList
D	hasPositionName	owl:Thing	string
O	hasPosition	owl:Thing	Point
D	hasPostalCode	owl:Thing	string
O	hasPrecedingEvent	Event	Event
D	hasPrivacyAct	owl:Thing	boolean
O	hasProcedure	Observation OR Measurement	Procedure
O	hasProcess	ProcessList	Process
O	hasProcessChain	owl:Thing	Process
O	hasProcessList	Processes	ProcessList
O	hasProcessMethod	owl:Thing	ProcessMethod
O	hasProcesses	Sensor	Processes

O	hasProcessingStep	SequencedProcedure	Procedure
O	hasProperties	owl:Thing	Properties
O	hasProperty	PropertyList	Property
O	hasPropertyList	owl:Thing	PropertyList
D	hasQualifier	Value	string
O	hasRecordDefComponent	RecordSchema	ItemDefinition
O	hasRecordItem	Record	Value
D	hasRecordLength	owl:Thing	int
O	hasReferenceFrame	Instrument	ReferenceFrame
O	hasReferences	owl:Thing	References
O	hasResponsibleParty	owl:Thing	SensorResponsibleParty
O	hasRestriction	SimpleData	Restriction
O	hasResult	Result OR Component	Value
O	hasResultDefinition	owl:Thing	ResultDefinition
O	hasRights	owl:Thing	Rights
D	hasRole	Contact OR Value OR FeatureOfInterest	string
D	hasSRSName	Point	string
O	hasSamplePosition	Station	SamplePosition
D	hasScale	Component OR Axis	float
O	hasSecurity	owl:Thing	Security
O	hasSecurityConstraints	owl:Thing	SecurityConstraints
O	hasSensor	SensorCollection OR Station	Sensor
O	hasShape	LayerOfInterest	Shape
O	hasSolid	Result OR Event	Solid
O	hasSource	Link	Source
O	hasSourceArray	owl:Thing	SourceArray
O	hasSpatialInfo	Definition	GeographicSpaceConcept
O	hasStructure	owl:Thing	Binary
D	hasSurname	owl:Thing	string

O	hasTerm	Classifier OR Identifier	Term
D	hasTermQualifier	Term	string
D	hasTermValue	Term	string
O	hasTime	ObservationCollection OR CoordinateReferenceSystem OR Observation OR Measurement OR Event	DateTime
O	hasTimePosition	TimeInstant	TimePosition
O	hasTimeStamp	owl:Thing	TimeStamp
D	hasTokenSeparator	AsciiBlock	string
D	hasTopic	owl:Thing	string
O	hasTransducer	Process	Transducer
O	hasTupleMap	owl:Thing	RecordSchema
D	hasTupleSeparator	AsciiBlock	string
D	hasTupleType	owl:Thing	string
D	hasTupleValues	Curve	string
D	hasURI	Procedure OR Shape OR Component	string
D	hasUserID	owl:Thing	string
O	hasValidTime	owl:Thing	ValidTime
O	hasValue	ItemDefinition OR Parmaeter OR Record OR Observation OR Measurement OR Output OR Input	Value
D	hasVoice	owl:Thing	string
O	hasXMLTuple	owl:Thing	string
O	hostsProcedure	LayerOfInterest OR Station	Procedure

D	id	TimeInstant OR Point OR Measurement OR DataGroup OR LocalCoordinateFrame OR Station OR Restriction OR ObservationCollection OR LayerOfInterest OR RecordSchema OR CompoundPhenomenon OR Transducer OR Solid OR SensorCollection OR TimePeriod	string
D	identifier	Identifier	string
D	indeterminatePosition	TimePosition	string
D	integer	Integer	int
D	integerData	owl:Thing	int
O	isElementOf	Procedure	PackagedProcedure
O	isFeatureOfInterestOf	LayerOfInterest OR Sensor OR FeatureOfInterest OR SensorCollection	Observation OR Measurement
D	isFixed	owl:Thing	boolean
O	isFollowingEventOf	Event	Event
O	isGeneratedBy	Result	Procedure
O	isHostedBy	Procedure	LayerOfInterest OR Station
O	isLocationOf	GeographicSpaceConcept	Result OR Event
O	isObservedPropertyOf	ObservedProperty	Observation OR Measurement
O	isPrecedingEventOf	Event	Event
O	isProcedureOf	Procedure	Observation OR Measurement
O	isProcessingStepOf	Procedure	SequencedProcedure
O	isPropertyOf	Property	PropertyList
O	isResponsibleFor	ResponsibleParty	Event
O	isResultOf	Value	Result OR Component

O	isTimeOf	DateTime	ObservationCollection OR CoordinateReferenceSystem OR Observation OR Measurement OR Event
O	isTupleMapOf	owl:Thing	GridDefinition
D	linkRef	Source OR Destination	string
O	lowerCorner	Envelope	Point
D	maxInclusive	Range OR Restriction	string
D	member	owl:Thing	string
D	method	Transducer	string
D	mimeType	Value	string
D	minInclusive	Range OR Restriction	string
D	name	ObservationCollection OR Classifier OR Process OR Identifier OR Output OR Connection OR Input OR Component OR Datum OR Axis	string
D	pos	owl:Thing	string
D	position	owl:Thing	string
D	recordLength	RecordDefinition	int
D	referenceSystem	Value	string
D	serialNumber	owl:Thing	string
D	string	String	string
D	stringData	owl:Thing	string
D	time	Time	time
D	timeDefinition	Component	string
D	timePosition	TimePosition	string
D	tokenData	owl:Thing	string
D	type	Value	string
O	underResponsibilityOf	Event	ResponsibleParty

O	upperCorner	Envelope	Point
D	usesCS	LocalCoordinateFrame	string
O	usesDatum	LocalCoordinateFrame	Datum
D	value	Value	string
D	x	Point	string
D	y	Point	string
D	z	Point	string

Appendix B: Fact-Function Syntax

This appendix details the syntax for using "fact-functions", introduced in Section 5.1. The following items should be noted when using these commands:

- Items written in angle brackets (< >) are placeholders that are defined later on in the syntax
- Quotation marks (" ")are part of the syntax wherever shown
- Not all combinations of <filter> <structure> may be used together. Spatial filters and spatial structures should be paired, as should Temporal filters and temporal structures. The spatial filters are: BBOX, Contains, Intersects, Overlaps. The Temporal filters are: After, Before, During, TEquals. The spatial structures are: env and pt. The temporal structures are a time instant value or a pair of time instants representing a range.
- Any number of <featureOfInterest> and <ResultFilters> may be used, they are chained together with a logical AND operation. Contradictory filters are allowed. For example, limiting results to those above 10 and below 5 would be perfectly valid syntactically, but would never yield results since a value cannot be greater than 10 and less than 5 at the same time. Likewise, searching for all features contained within two disjoint bounding boxes is a valid request, but will not return any results.

SYNTAX for (send-SOS-request) command

(send-SOS-request "obs" <OFFERING> <PROCEDURE> <OBSERVEDPROPERTY> [<FEATUREOFINTEREST> <RESULTFILTERS>])

(send-SOS-request "cap" <SECTIONS>)

(send-SOS-request "des" <PROCEDURE>)

(send-SOS-request "sen")

(send-SOS-request "mro" <OFFERING> <OBSERVEDPROPERTY>)

Option	Description
"obs"	Creates an SOS GetObservation request
"cap"	Creates an SOS GetCapabilities request
"des"	Creates an SOS DescribeSensor request
"sen"	Creates an SOS GetCapabilities request that is equivalent to (send-SOS-request "cap" "c")
"mro"	Creates an SOS GetObservation request for retrieving the most recent set of sensor measurements for a given offering and phenomenon
<SECTIONS>	Describes the sections of the SOS Capabilities document that should be returned by a GetCapabilities request. Can either be "all" or any combination of: • "si" (Service Identification) • "sp" (Service Provider)

	• "om" (Operations Metadata) • "fc" (Filter Capabilities) • "c" (Contents)
<OFFERING>	A string that represents the SOS Offering that observations are being requested from. Ex, "DOWNSLOPE_MOTION"
<PROCEDURE>	A string that represents a procedure (sensor name). Ex, "urn:ogc:def:procedure:inclinometer-4"
<OBSERVEDPROPERTY>	A string that represents the phenomena being measured by a given procedure. Ex, "urn:ogc:phenomenon:distance"
<FEATUREOFINTEREST>	A collection of options that defines a spatial or temporal filter on the results of the query. Takes the form <FILTER> <STRUCTURE>.
<RESULTFILTERS>	A collection of options that defines a numerical filter on the results of the query. Takes the form <OPERATOR> <VALUE>.
<FILTER>	A spatial or temporal operator. One of: BBOX, Contains, Intersects, Overlaps, After, Before, During, TEquals
<STRUCTURE>	A spatial or temporal structure that serves as an operand for a <FILTER>. One of: • env <SRSNAME> (<LOWERCORNERX>,<LOWERCORNERY>) (<UPPERCORNERX>,<UPPERCORNERY>) • pt <SRSNAME> (<X>,<Y>) • <TIME> • <STARTTIME> <ENDTIME>
<SRSNAME>	A string denoting the Spatial Reference System used by the coordinates supplied. Ex, EPSG:4326
<LOWERCORNERX>	A number representing the x-coordinate of the lower-left corner of an envelope. Ex, 23.54
<LOWERCORNERY>	A number representing the y-coordinate of the lower-left corner of an envelope. Ex, 23.54
<UPPERCORNERX>	A number representing the x-coordinate of the upper-right corner of an envelope. Ex, 23.54
<UPPERCORNERY>	A number representing the y-coordinate of the upper-right corner of an envelope. Ex, 23.54
<X>	A number representing the x-coordinate of a point. Ex, 23.54
<Y>	A number representing the y-coordinate of a point. Ex, 23.54
<TIME>	A time stamp representing an instant in time. Ex, 2007-05-12T16:25:00
<STARTTIME>	A time stamp representing the beginning of a time range. Ex, 2007-05-12T16:25:00
<ENDTIME>	A time stamp representing the end of a time range. Ex, 2007-05-12T16:25:00
<OPERATOR>	A comparative operator that filters the results of a query using Boolean logic. Objects that pass the query are returned as part of the result set while objects that fail are not. One of: • Between <LOWERVALUE> <UNIT> <UPPERVALUE> <UNIT>

	• EqualTo <VALUE> <UNIT> • NotEqualTo <VALUE> <UNIT> • LessThan <VALUE> <UNIT> • LessThanEqualTo <VALUE> <UNIT> • GreaterThan <VALUE> <UNIT> • GreaterThanEqualTo <VALUE> <UNIT> • Like <VALUE>
<VALUE>	The operand of a ResultFilter's Operator. Is a numerical value with all operators except Like. The Like operator must have a String as its operand. Ex, 23.54
<UNIT>	The unit of measure for a numerical value. Ex, m

Appendix C: Contents of Companion CD

The software, tools, and documentation developed over the course of this thesis have been collected onto a CD that supplements the contents of this document. The contents of the CD are broken down by directory in this Appendix. If the CD did not accompany the thesis contact Phil Graniero, Department of Earth and Environmental Sciences, University of Windsor, graniero@uwindsor.ca

2owl: XSLT stylesheets that automate the transformation of SensorML and O&M documents into OWL documents

conversionControl: A Visual Basic application that can be used to run OGC to OWL, OGC to CLIPS, and OWL to CLIPS transformations

ontology: The sensor ontology as an OWL document

Ontology Design: Notes on the development methodology used to create the ontology, as well as the results of various steps of the development process

owl2clips: Java code and design notes for the OWL to CLIPS conversion tool

REASON: The REASON system as used in the slope monitoring demo in Chapter 6. Includes CLIPS code, sample data, an ArcMap document, and the ArcMap template that contains the VBA code needed to interact with the SOS.

semanticRepair: Design notes and partial implementation of a semantic error detection and report generation program.

SOS: SQL files containing the statements used to populate a 52°North SOS server with sample sensors and measurements. The "Sensors" subdirectory contains SensorML files that correspond to the contents of the SOS.

thesis_proposal: A PDF version of the thesis proposal and presentation

thesis: A PDF version of this thesis and the defense presentation

visio: Visio diagrams associated with the design and development of the various software components

Vita Auctoris

James Dwight McCarthy was born in 1982 in Windsor, Ontario. He graduated from Assumption High School in 2001. From there he studied in the Urban Planning program at the University of Waterloo for a year before transferring to the Geoinformatics program at the University of Windsor from which he graduated with Honours in 2005 with a Bachelor of Science. He graduated with a Master of Science degree in Earth Sciences from the Department of Earth and Environmental Sciences at the University of Windsor in September of 2007.