ONTOLOGY DRIVEN MULTI-AGENT SYSTEMS: AN ARCHITECTURE FOR SENSOR WEB APPLICATIONS

by

DESHENDRAN MOODLEY

Submitted in fulfillment of the academic requirements for the degree of Doctor of Philosophy in the School of Computer Science, Faculty of Science and Agriculture, University of KwaZulu-Natal, Durban, South Africa, December 2009

As the candidate's supervisor I have approved this dissertation for submission.

Signed:____________ Name:_____________ Date:_____________

ABSTRACT

Advances in sensor technology and space science have resulted in the availability of vast quantities of high quality earth observation data. This data can be used for monitoring the earth and to enhance our understanding of natural processes. Sensor Web researchers are working on constructing a worldwide computing infrastructure that enables dynamic sharing and analysis of complex heterogeneous earth observation data sets. Key challenges that are currently being investigated include data integration; service discovery, reuse and composition; semantic interoperability; and system dynamism. Two emerging technologies that have shown promise in dealing with these challenges are ontologies and software agents. This research investigates how these technologies can be integrated into an Ontology Driven Multi-Agent System (ODMAS) for the Sensor Web.

The research proposes an ODMAS framework and an implemented middleware platform, i.e. the Sensor Web Agent Platform (SWAP). SWAP deals with ontology construction, ontology use, and agent based design, implementation and deployment. It provides a semantic infrastructure, an abstract architecture, an internal agent architecture and a Multi-Agent System (MAS) middleware platform. Distinguishing features include: the incorporation of Bayesian Networks to represent and reason about uncertain knowledge; ontologies to describe system entities such as agent services, interaction protocols and agent workflows; and a flexible adapter based MAS platform that facilitates agent development, execution and deployment. SWAP aims to guide and ease the design, development and deployment of dynamic alerting and monitoring applications. The efficacy of SWAP is demonstrated by two satellite image processing applications, viz. wildfire detection and monitoring informal settlement. This approach can provide significant benefits to a wide range of Sensor Web users. These include: developers for deploying agents and agent based applications; end users for accessing, managing and visualising information provided by real time monitoring applications, and scientists who can use the Sensor Web as a scientific computing platform to facilitate knowledge sharing and discovery.

An Ontology Driven Multi-Agent Sensor Web has the potential to forever change the way in which geospatial data and knowledge is accessed and used. This research describes this far reaching vision, identifies key challenges and provides a first step towards the vision.

PREFACE

The research work described in this dissertation was carried out in the School of Computer Science, University of KwaZulu-Natal, Durban, from March 2001 to December 2009, under the supervision of Prof. Jules R. Tapamo and Prof. Johnson D.M. Kinyua

These studies represent original work by the author and have not otherwise been submitted in any form for any degree or diploma to any tertiary institution. Where use has been made of the work of others it is duly acknowledged in the text.

DECLARATION 1 - PLAGIARISM

I, Deshendran Moodley, declare that:

1. The research reported in this thesis, except where otherwise indicated, is my original research.

2. This thesis has not been submitted for any degree or examination at any other university.

3. This thesis does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.

4. This thesis does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

   (a) Their words have been re-written but the general information attributed to them has been referenced

   (b) Where their exact words have been used, then their writing has been placed in italics and inside quotation marks, and referenced.

5. This thesis does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the thesis and in the References sections.

Signed: ______________

DECLARATION 2 - PUBLICATIONS

1. MOODLEY, D., AND KINYUA, J. A multi-agent system for electronic job markets. In *Proc. 6th International conference on Business Information Systems, Colorado Springs, USA, 4-6 June 2003, published by Dept. of Management Info. Systems, The Poznan University of Economics, Poznan* (2003), pp. 42–48

2. MOODLEY, D. The future of the Internet: The semantic web, web services and a multi-agent system infrastructure for the Internet. In *Proc. South African Computer Lecturers Association 2004, 4-6 July Durban, 2004* (2004)

3. MOODLEY, D., AND KINYUA, J. D. M. Towards a multi-agent infrastructure for distributed Internet applications. In *8th Annual Conference on WWW Applications, Bloemfontein, South Africa, 5-6 September* (2006)

4. MOODLEY, D., TERHORST, A., SIMONIS, I., MCFERREN, G., AND VAN DEN BERGH, F. Using the sensor web to detect and monitor the spread of wild fires. In *Second International Symposium on Geo-information for Disaster Management (Gi4DM) September 25-26, Pre-Conference Symposium to ISPRS TC-IV and ISRS Symposium on Geospatial Databases for Sustainable Development September 27-30, at Goa, India* (2006)

5. MOODLEY, D., AND SIMONIS, I. A new architecture for the sensor web: the SWAP-framework. In *Semantic Sensor Networks Workshop, A workshop of the 5th International Semantic Web Conference ISWC 2006, November 5-9, Athens, Georgia, USA* (2006)

6. TERHORST, A., SIMONIS, I., AND MOODLEY, D. A service-oriented multi-agent systems architecture for the sensor web. In *SAEON Summit, Centurion, South Africa* (2006)

7. MOODLEY, D., VAHED, A., SIMONIS, I., MCFERREN, G., AND ZYL, T. V. Enabling a new era of earth observation research: scientific workflows for the sensor web. *Ecological Circuits 1* (2008), 20–23

8. TERHORST, A., MOODLEY, D., SIMONIS, I., FROST, P., MCFERREN, G., ROOS, S., AND VAN DEN BERGH, F. *Geosensor Networks, Lecture Notes in Computer Science, Volume 4540/2008*. Springer-Verlag, 2008, ch. Using the Sensor Web to Detect and Monitor the Spread of Vegetation Fires in Southern Africa, pp. 239–251

Signed: ______________

ACKNOWLEDGEMENTS

Many people have supported me through this endeavour. I am deeply grateful to my wife for her patience, understanding, support and for allowing the PhD to permeate our lives over the past few years.

I wish to express sincere gratitude to my supervisor Jules Tapamo for his support and guidance, and to my co-supervisor Johnson Kinyua for his support and guidance through the early stages.

I would like to thank my parents, my family and my dear friends who have supported me. A special thank you to Anban Pillay for being a dear friend, and for willing to sacrifice many hours for proof reading.

I wish to also acknowledge and thank: Chetna Parbhoo for designing and implementing the second case study application as part of her Masters research; Pravi Moodley who assisted with the proof reading; members of the ICT4EO and KSG groups at the Meraka Institute, CSIR, Pretoria, specifically Ingo Simonis and Tommie Meyer.

I also acknowledge the financial support provided by the National Research Foundation and the German Academic Exchange Service (DAAD) .

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AA	Adapter agent
ABox	Assertional Box
ACL	Agent Communication Language
AFIS	Advanced Fire Information System
AIGA	Agent Based Imagery and Geospatial processing Architecture
API	Application Programming Interface
ASCML	Agent Society Configuration Manager and Launcher
BDI	Belief Desire Intention
BN	Bayesian Network
BNJ	Bayesian Network tools in Java
BT	Brightness Temperature
CA	Contextual Algorithm
CPT	Conditional Probability Table
CWA	Closed World Assumption
FD	Fire Detection
FIPA	Foundation for Intelligent Physical Agents
FS	Fire Spread
GEO	Group on Earth Observations
GEOSS	Global Earth Observation System of Systems

GIS	Geographical Information System
GUI	Graphical User Interface
HD	Hotspot Detector
IrisNet	Internet-scale Resource-Intensive Sensor Network Services
IWMAS	Internet Wide Multi-Agent System
JPL	Jet Propulsion Lab
JTS	Java Topology Suite
MAS	Multi-Agent System
MASII	Multi-Agent System Infrastructure for the Internet
MSG	Meteosat Second Generation
O&M	Observation and Measurement
OBO	Open Biological Ontologies
ODGIS	Ontology Driven Geographical Information System
ODIS	Ontology Driven Information System
ODMAS	Ontology Driven Multi-Agent System
OGC	Open Geospatial Consortium
OWL	Web Ontology Language
OX-Framework	OGC Web Service Access Framework
POC	Plant Ontology Consortium
QoS	Quality of Service
RA	Registry Agent
RCC	Regional Connection Calculus

RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SAS	Sensor Alert Service
SensorML	Sensor Model Language
SOS	Sensor Observation Service
SPS	Sensor Planning Service
SUMO	Suggested Upper Merged Ontology
SWAP	Sensor Web Agent Platform
SWE	Sensor Web Enablement
SWRL	Semantic Web Rule Language
TBox	Terminological Box
TML	Transducer Model Language
URI	Uniform Resource Identifier
WNS	Web Notification Service
WSML	Web Service Modeling Language
WSMO	Web Service Modeling Ontology
WSMX	Web Service Modeling eXecution environment
XML	Extensible Markup Language

Chapter 1

INTRODUCTION

1.1 BACKGROUND

Monitoring and understanding our natural environment has become a priority in view of the devastating effects of climate change. Access to high quality information is critical to minimise the impact of natural disasters and to make decisions that ensure sustainable human interaction with the natural environment. Many countries have recognised the importance of sharing earth observation data and monitoring the earth as a continuous system. Earth observation data such as satellite imagery which was previously expensive or inaccessible is increasingly available for non commercial purposes at no or low cost. The Group on Earth Observations (GEO) is an intergovernmental group that was formed in 2005 to build a Global Earth Observation System of Systems (GEOSS) [83]. GEO aims to provide a global computing platform for sharing earth observation data and for monitoring the earth as a continuous system.

The Sensor Web [77, 123] provides a more ambitious vision than GEOSS. It involves constructing a worldwide computing infrastructure that enables sharing, analysis and distribution of earth observation data. The Sensor Web will be especially beneficial for developing dynamic real time monitoring applications. In such applications data transformation, data analysis, and information extraction algorithms are assembled into executable workflows. These workflows are continuously applied to sensor data to extract and deliver relevant information to decision makers. The algorithms, theories and knowledge used to compose individual workflows can be captured, shared and reconfigured for reuse in other workflows. This can ease the development of new applications and facilitates scientific experimentation.

Many challenges must be overcome before a worldwide Sensor Web becomes a reality. A publicly accessible distributed computing infrastructure is required where heterogeneous sensor services and complex end-user applications can be deployed, automatically discovered and accessed. Since services will be developed separately by different organisations, heterogeneity must be overcome for services to interoperate. The geospatial community has specified a set of standard services and common data formats and encodings [31]. However, semantic interoperability and the management of dynamism are outstanding challenges. Advancements in sensor technology will result in the availability of new and higher quality data sets. Current theories, algorithms and data models will continuously evolve in line with our understanding of the natural environment. Monitoring applications should ideally be reconfigured and redeployed to incorporate the latest data and theories.

The most difficult challenge is context-based information extraction. Users may be overwhelmed by the scale and complexity of sensor data. Considering the rate at which new data is being generated, manual querying, integration and interrogation is not sustainable. The technical skill and time required to extract appropriate information from sensor data may form a barrier to a potentially large end-user community who could benefit from this data. Users should ideally be presented only with information that can aid them in their task. Depending on their requirements (context), users may be interested in different aspects of the data or may require different views of the same data.

Two emerging technologies that have shown promise for overcoming these challenges are ontologies and software agents. Agent researchers propose the use of software agents as logical abstractions to model and manage software components in large scale, dynamic and open environments [101, 186, 203, 212]. Software agents are autonomous software components that communicate at the knowledge level [73, 101]. Autonomy is exhibited in two ways: firstly the agent functions independently without continuous intervention by its human owner; and secondly an agent represents the interests of and acts on behalf of its owner. Owners will dictate the underlying technology used to build their agents as well as the behaviour and goals of their agents. In order to communicate at the knowledge level, the content and types of messages exchanged between agents must have well defined semantics that are specified within a pre-agreed knowledge model or ontology. These ontologies explicitly specify the meaning as well as the relationships that exist between the concepts that are used in agent interactions. Communicating at the knowledge level allows agents to overcome system and syntactic heterogeneity.

Software agent technology has been around for more than a decade [75, 206] and has been successfully used for a variety of applications [102, 157, 160, 205]. However, the promise of widespread deployment of software agents remain elusive [90, 126]. A key bottleneck is the sharing and distributing of knowledge [90]. This requires a specific user community to first agree on, build and maintain a common knowledge model. Sensor Web users represent a large global user community that is driven by an urgent requirement to share earth observation data and services. This community can justify the effort required to develop and agree on a shared and explicit knowledge model.

1.2 PROBLEM STATEMENT

A worldwide Sensor Web must cater for the continuous deployment and modification of geospatial data, knowledge, data processing and predictive modeling services by a wide user community that have different requirements, skills and backgrounds. These services must be discovered and assembled in different configurations to extract information, to test theories and ultimately to capture and to advance our knowledge and understanding of the natural environment. It must also support the construction and deployment of real time end user alerting and monitoring applications that incorporate these services. Applications must be easily modified to reflect new service offerings in order to provide relevant and accurate information to decision makers. Ontologies have shown promise as a technology for sharing and integrating data in open environments, while software agents provide mechanisms to dynamically discover, invoke and assimilate these services. Current ontology and agent based approaches either have limited support for describing services, data and theories, or limited support for building, deploying and reconfiguring end user applications.

1.3 EXPECTED IMPACT

An Ontology Driven Multi-Agent System (ODMAS) Sensor Web has the potential to forever change the way in which geospatial data and knowledge is accessed and used. Potential benefits of the approach include:

- promoting the sharing and reuse of data, knowledge and services
- facilitating human collaboration and scientific experimentation
- reducing information overload and system complexity
- managing both data and system dynamism

- increasing automation and machine intelligence

An ODMAS Sensor Web can provide specific benefits to a wide range of users in the earth observation community. Decision makers can access, manage and visualise information provided by real time monitoring applications. Earth observation scientists can capture and share earth observation data and knowledge, and use the Sensor Web as a platform for experimentation, collaboration and knowledge discovery. Developers can design, develop and deploy consistent agent based Sensor Web services and end user applications.

This research attempts to investigate practical issues around creating an ODMAS for the Sensor Web and is intended to provide a first step towards this vision. The ideas presented can also be used to build ontology driven multi-agent frameworks for other domains.

1.4 THE SWAP FRAMEWORK

The major contribution of this research is a framework that tightly integrates ontologies and agents, i.e. the Sensor Web Agent Platform (SWAP). SWAP addresses both ontology construction and use as well agent based design, implementation and deployment. It provides a semantic infrastructure with a set of ontologies and associated reasoners; an abstract architecture that guides the design of agent based Sensor Web applications; an internal agent architecture to guide the internal operation of an ontology driven agent; as well as a flexible multi-agent system (MAS) infrastructure that eases the implementation, deployment and execution of individual agents.

1.4.1 Semantic infrastructure

The semantic infrastructure aims to bridge the semantic gap between machine and human by providing a common, but dynamic modeling framework. It delineates conceptual ontologies for modeling and representing observations and theories about the physical world from technical ontologies for modeling and representing the software entities (agents) that will host and process these observations and theories. The conceptual ontologies are based on the three systems of knowledge used for human cognition, i.e. theme, space and time [132]. An additional system for representing and reasoning about uncertainty is introduced. These four systems simplify the conceptual modeling of complex real world observations.

They provide an holistic approach to capture the different aspects of observations and theories. The technical ontologies provide support for describing the services offered by different agents and the agent interactions used to invoke these services. Support is also provided for constructing complex information processing chains or workflows that may be stored, shared and executed on demand. Since service descriptions and data models are captured within shared ontologies, they become dynamic entities that can be accessed, queried and modified at runtime. Selected services can be assembled into different configurations to form complex executable workflows that may be deployed as new composite services. This approach facilitates interoperability between agents, and between agents and humans. It also allows for data models and service offerings to change, and evolve naturally with minimal impact on and without having to re-engineer the system.

1.4.2 MASII: An Internet Wide Multi Agent System middleware

The Multi-Agent System Infrastructure for the Internet (MASII) provides an agent development, execution and deployment platform. MASII provides an agent transport layer for agent communication, an agent execution model and a flexible adapter based framework that facilitates application and service deployment. MASII delineates between static core infrastructure services that are domain and application independent and an application infrastructure that customises and extends these services for specific applications. The concept of an application adapter that contains application specific components, such as ontologies, interaction protocols and message handlers, is a key feature that is introduced. The adapter architecture allows for the continuous change and deployment of application components. Application adapters can be discovered, downloaded and installed at runtime (see section 3.2). MASII provides the underlying agent middleware services for SWAP agents. This allows SWAP developers to focus on developing and deploying application adapters without requiring detailed knowledge of lower level agent infrastructure services.

1.4.3 Framework for designing and developing Sensor Web agents and applications

The SWAP development framework includes an abstract architecture and internal agent architecture that guides and eases the design and development of ontology driven agents and applications. The architecture facilitates agent reuse, agent service composition, as well as provenance and information extraction.

It specifies three different abstraction layers and six logical agent abstractions for designing and deploying complex Sensor Web applications. The internal agent architecture guides the implementation of ontology driven agents. It includes a data mapping API (see section 4.7.1) that allows ontology instance data received by other agents to be converted into standard geospatial Java objects. This allows an agent developer to incorporate open source libraries or even remote geospatial web services [31] to perform the internal agent processing. In this way SWAP attempts to incorporate both the declarative programming paradigm of the agent and ontology community and the imperative programming approach of the object oriented and web services community.

1.4.4 Application case studies

The semantic infrastructure together with the abstract architecture, internal agent architecture and MASII middleware platform provide comprehensive support to facilitate the design, development and use of ontology driven Sensor Web agents as well as agent based alerting and monitoring applications. The development of two satellite image processing applications, i.e. wildfire detection and informal settlement monitoring, is used to demonstrate the efficacy of the SWAP framework.

These applications show:

- the effectiveness of ontologies to model and represent the different conceptual and system entities in a practical Sensor Web application
- how to implement and deploy agents and applications that are driven by these ontologies
- how to assemble agent services into processing workflows that may be shared, modified, executed on demand and incorporated within end user applications
- how ontologies can be used to govern the behaviour of agent services and workflows and to dynamically reconfigure an application
- how ontologies can facilitate interoperability between agents and users

1.5 ORGANISATION OF THE THESIS

The thesis is organised as follows: A review of current research on the Sensor Web, agents and ontologies is presented in Chapter Two. In Chapter Three, the design and implementation of a multi-agent system infrastructure for the Internet is described. Chapter Four describes an Ontology Driven Multi-Agent System for the Sensor Web, the Sensor Web Agent Platform. The support for representing and reasoning about uncertainty is described in Chapter Five. The development of two satellite image processing applications is described in Chapter Six. Chapter Seven provides a discussion, outlines areas that require future work and draws some conclusions.

Chapter 2

LITERATURE REVIEW

A single worldwide Sensor Web will be a large scale open environment where heterogeneous systems spanning organisational boundaries, must interact and operate effectively within rapidly changing circumstances, with dramatically increasing quantities of available information, while still maintaining individual autonomy. Two important research activities are attempting to address the complexity and challenges of developing and deploying applications in such environments. The first is research on open multiagent systems and the second is on ontology driven information systems. This chapter describes the vision of, and current efforts towards a worldwide Sensor Web. It also provides an overview of software agents and ontologies, their use in current Sensor Web approaches and the limitations of these approaches.

2.1 THE SENSOR WEB

Advances in sensor hardware and wireless communication technology have resulted in smaller, more accurate and cheaper sensors that can be easily deployed in most environments. Sensor technology is not just confined to scalar data measurements of the physical environment such as temperature, pressure and humidity, but also for audio and visual data [18]. Sensor data can be used for a wide variety of applications: in the military for surveillance and target tracking [29, 103]; for civilian applications such as locating and monitoring parking [77, 201], traffic monitoring [123] and visual scene surveillance [17]; for observing natural phenomena such as observing coastlines [41]; weather forecasting [22], measuring snowpack [56] and monitoring volcano eruptions [47]. The resultant increase in sensor data has triggered research activity into the development of infrastructure support services to collect, aggregate and process sensor data for diverse monitoring applications [18, 153, 209].

The concept of a Sensor Web first originated at the Jet Propulsion Lab (JPL) at NASA [55]:

> ”... a system of intra-communicating spatially distributed sensor pods that can be deployed to monitor and explore new environments.”

which they have recently revised [56]:

> ”... is a type of wireless network of sensors that acts as a single, autonomous macroinstrument. It is a temporally synchronous, spatially amorphous network, creating an embedded, distributed monitoring presence.”

From this view the Sensor Web is considered to be a network, where sensors cooperate towards achieving a common goal. It allows for many different isolated Sensor Webs, each separately deployed and controlled and not necessarily accessible over the Internet.

An alternative view, and the view taken in this research, is of *a single worldwide Sensor Web* [77], where sensor data is made accessible via the Internet. Users are able to query vast quantities of data generated by thousands of widely distributed heterogeneous sensors. A broader definition is given by Liang et al. [123].

> ”The Sensor Web is a revolutionary concept towards achieving a collaborative, coherent, consistent, and consolidated sensor data collection, fusion and distribution system. The Sensor Web can perform as an extensive monitoring and sensing system that provides timely, comprehensive, continuous and multi-mode observations.”

The Open Geospatial Consortium (OGC), a global industry consortium representing over three hundred organisations, is attempting to standardise the encoding and exchange of geographical data. The OGC view the Sensor Web as being a single global infrastructure that supports accessing sensors over the Internet. They define [31] a *sensor network* as :

> ”... a computer network consisting of spatially distributed autonomous devices using sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants, at different locations.”

and then define a Sensor Web as:

"... a web accessible sensor networks and archived sensor data that can be discovered and accessed using standard protocols and application programming interfaces (APIs)."

The Sensor Web is essential for monitoring and understanding our natural environment and to make decisions based on sound information that ensures sustainable human interaction with the natural environment. The importance of sharing earth observation data in order to monitor the earth as a continuous system is now being recognised and prioritised by many countries. Governments have already committed to sharing earth observation data. In 2005 the intergovernmental Group on Earth Observations (GEO) was created to build a Global Earth Observation System of Systems (GEOSS) to facilitate this [83].

2.1.1 Our vision of the Sensor Web

Our view of the Sensor Web follows that of Gibbons et al. [77], of a single worldwide Sensor Web, and falls between Liang's and the OGC definition:

The Sensor Web is a worldwide, Internet based computing environment that allows for the dynamic exchange, integration and analysis of earth observation data produced by multiple, globally distributed sensor sources to satisfy diverse earth observation monitoring requirements.

Most current sensing systems are designed either for domain specific applications or data collection purposes [125]. While this is significant for military (tactical sensor networks), robotics and other commercial applications, that operate in a closed and tightly controlled environment, it does not facilitate accessing, integrating and analysing the vast ever increasing amount of non-commercial sensor data that is currently available. Our view of the Sensor Web is an infrastructure that allows end users to automatically access, extract and use appropriate information from multiple sensor sources over the Internet (an open environment). The difference between sensors, sensor networks and sensor nodes is illustrated in figure 2.1. Data and tasking capabilities from sensors and sensor networks are exposed publicly via a Sensor Web sensor node over the Internet. Sensors within sensor networks can collaborate to produce data, but the sensor node packages and formats this data so that it is easily discoverable and usable on the Sensor Web. The Sensor Web also contains non-sensor nodes. These nodes provide other services, such as data processing, prediction, coordination and discovery services.

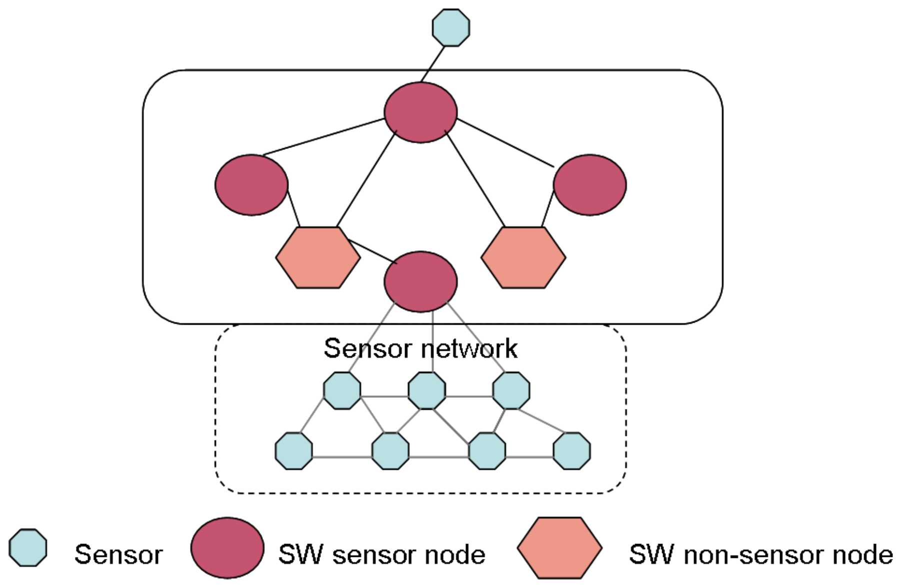


Figure 2.1: Sensor Web vs Sensor Networks

Three technical challenges for realising this vision are: Firstly, to create a publicly accessible open distributed computing infrastructure where heterogeneous sensor resources and complex end-user applications can be deployed, automatically discovered and accessed. The second is the problem of fusing data from different sensors with different temporal resolutions, spatial resolutions, data models and data formats. By fusing data from different sensors a higher spatial coverage and temporal resolution is achieved. The last challenge is to perform context-based information extraction. The technical skill and time required to extract appropriate information from sensor data provides a barrier to a potentially large end-user community. Users do not want to deal with the complexity and scale of sensor data, but would prefer a view of the data that only exposes information that can aid them in their application. Depending on their needs (context), users would be interested in different aspects of the sensor data. Thus, the same data could be used for various applications. Since sensor data forms the core of the Sensor Web, the challenges can also be viewed from a data centric standpoint [25], where essential infrastructure services for information management are identified.

Many initiatives are underway to build the Sensor Web. Two of these initiatives, both based on web services [100], are described below.

2.1.2 OGC Sensor Web Enablement

The Open Geospatial Consortium (OGC) [8] aims to provide publicly available standard interface specifications for accessing geographical data, and a standard way of encoding and exchanging this data over the Web. It uses a community driven, consensus based approach to develop standards. This promotes the uptake of the standards which makes them pragmatic and usable. OGC Web Services are based on a web services framework [12, 100]. It supports publishing, discovering and accessing geographical resources over the web. The Sensor Web Enablement (SWE) [31] initiative extends the OGC Web services by providing additional services for integrating Web-connected sensors and sensor systems.

2.1.2.1 The Sensor Web Enablement (SWE) Framework

SWE [31] currently defines four specialised Web service specifications, and three XML based models and encodings for observations and sensors respectively. The Sensor Observation Service (SOS) is used for data access; the Sensor Planning Service (SPS) is used for sensor tasking and feasibility studies; the Sensor Alert Service (SAS) is used for registering atomic conditions and push based notification; and the Web Notification Service (WNS) is used as a data transport protocol transformer. The Observation and Measurement (O&M) and Sensor Model Language (SensorML) are used as data and metadata exchange protocols. The Transducer Model Language (TML) provides a conceptual model and Extensible Markup Language (XML) schema for describing transducers and supporting real-time streaming of data to and from sensor systems.

2.1.2.2 Analysis of the SWE approach

SWE mostly addresses the first two challenges mentioned above. It is based on a web services architecture which allows for publishing and discovering services over the Web. However, the SWE approach is limited by its support for automatic data analysis, scalability and interoperability. It provides a common data model and data encoding format for fusing multiple data models and formats but it lacks an explicit formal conceptual model or ontology [84]. A key assumption is that all users will interpret and commit to the same meaning of the terms provided in the vocabulary. Even though SensorML can be used to describe sensors, there is no standard semantics for describing the sensors or the phenomena that it measures. This limits service discovery, data integration and service interoperability.

SWE abstractions for designing services are too broad, which limits service reuse and changeability of services. While SWE provides abstractions for distinguishing between service types, these abstractions are too broad. For example, a Sensor Observation Service can be used to offer unprocessed sensor data, semi processed or derived data as well as complex information extracted from the data. Service providers will typically hide complex application logic behind a single OGC service, even if the functionality could be decomposed into reusable services. This makes it difficult to modify or replace hardwired data retrieval and analysis mechanisms embedded within the service which limits reuse. As dependencies on other services are hardwired, applications must be manually upgraded when dependant services change and newer services appear. This problem is more apparent when the number of services increase, which questions the scalability of the SWE approach. Users may be aware of individual instances of OGC services, but may not be aware of the potential applications for which these services can be used. As the design and development of end-user applications are not specifically dealt with in the SWE framework, applications will be developed in an adhoc manner. In the absence of an explicit application framework more effort will be required to build, deploy and maintain new applications that incorporate SWE services.

Despite these drawbacks, the Open Geospatial SWE framework serves as a starting point for a single worldwide Sensor Web framework. It provides an open, standards based architecture that allows sensor and data resources to be published, and accessed in a standard way simply by implementing public interface and encoding specifications. Since the OGC consists of leading industry players and organisations that work with geographical information, and SWE is based on a consensus approach, SWE technologies and standards have the greatest chance of being adopted by major software vendors and organisations. SWE is arguably the most important step towards realising the Sensor Web. Applications using SWE implementations, such as the 52 North implementation [1], have already been deployed [138].

2.1.3 GeoSwift

Liang et. al [123] proposes GeoSWIFT Sensing Services as a distributed geospatial information infrastructure for the Sensor Web. The framework consists of a three layered architecture comprising of a *sensor layer*, i.e. the actual sensors, a *communication layer*, i.e. the physical network or data communications link between the various components and the *information layer*, that provides interoperability and integration of data from different sensors. The implementation prototype uses a web services approach for service registration and discovery. The architecture advocates use of OGC standards for integrating and exposing sensor data. An extension to the traditional web services approach is the introduction of a sensing server. The sensing server provides the information layer, which is able to integrate and store data in different formats from different sensors. It abstracts sensor specific data formats and communication protocols from the user. New sensors can be integrated into the system by extending the sensing server or deploying a new server. The sensing server provides a number of benefits to users. It hides sensor specific configurations and data formats, and provides a standard interface through which users can interact with different sensors. However, as with SWE, it lacks semantics and does not explicitly provide a mechanism for developing and deploying applications.

2.2 SOFTWARE AGENTS AND MULTIAGENT SYSTEMS FOR INTERNET COMPUTING

The Sensor Web is as an *open system* [93]. It will cross organisational boundaries, and comprise of systems that are developed separately and independently often by different and unrelated organisations. Individual computing nodes can be characterised [181, 186] as being: *autonomous*, i.e. they have their own goals, dictated by their host organisation, that may conflict with the goals of other nodes; *cooperative*, i.e. they are prepared to work together for specific tasks; *heterogeneous* i.e. the nodes in the system can use different technologies or data representation formats; and *dynamic* i.e. the individual nodes can and will fail in the environment and nodes will continuously change their configurations, abruptly leave the system and new nodes can enter the system. Software developers face serious challenges when using traditional software development paradigms, such as object oriented programming, to design and implement applications in these large scale, open environment. Agent researchers attempt to address these challenges by providing better logical abstractions and interaction mechanisms for software components in complex, large scale and dynamic environments [101, 186, 203, 212].

There are many perspectives on agents and multiagent systems [146, 186, 192, 198, 205]. This work is restricted to Internet agents [162, 186, 212], i.e. autonomous software entities that communicate over the Internet. Since information is a key commodity on the Internet, extensive work on Internet agents has been carried out as Information agents [54, 108, 109, 110, 111]. An information agent is [110]:

"... an autonomous, computational software entity that has access to one or more heterogeneous and geographically distributed information sources, and which is able to pro-actively acquire, mediate, and maintain relevant information on behalf of its users or other agents preferably just-in-time."

An earlier narrower definition considers agents as being software components that communicate using a high level agent communication language [75]. In this work, an (Internet) agent is considered to be an autonomous software component that communicates at the knowledge level [73, 101] over the Internet. To communicate at the knowledge level, the content and type of all messages exchanged between agents must have well defined semantics that are specified in a shared knowledge model or ontology (see section 2.3) that is accessible to both the receiver and the sender. Autonomy is exhibited in two ways. The first being the capability of the agent to function independently without continuous human intervention. The second is that different agents have different owners, either individual human users or organisations, whose interests the agent represents. Owners dictate the underlying technology used to build their agents, and the behaviour and goals of their agents (often independently of other owners). Goals may differ or conflict.

In general, agents can provide the following advantages [186]:

- *Speedup and efficiency*, due to asynchronous and parallel computation.
- *Robustness and reliability*, the whole system can undergo a graceful degradation when one or more agents fail.
- *Scalability and flexibility*, since it is easy to add new agents to the system.
- *Cost effectiveness*, assuming that an agent is a low-cost unit compared to the whole system.
- *Development and reusability*, since it is easier to develop and maintain modular software than monolithic ones.

2.2.1 Agent operation

In an open Multi-Agent System (MAS) an agent must be able to discover other agents, as well as interpret and respond to messages received from other agents.

2.2.1.1 Agent discovery

In an open MAS agents enter and exit the system dynamically and unpredictably [181]. In a large scale open MAS [203], it is not possible for agents to be aware of the capabilities of all other agents in the MAS. The infrastructure must provide mechanisms for agents to advertise their capabilities and for agents to discover the capabilities of other agents. The agent discovery mechanism used depends on privacy considerations, on robustness and on scalability and performance [26]. Two mechanisms for agent location are middle agents [54] or peer-to-peer location methods [120, 148]. Middle agents [54] maintain a registry of agents that wish to publicise their capabilities. Middle agents can serve as broker agents [202], directory facilitators [69] or as a blackboard. A broker agent receives a request, forwards the request to an appropriate agent, and returns the results to the requester. A directory facilitator or matchmaker acts as a yellow pages. It stores capability advertisements and returns matching advertisements to requestors. The requesting agent then contacts the appropriate agents directly. A middle agent can also serve as a blackboard on which agents advertise requests and other agents search and fulfill these requests according to their capabilities. However, the middle agent approach may present a bottleneck as well as a central point of failure. To alleviate this, decentralised peer-to-peer location methods have been proposed [120, 148]. Service description languages such as OWL-S (see section 2.3.1.7) provide a domain independent language in which service agents can specify their capabilities. Agent service composition has also been investigated, using OWL-S [129] as well as using finite state machines [196].

2.2.1.2 Agent communication

A key aspect of agents for open environments is the ability to communicate at the knowledge level [73, 101] generally using some high level Agent Communication Language (ACL) [75, 114, 118]. This removes and allows for disparities in implementation and internal representation of knowledge. Individual agents can use different representations of internal models of the world and different implementation technologies. If the communication is independent of these, there is a looser coupling between agents. ACLs, protocols and conversational policies [81] are an essential part of a MAS. An ACL specifies the syntactic structure of messages as well as the semantic interpretation of the messages so that an agent understands the contents of the message. The semantic interpretation relies on the specification of a shared ontology (see section 2.3) in which all terms used in the messages are defined. The MAS infrastructure stores shared ontologies, while individual agents can have their own private ontology. For meaningful conversation to occur, the recipient of a message is only able to interpret the message content if the recipient has access to the ontology used by the sender.

An interaction protocol [180] specifies patterns of interaction between agents by specifying the sequence in which messages can be exchanged for specific purposes. A protocol allows agents to know how to respond to messages and to have more meaningful and complex interactions. Furthermore a message can be interpreted by an agent in the context of an overall conversation and not a single isolated message. The Foundation for Intelligent Physical Agents (FIPA) ACL [68] is the most widely known agent communication language. It provides formal semantics for seven distinct domain independent communicative acts (based on speech act theory [175]), which are used in eleven standard agent interaction protocols. It is assumed that all agents are aware of these protocols. Thus a receiving agent is able to unambiguously interpret the intentions of the sending agent. However, it is unlikely that the protocols in a large scale and open MAS will be static. Other approaches attempt to allow agents to communicate using protocols of which they had no prior knowledge, e.g. using message templates [158] and ontologies [51].

2.2.1.3 Internal agent architecture

Agents require some internal mechanism to initiate actions in the system. These mechanisms may be deliberate, reactive or hybrid. Deliberate agents maintain an internal model of the world, which is usually based on symbolic logic. The model is used for planning and to initiate actions to achieve their goals. The most popular deliberative agent architecture is the Belief Desire Intention (BDI) architecture [33, 166]. Beliefs reflect the agent's world model, while desires specify their purpose or end state. An agent's goals are a non conflicting subset of the desires. An agent's intentions is its commitment to undertake a series of actions that are planned in order to achieve its goals. Given the recent emergence of inference rules in the Semantic Web, rule based agents are starting to appear [115, 142]. Alternative agent architectures [198, 205] include reactive architectures such as Brooke's subsumption architecture [36] and hybrid architectures such as InteRRaP [141].

2.2.2 MAS infrastructure models and platforms

An Internet Wide MAS (IWMAS) must be a large-scale open agent system where tens of thousands of agents must find and interact with each other for various applications [203]. The numbers and availability of agents, including the communication languages, ontologies and types of applications may vary over time.

From a technical viewpoint, a MAS infrastructure is a domain independent set of services, conventions and knowledge that allows agents to discover and communicate with each other [188]. A more encompassing definition of a MAS infrastructure is given by Gasser [74] in terms of MAS technology, its application and use, as well as MAS scientific and educational activities. Two MAS infrastructure architectures, the FIPA abstract agent architecture and Sycara's abstraction hierarchy are described below.

2.2.2.1 FIPA abstract agent architecture

The most widely known open MAS architecture is the FIPA architecture. The Foundation for Intelligent and Physical Agents [1] (FIPA) is a standards body, consisting of organisations from both academia and industry that aims to standardise MAS infrastructure services, so that heterogenous agents can communicate and interoperate effectively. FIPA provides an abstract architecture [66] that specifies the minimum components of a heterogenous intentional agent system: a *message transport* for agents to transmit messages to each other, an *agent directory* and a *service directory* for agents and services to register themselves and to be discovered by other agents; and a standard *agent communication language*, FIPA ACL [68] with formal semantics, so that the intentions of the sending agent can be unambiguously interpreted by the receiving agent.

JADE The JADE [6] agent toolkit is a robust implementation of the FIPA abstract architecture. This Java based open source agent platform is well documented and is probably the most widely used agent platform. It rigidly follows the FIPA standards and is one of the reference architectures for FIPA. JADE has limited support for Semantic Web technologies. There are extensions in this regard. AgentOWL [119] provides support for OWL ontologies using JADE agents. Another extension [142] provides support for inference rules (see section 2.3.1.6).

[1] http://www.fipa.org

AgentScape, another MAS infrastructure, attempts to deal with scalability issues in an Internet scale Multi-Agent System. Bordini [30] provides a broader review of agent programming languages, IDEs and platforms.

2.2.2.2 Sycara's abstraction hierarchy

Sycara provides an in-depth study of the practical functional requirements of a MAS infrastructure and proposes a layered abstract hierarchy of nine services that a MAS infrastructure should offer [188]. These are an *operating environment*, *communication infrastructure*, *ACL infrastructure*, *multi-agent management services*, *security*, *agent name to location mapping (ANS)*, *mapping capabilities to agents* and *MAS interoperation*. An individual agent requires modules to use each of these services. Sycara's infrastructure model provides underlying technical services that enable agents to discover and communicate with each other. However, it does not include explicit support for governing agent interactions. Omicini et al [149] pointed out that the role of infrastructure is not only to model and shape the agent environment from an individual agent's point of view (subjective view) but also from a MAS designer's point of view (objective view). The infrastructure should provide runtime abstractions to model MAS applications. It must describe and enforce interaction constraints, coordination laws and social norms and allow MAS designers to design and embed elements of control using these abstractions in the infrastructure. This reduces the gap between, design, engineering and operation. It not only allows designers to better observe and understand the runtime behaviour of the system but also facilitates incremental changes to applications using these abstractions.

Sycara's model does not explicitly distinguish between the environment, the MAS infrastructure and individual agents. Weyns et al. [200] propose an abstract model of a MAS infrastructure that treats the environment as a first class abstraction and that clearly distinguishes environment responsibilities from agents responsibilities. Valckenaers et al. [195] provide different environment configurations for three classes of applications: simulation applications, applications that interact with the physical world and applications that operate within a virtual world.

The need for abstractions to assist in designing MAS applications is clear. However, the environment may be also incorporated into the infrastructure. The environment provides a set of trusted services that can be made available to multiple agents. This functionality may be incorporated into the services provided by the infrastructure together with the necessary abstractions.

2.2.3 Challenges for building an Internet Wide MAS (IWMAS)

While there is an active software agent research community there is no evidence of the imminent widespread use of MAS technology as was promised a decade ago [90]. Several challenges must be overcome before the vision of an IWMAS is realised. Mass deployment of agent applications has been hindered by: the complexity of agent platforms; to some extent the overhead of complying with standards, e.g. Foundation for Intelligent and Physical Agents (FIPA), and the lack of industry strength design and development methodologies and supporting integrated development environments and toolkits. Current MAS models seem to focus more on individual aspects of agents systems or proving a specific methodology than on practical application development and deployment in open environments. While these activities are important, failure to design and field an Internet scale MAS infrastructure hinders widespread MAS usage. Gasser [74] put this clearly by stating:

> " ... widespread use of MAS won't occur until a critical mass of fielded systems, services and components exists . . . " and ". . . until we have a stable, operational, widely accessible, and low-apparent cost MAS infrastructure populated with a critical mass of MAS services and systems that provide value to prospective users; MAS is likely to languish as a technology with potential, not kinetic, energy."

This is further emphasised by Sycara [188] and by Luck et al. [126] in their review of Agent Technology. They suggest that even though there are many fielded systems, these are largely designed by one design team, in one corporate environment and with agents sharing high level goals within a single application domain, i.e. a more closed than open system. Furthermore, most approaches are based on a static infrastructure. Agent researchers have attempted to separate the dynamic parts of the infrastructure and call this the agent environment [195, 200], but our view is that in a true IWMAS, the infrastructure itself should be designed for change and be allowed to evolve in line with its usage. Currently, to our knowledge, there is no MAS platform designed specifically to provide a global infrastructure for designing, developing and deploying applications across multiple application domains on the Internet.

In the context of this work, two specific aspects of an IWMAS that must be addressed, viz. support for deployment, and an explicit tightly integrated semantic infrastructure.

2.2.3.1 Deployment

Due to the limited number of fielded open multi-application MASs the issue of deployment has been neglected. In an IWMAS where different design teams will continuously modify existing agents and deploy new agents, it is crucial to provide guidelines for agent and application deployment. These guidelines will ensure that a properly developed distributed application can be packaged into a reusable, maintainable, and configurable piece of software. Deployment in an IWMAS is a considerable challenge. New applications should reuse and integrate with existing agent functionality where possible, but should not negatively alter the behaviour of existing applications. An application must also be easily reconfigurable to cater for the dynamism of the agents that are incorporated into the application. There is a lack of tools for the launching and dynamic reconfiguration of complex agent based applications [34]. To deal with this, Brauback et. al [34] propose high level abstractions for applications in the form of MAS societies. Each society has a specialised configuration manager and launcher agent, an ASCML (Agent Society Configuration Manager and Launcher) agent, which is responsible for controlling and managing agents that belong to that society. The use of a configuration management agent has also been proposed in a deprecated FIPA specification [67].

2.2.3.2 Lack of an explicit semantic infrastructure

A bottleneck in agent based systems is sharing and distributing knowledge [90]. Agent researchers have neglected the ontological infrastructure [146].

In an IWMAS shared ontologies facilitate agent discovery and selection, and give meaning to agent interactions. However, the development and management of these ontologies is not addressed in most MAS platforms and systems, and is delegated to the application developer. The FIPA ontology agent specification [65] specifies the requirements of an ontology management agent, but the design is incomplete and has very few implementations. As pointed out by leading agent researchers, it is essential that agents build on the successes of the Semantic Web community [38, 99, 100, 126, 181, 197] which has a more limited scope and focus than the agent community. The Semantic Web community has made significant advances in standardising knowledge representation and reasoning, and providing tools that support these technologies. It is crucial that technologies such as OWL and rules be integrated with agents based systems, so that agent based systems can leverage the continuous advances in the Semantic Web community.

2.3 ONTOLOGIES AND THE SEMANTIC WEB

A commonly cited definition of an ontology is by Thomas Gruber [84]. Gruber investigated the use of ontologies for communication in knowledge based systems where agents communicate using statements in a formal knowledge representation language. Gruber distinguished between the representation language format, agent communication protocol and the specification of the content of shared knowledge, with the first two being independent of the content being communicated. Gruber defines an *ontology* as:

> ”... an explicit specification of a conceptualisation.”

The aim of this explicit specification is for sharing this conceptualisation in order for agents to have a basis for communication, i.e. a common conceptualisation for knowledge sharing among distributed programs. The reason for this explicit formalisation is to prevent ambiguity when interpreting messages from agents.

Guarino [86] later pointed out that different entities may have different conceptualisations about the same reality and defines an ontology as:

> ”... being constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words.”

In the simplest case an ontology describes a hierarchy of concepts related by subsumption relationships. In more sophisticated cases, suitable axioms are added in order to express other relationships between concepts or to constrain their intended interpretation, i.e. to remove ambiguity. However, even if two systems adopt the same vocabulary, this does not guarantee that they commit to the same conceptualisation. If each system has its own conceptualisation then a necessary condition to make agreement possible is that the models of the relevant conceptualisations overlap.

Other simpler definitions exist, such as: an ontology is a formal definition of a body of knowledge [89] or a definition by Welty [199] who defines ontology as being about meaning more specifically about making meaning as clear as possible. However these definitions are broad and can cause controversy when deciding what is, and what is not, an ontology. Our definition incorporates the purpose of ontologies.

An ontology is an explicit description of a conceptualisation (body of knowledge) that is used for sharing information pertaining to a body of knowledge. The meaning of the elements in the conceptualisation must be clear and unambiguous, to enable both software agents and human users to consistently and correctly interpret all information that follows the conceptualisation and to generate and share their own information using the conceptualisation.

Gruber provides five design principles [84] (also summarised in [80]) for the development of ontologies:

Clarity: communicate effectively the intended meaning of defined terms. Definitions should be objective, complete and documented with natural language.

Coherence: sanction inferences that are consistent with the definitions. If a sentence inferred from the axioms contradicts a definition then the ontology is incoherent.

Extendibility: enable the definition of new terms for special uses based on the existing vocabulary and that avoids the revision of the existing vocabulary.

Minimal encoding bias: be specified at the knowledge level without depending on a particular symbol level encoding.

Minimal ontological commitment: specify the weakest theory and define only those terms that are essential to the communication of knowledge consistent with the theory.

In addition, in keeping with our definition, it is also essential that ontologies are:

Accessible to human users: be designed to specify concepts in a way, that these specifications are understandable by a human user familiar with the domain it specifies and accessible or downloadable by these users.

Accessible to software programs: written in a representation language that allows computer programs to interpret and reason about information that is structured using the ontology and generate new information that follows the structure of the ontology.

The primary purpose of ontologies is to achieve semantic interoperability. Ontologies provide an explicit description of the data exposed in the information system. This allows for information to be

SOCIAL WORLD - beliefs, expectations, commitments, contracts, law, culture, ...
PRAGMATICS - intentions, communication, conversations, negotiations, ...
SEMANTICS - meanings, propositions, validity, truth, signification, denotations, ...
SYNTACTICS - formal structure, language, logic, data, records, deduction, software, file, ...

Figure 2.2: Interoperability framework for open systems [151]

automatically discovered and integrated by machines. Semantic interoperability supports high level, hence easier to use, context sensitive information requests over heterogenous information sources, hiding system, syntax and structural heterogeneity [152, 179]. Ontologies have been used widely to overcome semantic interoperability and information sharing in a distributed computing environment [84, 80, 106, 145, 194]. However, in an open real world system, where owners are globally distributed and have different and often conflicting goals and social and cultural backgrounds, other types of heterogeneity may exist. While ontologies can be used to overcome semantic heterogeneity, there are still other types of heterogeneity that exist in an open system. A more encompassing model that better reflects the types of heterogeneity found in an open real world system is shown in figure 2.2 [151].

The Semantic Web [44, 52, 91] initiative proposed by Tim Berners Lee [27] is one of the main drivers of ontology research. It aims to mark up the content in the billions of web page on the Web so that machines are better able to dynamically process and ”understand” the data that they merely display at present. The use of shared ontologies to add semantics to the content of web pages is central to the vision. In this way software agents (programs) can use these ontologies to recognise, interpret and reason about the content. The real power of the Semantic Web will be realised when people create many agents that collect Web content from diverse sources, process the information and exchange the results with other programs. The effectiveness of such software agents will increase exponentially as more machine-readable Web content and automated services (including other agents) become available. The Semantic Web promotes this synergy: even agents that were not expressly designed to work together can transfer data among themselves when the data comes with semantics.

Progress has been made towards the Semantic Web. This includes the emergence of standard ontology representation languages including tools to create and manage ontologies.

2.3.1 Ontology Representation languages

Ontology languages [80] vary in their expressivity, but typically provide various constructs for representing concepts and relations between these concepts. The following discussion is based on languages used within the Semantic Web community. A discussion of other ontology languages can be found in [80]. Standard ontology languages for the Semantic Web community, such as the Resource Description Framework (RDF) [112], RDF schema (RDFS) [35] and the Web Ontology Language (OWL) [131] are now established with stable tools for creating and managing ontologies.

2.3.1.1 RDF and RDFS

The Resource Description Framework (RDF) [112] is a key knowledge representation framework for the Semantic Web. The underlying structure of any expression in RDF is a collection of triples, each consisting of a subject, a predicate and an object. The predicate asserts a relation between the subject and the object. A RDF statement is a graph with the nodes being the subjects and objects. The meaning of a RDF graph is the conjunction of all the statements corresponding to all the triples it contains. Each node (objects and subjects) and predicate is uniquely identified by a Uniform Resource Identifier (URI). RDF graphs are usually serialised and stored in Extensible Markup Language (XML). While RDF provides a representation framework, RDF schema (RDFS) [35] provides the vocabulary for specifying ontologies or schemas to structure the information captured in RDF. RDFS uses a standard vocabulary for denoting classes (rdfs:Class), properties (rdf:Property) and the domain (rdfs:domain) and the range (rdfs:range) of these properties. Relations such as subclass (rdfs:subClassOf) are also required to represent class hierarchies.

2.3.1.2 OWL

The Web Ontology Language (OWL) [131] is the recommended standard ontology language for the Semantic Web. OWL was designed to address the limitations of RDF and RDFS. OWL builds on RDF and RDFS by incorporating additional vocabulary for describing properties and classes including: relations between classes (e.g. disjointness), cardinality (e.g. ”exactly one”), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes [131]. In OWL, a class is represented by *owl:Class* which is a specialisation of *rdfs:Class*, *rdf:Property* is partitioned into *owl:ObjectProperty* and *owl:DatatypeProperty* to differentiate properties that apply to objects and to literal antetypes respectively. OWL has three sub-languages with increasing level of expressivity, OWL Lite, OWL DL and OWL Full [131]. The tradeoff is between expressibility and efficient reasoning [197]. In general, the more features that we have in a language, the more difficult it is to reason with the language. OWL-Lite is the least expressive. It provides sufficient support for defining most ontologies, but excludes enumerated classes and disjoint statements and only allows for restricted cardinality (0 or 1). OWL Lite is the easiest to reason with as it has a lower formal complexity than OWL DL. OWL Lite should be used where high expressivity is not required, especially for simple class hierarchies or taxonomies. OWL DL extends OWL Lite and has a well defined semantics based on Description Logics [24]. It supports all the OWL language primitives and provides maximum expressiveness while retaining completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). In order to retain decidability OWL-DL restricts how OWL primitives can be used. OWL DL constrains mixing OWL with RDFS primitives and it requires strict disjointness of classes, properties, individuals and data values, e.g. an entity in OWL-DL can not be used both as a class and an instance. OWL Full also uses all the OWL language primitives, but has the fewest restrictions. It allows arbitrary combination of OWL primitives with RDF and RDFS which means that every RDFS ontology is a an OWL-Full ontology. However, this level of expressivity is at the expense of decidability.

2.3.1.3 Open world assumption

One of the key features of Semantic Web languages that distinguish them from other declarative languages such as SQL and Prolog, is the adoption of the *Open World Assumption*. This assumes that knowledge is always incomplete. Thus, if a statement can not be proved to be true using current knowledge it can not be concluded that it is false. In such a case it is concluded that the truth of the statement is unknown. This is in contrast to SQL and Prolog which adopt the Closed World Assumption (CWA) where complete knowledge is assumed. Under the CWA if a statement can not be proved to be true then it is false. Following the open world assumption in the Semantic Web is very important as incomplete information is common and fragments of knowledge are often distributed within multiple ontologies across the Web.

2.3.1.4 Reasoning with OWL

Description Logics (DL) [24] are a subset of first order predicate logics and provide the underlying formal framework for OWL and RDF. A knowledge base in a description logic consists of two parts traditionally referred to as the Terminological Box (TBox) and the Assertional Box (ABox). The TBox consists of several assertions about concepts and roles (the ontology). The ABox consists of specific facts about objects (instances), including statements about an object belonging to a particular concept or a particular pair of objects belonging to a particular role. A reasoner is used to produce additional inferences by applying the rules of the language to statements in both the TBox and the ABox. Most reasoners, e.g. Pellet [182] and Racer [87] incorporate variations and optimisations of tableaux algorithms to perform inferencing.

2.3.1.5 Limitations of OWL

Although OWL can be used for structuring and classifying knowledge it has many limitations. There is weak support for antetypes. More specifically there is no support for custom antetypes [154, 155], e.g. for creating a custom data type *atLeast18* (a value of 18 or more) for restricting a class *Adult*, to be all *Persons* with an age value of *atLeast18*. Furthermore, OWL does not support the use of variables, n-ary predicates [144], sequences such as lists, [59] or metamodeling [140], i.e using classes and properties as individuals. Many of these limitations have been recognised and are being addressed in OWL 2 [79]. Within the context of the Sensor Web, OWL lacks support for representing and reasoning about time, space and uncertainty [177] which are essential for representing geospatial information. OWL also lacks representational support for describing the system entities, i.e. the agents, services and processes, that will form a part of an agent based Sensor Web.

2.3.1.6 Extending OWL with rules

While OWL is good for structuring and classifying knowledge, an agent must still encode knowledge that enables it to process available information and respond to requests and new information correctly. While ontologies provide the ”what” (semantics), i.e. knowing what is being meant by certain concepts, rules can be used to provide the ”how” (pragmatics), i.e. how to process and respond to information [181]. Rules have been recognised as being an important part of the Semantic Web [42, 70, 82, 97].

The simplest type of rule is an inference rule [32, 171], which takes the form of:

$$if\ \textbf{antecedent}\ then\ \textbf{consequent}$$

or

$$\textbf{ascendant} -> \textbf{consequent}.$$

From a given set of facts in a knowledge base an inference engine uses a set of inference rules to draw additional logical conclusions, which result in new facts being added to the knowledge base. A mapping between ontology and rule languages is important for many aspects of the Semantic Web [70, 82]. The implication and conjunctive operators and the support for variables alleviate many of the limitations of OWL. Rules allow for instance reasoning, conjunctive queries and composition relations of properties, e.g. the relation between the brother, parent and uncle can be represented as the conjunctive rule:

$$brotherOf(Z, X)\ \textbf{and}\ parentOf(X, Y) -> uncleOf(Z, Y).$$

Rules can also be used for representing policies and for describing services by specifying relations between service inputs and outputs. The Semantic Web Rule Language (SWRL) [97, 98] has been proposed to integrate rules and OWL. Practical software applications [78, 40] that integrate rules and ontologies are also being developed.

2.3.1.7 Service representation

Semantic web services are an integral part of the Semantic Web [92]. The vision of Semantic Web Services [38, 44] research is to advertise web services so that they can be automatically discovered and invoked by computer programs. In order for this to occur, services must be able to describe the information that they provide and how this information can be retrieved. A number of languages are being developed to extend OWL to describe services.

OWL-S [129] is an ontology language for describing services on the Semantic Web. It provides a standard vocabulary that extends OWL for creating service descriptions. The current trend in service description languages is the representation of services as processes. OWL-S models each independent service as an atomic process and provides flow constructs for assembling atomic processes into composite processes. It also provides a dynamic execution environment for web services.

The Web Service Modeling Ontology (WSMO) [107, 214] provides a framework for the description and the automatic discovery of Semantic Web Services. WSMO distinguishes between ontologies, web services, goals and mediators. Goals specify objectives which could potentially be achieved by executing the appropriate web services, while mediators are used to overcome interoperability problems. WSMO provides a formal specification of concepts for describing web services and is based on the Web Service Modeling Language (WSML), a formal language based on Descriptional Logics and Logic Programming. WSMO has its own execution environment the Web Service Modeling Execution Environment (WSMX). While WSMO takes a more holistic view of describing and discovering services, integrating WSMO into OWL based systems may be challenging.

Other service description initiatives include WSDL-S [122] and FLOW [85].

2.3.2 Ontology development and management

In an open Internet wide system it is naive to assume the existence of a common global ontology. In reality there will be a multitude of ontologies developed by different groups, for different purposes, representing different views and subject to continuous change.

2.3.2.1 Categories of ontologies

An approach to ease ontology development and management is to distinguish between categories of ontologies. As shown in figure 2.3, ontologies can be split into three broad categories or levels. Upper or reference ontologies provide very general concepts or real world notions that are referenced by all root terms in other ontologies. This may include concepts for representing space and time or general concepts such as *Object*, *Money*, *Animal*, *Person* and *Event*. The upper ontologies form a common and consistent high level model that grounds all other ontologies in the system. Several upper ontologies exist, e.g. Suggested Upper Merged Ontology (SUMO) [10] and DOLCE [72].

Domain and domain task ontologies specify terms, relationships between terms and generic tasks and activities that are relevant in a particular domain. Domain ontologies reuse or specialise concepts from

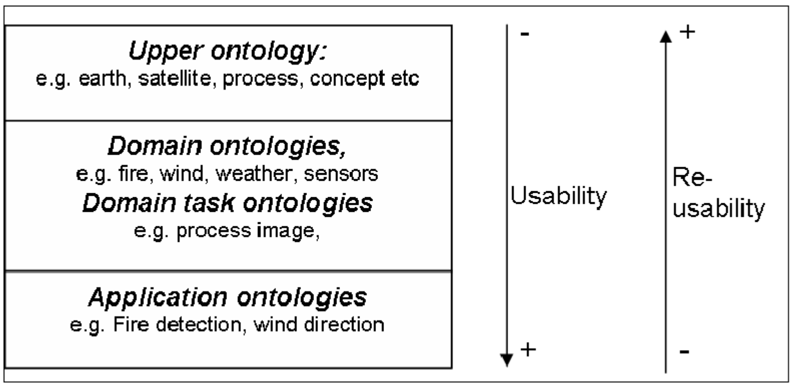


Figure 2.3: Different types of ontologies according to Guarino [86]

the upper ontologies. Thus, domain experts can focus on modeling the specifics of their domain rather than formulating high level models of space and time. The integration of different domain ontologies that extend the same upper ontologies or high level model is also possible [145]. Large scale domain ontologies have already been developed which are being used within several scientific communities to share, integrate and analyse the vast quantities of data that is available in these communities. These include the Open Biological Ontologies (OBO) [7, 183], the Plant Ontology Consortium (POC) [4], the NASA SWEET ontologies [168] a set of earth observation ontologies and SNOMED [5] that provides a medical ontology.

Application ontologies take an application specific view of the world and re-use and extend terms from one or more domain and domain task ontologies to realise the knowledge for a specific application. An application ontology generally can not be reused for other applications. Domain ontologies however are application independent and can be re-used for a variety of applications. This classification promotes the re-use of existing domain ontologies and provides a starting point for building new application ontologies.

Representing tasks is crucial in any application. Tasks can be specific to an application or can be generic and reused across multiple applications. Tasks can also be viewed as processes as described in section 2.3.1.7.

2.3.2.2 Ontology merging and mapping

An application may require access to multiple independently created ontologies. These ontologies may have been independently developed and in the worst case may not even use the same upper ontologies.

Mapping between concepts and aligning relationships from these ontologies can prove to be difficult. Much work has been carried out on ontology merging and mapping. A good survey of mapping techniques is given by Kalfoglou and Schorlemmer [104] who review over thirty works in this area while Choi et al [48] review and provide a comparison of several ontology mapping tools.

2.3.2.3 Ontology acquisition

A number of methodologies exist for developing ontologies. A description and comparison of different methodologies, such as the KACKTUS and METHONTOLOGY approaches is given by Gomez-Perez et. al. [80]. Ontology learning involves the automatic acquisition of ontology content. Shamsfard and Barforoush [178] compare seven ontology learning systems mostly generated from text corpora. A more recent review by Zhou [215] provides a framework for ontology learning and describes some open challenges in the field. Most approaches usually require background knowledge, such as text corpora, but there are attempts to locate and use fragments of existing online ontologies to construct new ones such as the system proposed by Alani [19].

2.3.2.4 Tools and development libraries

There are many tools for developing ontologies [80]. Protege [9] is an open source Java based ontology editor maintained by the Stanford Center for Biomedical Informatics Research. It is well established, with a large user community with 68% of Semantic Web users using Protege as their ontology editor [43]. Protege has many useful plugins for visualising, merging and acquiring ontologies. SWOOP [105] is also a Java based open source ontology editor. It originated from the Mindswap group at the University of Maryland. Protege is ideal for research and experimentation as it provides many different plugins that provide a much wider range of functionality than SWOOP. From our experience we found the Protege user interface better suited for users who are new to ontologies. SWOOP is more suitable for experienced users as it is lightweight, is more responsive and more robust when working with multiple ontologies. Commercial tools that target industry users are also available, e.g. Topbraid Composer [13]. Traditional user interfaces are limited for visualising the contents of OWL knowledge bases. The Ontoviz and Graphviz plugins for Protege provide support for visualising OWL ontologies. A description of ontology visualisation approaches is provided in [76]. Many APIs have appeared which allow developers to access and manipulate OWL knowledge bases programmatically. These include the Protege API [9], the JENA API [3] and the WonderWeb API [14]. All provide some form of support for reasoning (see sections 2.3.1.4 and 2.3.1.6). An SQL type query language, SPARQL can be used for performing queries on RDF data [164]. Most editors and OWL APIs incorporate support for SPARQL.

2.3.3 Ontology based systems

Ontologies can be used at development time to guide the conceptual modeling and specification of the system or at runtime where it forms an operational component of the system [86]. At development time, existing application ontologies could be reused and adapted to develop the domain model as well as to model the tasks required for a particular application [15]. Alternatively, if no appropriate application ontology exists, then domain and upper ontologies can be reused and extended to create the appropriate application ontology. In both scenarios the ontologies assist the developer to model the concepts and tasks required for an application.

At runtime ontologies can be used in a number of ways in an information system. In an ontology aware system, a part of the system is aware of and poses queries to a knowledge base to achieve a degree of automated reasoning. A more ambitious vision is that of an Ontology Driven Information System (ODIS) [86, 193] where ontologies play a more integral role in the system. The vision proposed by David Gunning [174] for ontologies follows the trend of programmers working at increasing levels of abstraction over time:

> ”Years ago, people had to write software in machine language. Now they are at least writing much higher-level statements in Java and other programs like it. With ontologies, we would like to get to the point where programmers are selecting the concepts that they want to think about, and let that drive the software ...”

In an ODIS, ontologies are not only used to model concepts and tasks from the real world, but also to model the system entities, such as the processes, services and agents. In this way the ontologies form a runtime component, which specifies the model and controls the operation of system entities. Developers can reconfigure aspects of the system at runtime by manipulating the appropriate ontologies. Some attempts were made towards developing an ODIS [64, 116]. These approaches revolve around generating Java classes from ontology classes. Other approaches such as Ontology Driven Architectures investigate the benefits of using ontologies to manage middleware [147, 163]. This is illustrated using two case studies for managing application server components as well as for the management of web services [163].

2.3.4 Agents and the Semantic Web

Software agent technology has been around for more than a decade [75, 206] and has been successfully used in a variety of applications [102, 157, 160, 205]. However, as indicated by leading agent researchers [90, 126], the promise of widespread deployment of software agents over the Internet [102] has remained elusive. One of the bottlenecks in agent based systems is sharing and distributing knowledge [90]. Agent researchers have largely neglected the ontological infrastructure [146]. This is the focus of the Semantic Web community [44, 52, 91] which has produced standard languages such as OWL, as well as techniques and tools for developing, managing and aligning ontologies. OWL ontologies are increasingly being incorporated within agent based systems [119, 184, 197, 216].

Semantic web services form an integral part of the Semantic Web [92]. Semantic Web Services take a bottom up approach by adding semantics to web service descriptions to aid service discovery, reuse and composition. Agent based systems have a more ambitious, top down vision of creating autonomous and potentially intelligent software components that can react to system dynamism, knowledge dynamism and data dynamism that is inherently a part of the Web. There are still many challenges to engineer, select, co-ordinate and integrate individual web services to provide higher level services. As many of these issues have previously been investigated within the agent community, many researchers [38, 99, 100, 126, 181, 197] feel that agents have an important role to play in the Semantic Web.

Agents can be incorporated into a Semantic Web Services architecture. For example, clients and services can act as software agents with goals [38]:

- All agents can access and interpret Web-published ontologies and can communicate using messages whose content is represented, or can be interpreted, in terms of published ontologies
- Service providers publish semantic descriptions of their service capabilities and interaction protocols, which prospective service consumers can interpret when selecting appropriate services and when formulating their interactions with those services

- Requestor agents wishing to delegate internal objectives to external agents can reformulate those objectives as well-formed requests to service providers by using those providers' semantically described service interfaces as guides

2.4 AGENTS AND ONTOLOGIES ON THE SENSOR WEB

In this section a number of existing agent and ontology based Sensor Web systems are described.

2.4.1 Agent based approaches

Agent technology has been used widely for specific aspects of environmental information systems, such as data management, decision support or simulation [21, 176]. Below we describe four distributed agent based systems that provide architectures for managing and analysing data from multiple sensor data sources.

IrisNet The Internet-scale Resource-Intensive Sensor Network Services (IrisNet) [41, 77] is a distributed software architecture that provides high level sensor services to users. It proposes a two tier architecture comprising of Sensing Agents (SA) that pre-process and filter raw data from one or more sensor feeds, and Organising Agents (OA) that collect and store data from multiple SAs to provide specific application services, e.g. to identify free parking spots in a parking space application. A key feature is that SAs are dynamically programmable which allows an SA to be dynamically reconfigured to filter its data for different applications. An OA can send different pre-processing steps, called senslets, to an SA which executes these steps on its raw data stream and returns the results. The architecture addresses issues related to storing and processing large scale distributed data. IrisNet also supports data reduction by extracting higher level features (OA) from raw data from different sensors (SA).

Abacus Abacus [22] is a multi-agent system that has been used for managing radar data and providing decision support base on this data. Abacus has a three layered architecture. The *Contribution Layer* contains agents that wrap physical sensors, and provide some pre-processing such as error correction and noise removal. The *Management and Processing Layer* contain a community of agents with similar processing functionality. Agents are responsible for processing data for a given spatial location or spatial sector. Different constraints or rules can be specified for different sectors and local alarms are triggered according to these. A coordinating agent or abacus agent assembles results from the processing agents to provide a joint data view, and processes local alarms from the processing layer to trigger global alarms. The *Distribution Layer*, provides user interfaces for data visualisation, and disseminates warnings via email or the web. User interfaces are also available for entering decision rules to generate alarms for the processing and the abacus agents. This architectural model was also used in an application to monitor and evaluate air quality data [23].

AIGA The Agent Based Imagery and GeoSpatial Processing Architecture (AIGA) [143] is a multi-agent infrastructure that supports image processing, geospatial data processing as well as text mining. The architecture focuses on performance and scalability issues related to using mobile agents for image processing, geospatial processing and text mining applications, especially the agent mobility and scalable discovery aspects. The architecture differentiates between data agents, processing agents and co-ordination agents that provide directory services. Agents communicate with each other through a shared communication space. The system provides a user interface for end-users to compose workflows of agents to provide different application functionality. These workflows are represented in an RDF based language and can be stored in the system and re-used at a later time. The AIGA architecture was used to implement a prototype application for cloud detection.

Biswas and Phoha approach The agent based approach by Biswas et al. [28, 29] also proposes a three layered logical architecture for developing sensor applications. The *data layer* consists of sensor nodes that provide data from physical sensors. The *application layer* fuses and interprets data from multiple sensor nodes. A *service layer* acts as an integration layer between the data layer and the application layer. It provides middleware services that includes discovery, mobility and data management. Agents are used in the application and the service layer and the CCL language is used to command and query sensors. Agents in the service layer integrate data from one or more sensors from the data layer and provide data to application agents in the application layer.

2.4.1.1 Analysis of agent based approaches

All agent based approaches described above propose some form of layered architecture [39] that provide abstractions to separate sensor agents from data analysis and filtering agents and to ease the modeling of agent based applications. In Abacus, different agents in the processing layer are responsible for different spatial sectors and detect and report alert conditions to a higher layer for distribution to users; IrisNet uses organising agents to collect and analyse data from sensor agents to answer specific classes of queries; and the Biswas and Phoha approach uses agents in the service layer to integrate data from sensors in the data layer to provide data to application agents in the application layer. These approaches address aspects of distributed processing such as mobility, data filtering and scalability. However, none provide a comprehensive ontological infrastructure that guides the creation of geospatial application ontologies, and allow for representing agents and their interaction protocols. This limits agent discovery, agent interoperability, agent composition and the potential for dynamic re-use and integration of data. While these approaches are promising for a single or a group of organisations building distributed agent based applications, more support is required for creating and managing the ontologies that will facilitate semantic interoperability on the Sensor Web.

2.4.2 Ontology based Sensor Web approaches

A worldwide Sensor Web will produce myriads of data encoded in different data formats and described using different terminologies. Real time and highly distributed aggregation, fusion and summarisation of the data is necessary to handle the data volumes and the distributed nature of the data generation [25]. Ontologies are being widely investigated within the geospatial community to standardise, dynamically integrate and query complex earth observation data [16]. The need for semantics in the geospatial domain is highlighted by Egenhofer [60] in his vision of the Semantic Geospatial Web. There are ongoing efforts to markup, query and visualise geospatial data [173] as well as the services that provide this data [113, 127].

Ontologies can provide many potential benefits to the Sensor Web:

- Facilitate the discovery of data for different purposes and in different contexts. Concepts can have different names and can be referred to at different levels of granularity. Semantics can be used to describe the real world entity and property being measured and not just the current usage and context. In this way data can be reused for different applications and even for future applications that may not seem intuitive in the current context. Semantic matching produces superior results to syntactic or keyword matching [60, 127].

- Facilitate integration of data. Data integration is often carried out manually and can be tedious and time consuming. Semantics can facilitate data integration by highlighting commonalities between data or automating aspects of data integration, e.g. the automatic conversion between related units of measure, identifying similar observations, e.g. brightness temperature and blackbody temperature measurements. However, data integration is a tricky process with the quality of the resultant merged data being difficult to determine.

- Facilitate the analysis of data. Ontologies can be used to represent data processing services and to assemble these processes into executable workflows [113, 133, 201]. Raw data can be automatically processed, transformed or filtered by plugging the data into predefined processing units.

- Model and simulate earth system processes, such as watershed run-off, ocean heat transport or atmospheric circulation [170]

Agarwal [16] summarises recent advances in ontology research in the geospatial community. Despite these advances there are still many outstanding challenges. The added temporal and spatial dimension associated with geospatial data requires additional representation support for modeling and formalising this domain [16, 25]. Sensor data also has an inherent level of uncertainty [25]. The Web Ontology Language (OWL) lacks support for representing time, space and uncertainty [177] as well as for representing system entities such as agents, services and processes. Furthermore, given the different theoretical positions for modeling geospatial entities that exist within the geospatial community, a single ontology that incorporates all standpoints may be difficult [16].

One intuitive approach to model geospatial entities is to follow the human cognition system. Humans store knowledge in three separate cognitive systems within the mind [132]. The *what* system of knowledge operates by recognition, comparing evidence with a gradually accumulating store of known objects. The *where* system operates primarily by direct perception of scenes within the environment, picking up invariants from the rich flow of sensory information. The *when* system operates through the detection of change over time in both stored object and place knowledge, as well as sensory information. In addition, categorisation plays a key role in geographical cognition. Taxonomic and partonomic hierarchies facilitate the recognition of objects from sensory perception of the environment by providing schema for generic types of objects. Thus, an object is classified when it is encountered. This allows humans to reason at an abstract level, without worrying about the detail. This schema is constantly revised to represent the latest view of the world. A recent effort in the Semantic Web community has proposed ontologies that provide separate representations for space, time and theme [161] and also show how Semantic Web applications can be built using these ontologies [88]. However, the approach does not address the representation of uncertainty nor representing and composing system entities.

The ontologies within the Semantic Web for Earth and Environmental Terminology (SWEET) [168] provides an upper-level ontology for Earth system science. The SWEET ontologies are represented in OWL and include several thousand terms, spanning a broad extent of Earth system science and related concepts. SWEET also provides for the representation of time, space and theme. OntoSensor is an OWL based ontology for representing sensors[172]. It includes definitions of concepts and properties from SensorML and ISO 19115 [2], and references the IEEE SUMO upper ontology.

The SOUPA [45] ontology was designed for pervasive context-aware systems. It is represented in OWL and provides support for representing geospatial data. It uses DAML-Time [94] for representing temporal concepts and OpenCyc Spatial Ontologies [169] and Regional Connection Calculus (RCC) [165] for representing space. The SOUPA ontologies are used within the COBRA architecture [46].

Fonseca et al [64] describes how ontologies can be used as an integral part of a geographical information system in terms of an ontology driven geographical information system (ODGIS). The system explicitly caters for image classification. In ODGIS, image classification is carried out iteratively at different levels, starting with general associations and proceeding up to a final classification that is more precise and detailed. Ontologies are not only used for domain modeling but in the actual development of the system as well, where concepts from the ontology are translated into Java classes that are used by developers to construct software components.

Lui and Zhao [125, 201] propose a semantic services framework to support a sensor infrastructure. They aim to create a system that is usable by ordinary users and to support multiple applications simultaneously. They note that in previous systems there is little reuse of the application software components, and that the system architectures are often closed with event semantics being hard-wired into an application. They propose a hierarchical architecture for sensor infrastructures, consisting of sensors, field servers, and gateway servers. At the bottom level are sensors and sensor systems, which forward data to the field servers in the next level. Field servers gather data from various sensor nodes and converts

[2]http://www.iso.org

and aggregates sensor data into a format that is open and directly usable. It is also capable of processing sensor data in response to specific user tasks. Users interact with a sensor infrastructure through gateway servers through which user requests and sensor information flow. This approach proposes a way in which event streams can be constructed from real-time sensor data. These streams can be connected to processing units that filter and process the data to produce new event streams. The representation of information uncertainty and quality of service are identified as key issues that are still to be addressed.

2.5 SUMMARY

While the ontology based systems have shown some success in modeling geospatial data, none provide a comprehensive framework for representing all aspects of geospatial data (space, time, theme and uncertainty) and the system entities (agents, services and processes) that will serve and process this data. The agent based approaches provides better abstractions for modeling and managing system entities. However, current multi-agents system infrastructures are inflexible and static and do not address ongoing application deployment and reconfiguration. Furthermore, most agent based architectures assume that ontologies already exist, and provide little support for creating and managing ontologies.

Chapter 3

DESIGN OF AN INTERNET WIDE MULTI-AGENT SYSTEM INFRASTRUCTURE

As highlighted in the previous chapter (see section 2.2.3), most existing Multi-Agent Systems (MAS) have a static or inflexible infrastructure. Furthermore, existing MAS platforms provide limited support for application deployment and reconfiguration. In this chapter the requirements for a flexible Internet Wide Multi-Agent System (IWMAS) infrastructure are first outlined. A new IWMAS infrastructure and a corresponding middleware platform, the Multi-Agent System Infrastructure for the Internet (MASII), is then described [1].

MASII is based on an infrastructure model that delineates between static core infrastructure services which are application independent, and an application infrastructure that customises and extends these services for specific applications. The MASII platform provides an agent transport layer for agent communication, an agent execution model and a flexible adapter based framework that facilitates the development and deployment of applications and services. MASII is used in the Sensor Web Agent Platform (SWAP) (described in the next chapter) for agent development, execution and deployment. MASII allows SWAP developers to develop and deploy agent based services and applications without requiring detailed knowledge of the lower level agent infrastructure services.

3.1 REQUIREMENTS FOR A SINGLE GLOBAL MULTI-AGENT INFRASTRUCTURE

An IWMAS application typically combines the functionality of the services offered by one or more Service Agents, and presents this functionality to end users via a set of user interfaces hosted at a User

[1] Parts of this chapter were previously published in [134, 135, 136].

Agent. Applications may require a single service offered by a single Service Agent or multiple services offered by multiple Service Agents and involving complex agent interactions. Four different application components can be identified in an IWMAS application:

- shared ontologies that specify the data model and tasks required for the application and that form the basis for meaningful inter agent communication.
- external behaviours, i.e. the interaction protocols and messages that allow for consistent inter-agent communication as well as any specialised service registration and querying mechanisms.
- specialised application user interfaces that are hosted at a User Agent and that provides IWMAS users access to the application.
- internal policies and procedures that govern the internal operation of each agent that participates in the application. This includes any additional data models, data stores and code (application logic) required for the internal processing of an agent.

An IWMAS should provide the flexibility for using different implementation techniques and technologies for different applications. New applications may reuse existing application components while others may require different application components to that which currently exist. In addition to traditional MAS infrastructure requirements as identified in [74, 187, 203] (see section 2.2.2) the following requirements are imposed for an IWMAS:

- *Flexibility*: The system must concurrently support a wide range of distributed applications in disparate application domains, e.g. the medical domain, the earth observation domain and electronic commerce applications. The system must provide support for developing basic single agent applications, e.g. a weather information application, but also allow for the development of more complex applications that require multiple agents that offer disparate services, e.g. informal settlement detection from satellite imagery. Application developers should be allowed to use different application components to satisfy different requirements imposed by different application classes. Where possible, components of a specific application class should be reusable in other application classes.

- *Seamless deployment and integration of new applications*: New applications and upgrades to existing applications must be easily deployed and made available at runtime to users, without impacting on other applications.

- *Low cost of entry to participate in the system*: An end user or developer must be able to easily participate in the IWMAS without requiring in-depth knowledge of agents.

Different application classes will have common requirements, i.e. functionality that is required across multiple application classes, and custom requirements that are specific to a single application class. It is then necessary to differentiate between infrastructure responsibilities that provide for the common requirements and application responsibilities that provide for the custom requirements of an application class. An infrastructure that makes this distinction and attempts to address IWMAS requirements is described next.

3.2 THE MULTI-AGENT INFRASTRUCTURE FOR THE INTERNET

The Multi-Agent Infrastructure for the Internet (MASII) aims to provide an agent programming and execution environment for developing, deploying and maintaining agent based applications over the Internet.

3.2.1 An abstract architecture for a IWMAS

The infrastructure design extends Sycara's abstract MAS architecture [187]. Sycara's abstraction hierarchy splits the infrastructure services into two parts. The infrastructure part provides infrastructural services and the individual agent part allows an agent to interact with the infrastructure services. Sycara's abstraction hierarchy is expanded to introduce a third part, the application infrastructure part shaded in grey in figure 3.1. The application infrastructure allows each application class in the MAS to use core infrastructure functionality but also allows this functionality to be extended and customised for different application classes. This allows for different applications classes to have different application infrastructures.

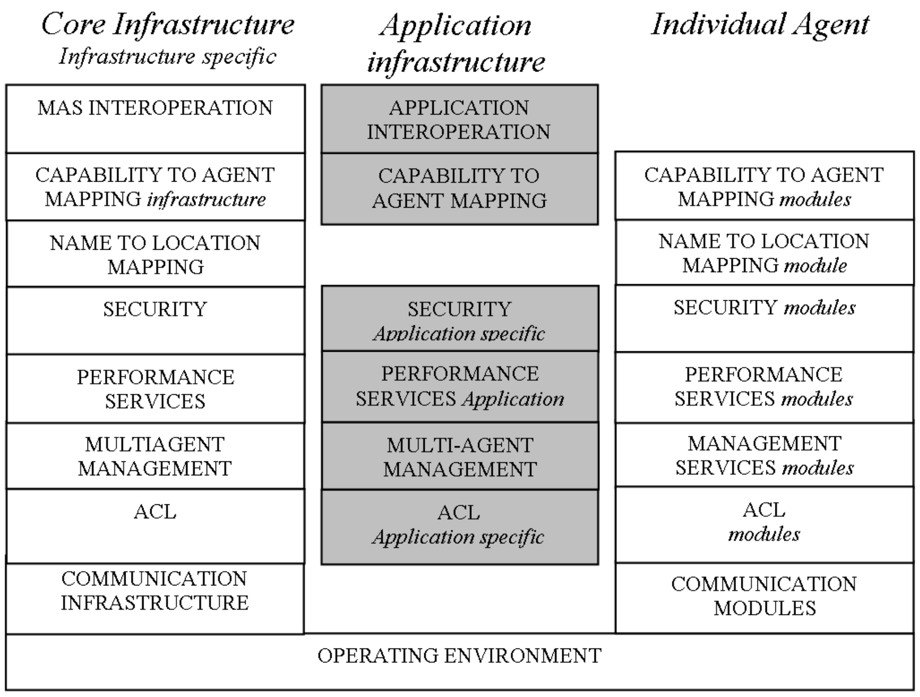

Core Infrastructure
Infrastructure specific
Application infrastructure
Individual Agent
MAS INTEROPERATION
CAPABILITY TO AGENT MAPPING infrastructure
NAME TO LOCATION MAPPING
SECURITY
PERFORMANCE SERVICES
MULTIAGENT MANAGEMENT
ACL
COMMUNICATION INFRASTRUCTURE
APPLICATION INTEROPERATION
CAPABILITY TO AGENT MAPPING
SECURITY Application specific
PERFORMANCE SERVICES Application
MULTI-AGENT MANAGEMENT
ACL Application specific
CAPABILITY TO AGENT MAPPING modules
NAME TO LOCATION MAPPING module
SECURITY modules
PERFORMANCE SERVICES modules
MANAGEMENT SERVICES modules
ACL modules
COMMUNICATION MODULES
OPERATING ENVIRONMENT

Figure 3.1: MAS infrastructure, application infrastructure and an individual agent that enables an agent to be a part of the MAS

Capability to agent mapping The core infrastructure provides a basic service registration and discovery mechanism. However, in practise, customised service description languages may be required for different application classes. Developers have the flexibility to select from existing registration and discovery mechanisms and may even deploy new ones if the existing ones are not satisfactory. Ideally, as the various discovery mechanisms evolve and mature in the system, a single discovery mechanism that is used by the majority of application classes will emerge.

Security The core infrastructure provides security services such as certificate authorities and secure messaging. It is responsible for preserving the integrity of the infrastructure. Additional higher-level security features such as access control and application level authentication can be provided in the application infrastructure.

Performance Services The core infrastructure provides performance services such as network response time and availability. The quality of service and performance of an agent, with regard to the service it provides, varies between application classes. Specialised quality of service measurements such as credibility and reputation can be provided in the application infrastructure.

Multi-agent management services The core infrastructure provides a basic set of management services. Additional management services such as additional logging services, agent configuration, launching and installation options can be provided in the application infrastructure.

Agent Communication Language (ACL) The core infrastructure provides an ACL for agents to communicate and function within the infrastructure. It specifies an ontology language and an infrastructure ontology. Concepts such as agent, service, message and interaction protocol are defined in the infrastructure ontology. Custom ontologies and interaction protocols can be provided in the application infrastructure. To facilitate interoperability between application classes, developers will be encouraged to reuse existing ontologies and protocols for new application classes.

MAS and application interoperation The application interoperation layer facilitates interoperation between different application classes. Applications within the same application class use a common application infrastructure. Although this enables interoperability between applications from the same class, this does not guarantee interoperation between application classes. This is especially apparent when there are major disparities between ontologies or their associated conceptual models. Interoperation then requires the creation of mappings between these ontologies (see section 2.3.2.2). The level of interoperability that can be achieved may vary. This will depend on the nature and the level of the disparities and the extent to which mappings can be used to overcome these. The application interoperation layer maintains an application ontology that defines concepts that can be used across application classes. As more application classes are deployed into the system, concepts which are used across multiple applications will emerge. These concepts can be incorporated into the application ontology for reuse in future application classes. The application ontology will evolve through the lifespan of the MAS and will facilitate interoperation between application classes. The MAS interoperation layer is retained in the core infrastructure to allow developers that use other MAS platforms to interoperate with this platform.

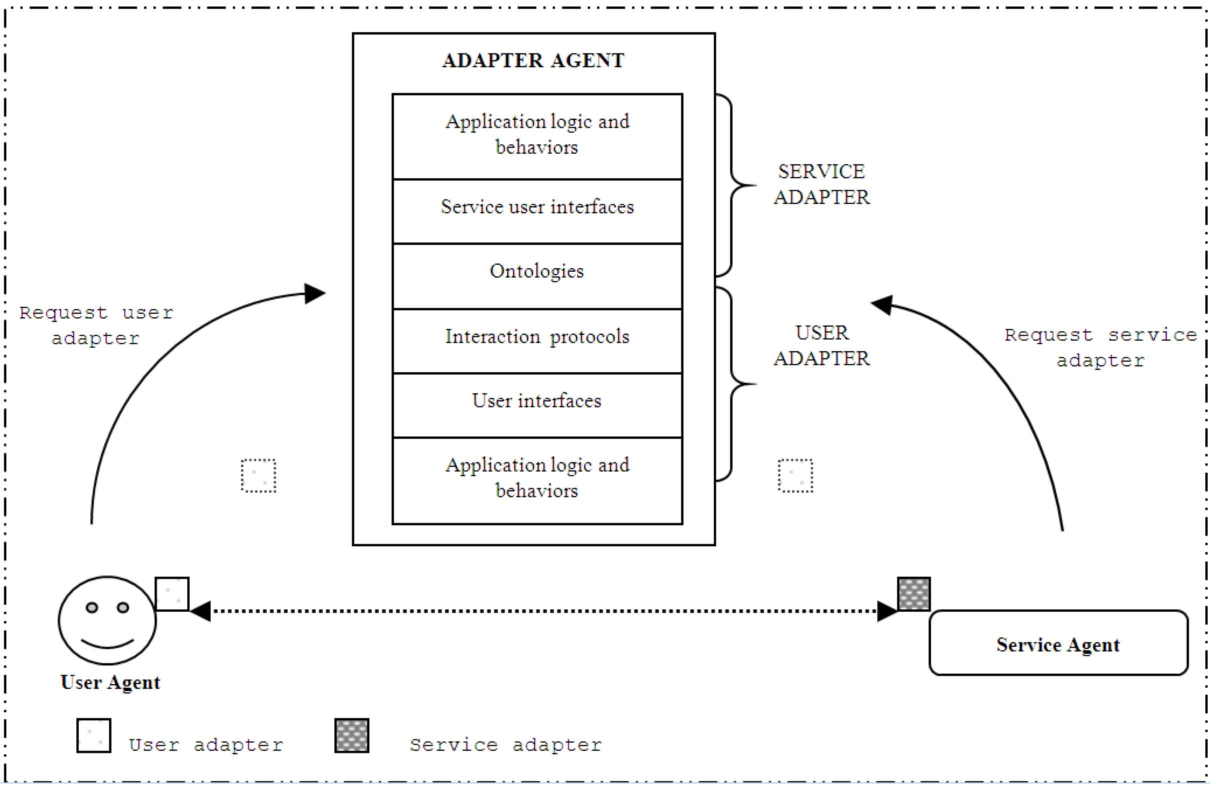


Figure 3.2: The MASII adapter architecture

3.3 MASII DESIGN AND OPERATION

A MASII application has four components (see section 3.1). These are: ontologies; agent interaction protocols with the corresponding message handlers; user interfaces for end users to access the application and optionally to manage services; and internal code for each agent's internal processing.

MASII agents use an adapter architecture. A basic agent incorporates a transport layer which allows it to discover and communicate with other agents. An agent extends its functionality by downloading and installing adapters. An agent service requires a user and a service adapter. The service adapter contains the application components required by a Service Agent to offer the service, and the user adapter allows another agent to access and use the service. Adapters are deployed on an Adapter Agent and can be retrieved and installed by any agent. Specialised user adapters containing end user interfaces are required by User Agents. As shown in figure 3.2 agents extend their functionality by downloading, installing and maintaining a local repository of adapters, to offer either services (Service Agents) or end user applications (User Agents).

Figure 3.3 shows the different types of agents in the MASII infrastructure. The infrastructure consists

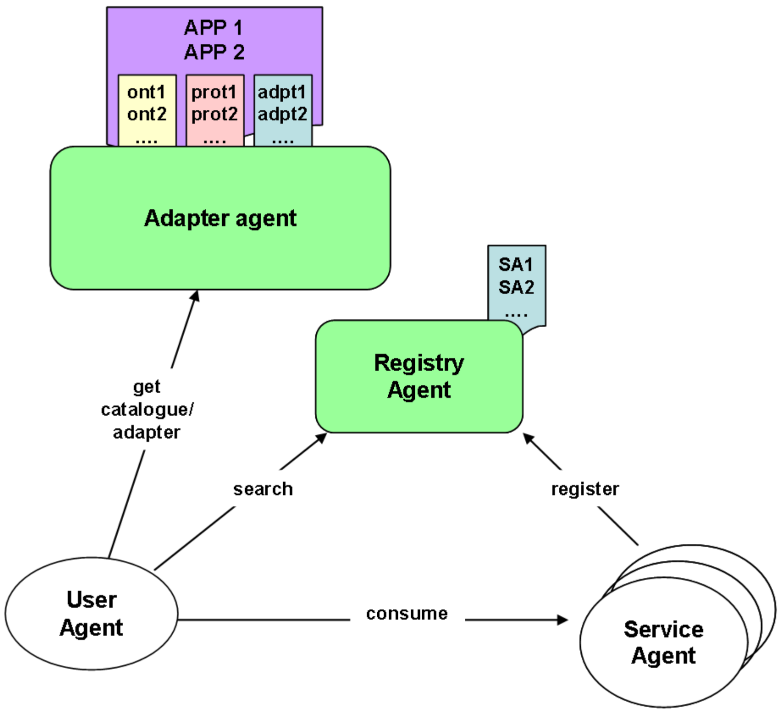


Figure 3.3: The MASII system architecture

of Registry Agents, Adapter Agents, User Agents and Service Agents.

3.3.1 Registry Agent (RA)

The RA provides middle agent (see section 2.2.1.1) functionality in the MAS. Service agents register their capabilities with the RA so that User Agents can discover and use their service offerings.

3.3.2 Adapter agent (AA)

The AA maintains an application catalogue that contains an entry for each application in the MAS. Each entry specifies the name and description of the application and the registry agent(s) to use to discover the services required in the application. The application description may include information about its functionality and the quality of service (QoS) properties. A User Agent periodically downloads the application catalogue from the AA and is thus aware of new application offerings in the MAS. The AA is responsible for storing the adapters, the protocols and the ontologies for each application. Agents download the required application components in order to use or provide services for an application.

When an application is ready for deployment, a user adapter together with the corresponding protocols and ontologies are uploaded to the AA and the application catalogue is updated. When a situation arises where the User Agent is required to offer an application for the first time, it makes a request to the AA for the required application components. The User Agent matches the requirements with existing application components in its local repository and downloads and installs the missing components. In this way the User Agent can dynamically acquire new capabilities and adapt to changes in the MAS. The service adapters may also be provided for downloading on the AA. Service Agents may then download, install and even replicate service offerings.

3.3.3 Platform implementation

A prototype platform was implemented in Java. The platform provides a generic agent that is able to send and receive messages to other agents. The current mechanism uses CORBA [185] (IIOP) for asynchronous message passing. The transport layer and message queue mechanism are hidden from the developer and no knowledge of CORBA is required. The transport layer is exchangeable and HTTP and email (POP, SMTP) protocols can be used as alternative transport protocols.

At an implementation level, the platform distinguishes between agents, services and protocols. An agent executes within an agent environment. The environment allows it to pass messages to other agents. It contains a message queue to receive and handle messages from other agents. The environment also allows for the dynamic loading of services and protocols. For each service offered by a Service Agent a service initialiser is executed to perform any initialisation tasks e.g. connecting to a knowledge base or initialising required data structures. A graphical user interface (GUI) is also initialised for each service for configuring and monitoring the service. For a User Agent to access a service, the User Agent executes a users side initialiser for the service and loads the user interfaces for the service. The implementation of a new service involves implementing a service initialiser and service GUI for a Service Agent to host the service and an initialiser and GUI for a User Agent to use the service. The service initialiser, user interfaces and data stores are packaged together into service and user adapters for Service Agents and User Agents respectively as illustrated in figure 3.2.

A protocol defines a conversation between agents. At an implementation level a protocol involves implementing message handlers at the Service Agent and the User Agent. Message handlers at the User

Agent handle incoming messages from the Service Agent. Message handlers at the Service Agent handle incoming messages from the User Agent. A protocol configuration file specifies the message handler (class) to use to process each incoming message. A User Agent can be extended to use a new service by installing the user side adapter and protocol handlers. Similarly a Service Agent can be extended to offer a new service by installing the service side adapter and protocol handlers.

The protocols, ontologies and adapters for each application are maintained on the Adapter Agent within Java archive (JAR) files. Agents are able to retrieve the required components for an application and install and activate these at runtime.

3.3.4 Application development

Three main activities are required for developing MASII applications.

3.3.4.1 Agent design

This involves identifying the roles, responsibilities and activities of each agent in the system. For each activity the interaction protocols between agents to perform the activity, are identified. This approach is based on the GAIA [213] methodology for designing agent based applications.

3.3.4.2 Ontology development

Once the agents, services and interactions have been identified, these are captured within ontologies. The ontologies describe the agents, the services they host and the content of the messages that the agents exchange during their interactions.

3.3.4.3 Adapter implementation

For each agent service, the developer implements a service adapter which contains the necessary message handlers and application logic for the service. A corresponding user adapter is also implemented which allows another agent to invoke the service. Typically this is deployed on a User Agent with appropriate user interfaces for end users to interact with the service. The user adapter together with the protocols and ontologies are then uploaded to the Adapter Agent where it can be downloaded and installed by User Agents.

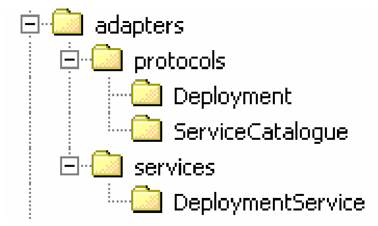


Figure 3.4: User agent adapter data store

3.4 APPLICATION DEPLOYMENT

In order for a end user to access applications, the user installs and executes a MASII User Agent. At this stage the User Agent has no application adapters and the end user has no knowledge of the applications that exist in the system. The User Agent can initially only access infrastructure services, i.e. the *Deployment* service, the *Deployment* protocol and the *Service Catalogue* protocol. The *Deployment* adapter and protocol enables the User Agent to download and activate selected user adapters from the Adapter Agent at runtime. All supported services and protocols are stored locally in an adapter folder within the User Agent's working directory. Figure 3.4 shows the adapter folder of a basic User Agent. Additional adapters can be retrieved from the Adapter Agent and installed locally. Downloaded adapters are persistent and are activated each time the User Agent starts.

3.5 DISCUSSION

Most MAS infrastructures are static. All agents use the same ontology, ACL and matchmaking mechanism. In such systems the protocols, ontologies and infrastructure services are burdened a priori with a requirement to cater for all current and future applications. These mechanisms are usually based on the requirements and knowledge of current applications or follow a specific agent methodology. This restricts the innovation of new infrastructure mechanisms and new types of applications. An IWMAS requires a more flexible infrastructure that supports the wide spread development of different classes of applications across multiple domains. Developers should be free to select from existing infrastructure services, extend these or deploy new ones if current ones do not satisfy the requirements for their applications. This flexibility will encourage IWMAS usage and allow parts of the infrastructure to evolve in line with the current requirements and usage.

In designing this IWMAS the aim was to build a flexible MAS middleware platform that provides explicit support for runtime application deployment. MASII is based on an infrastructure model that delineates between static core infrastructure services which are application independent, and an application infrastructure that customises and extends these services for specific applications. The platform provides an agent transport layer for agent communication, an agent execution model and a flexible adapter based framework that facilitates the development and deployment of applications and services. MASII provides the underlying agent middleware services for developing agent based services and applications. MASII allows application developers to develop and deploy agent based services and applications without requiring detailed knowledge of the lower level agent infrastructure services.

The adapter architecture allows for the runtime deployment of new applications and dynamic upgrading of application components of existing applications. End users are immediately aware of and can access new applications. User Agents extend their capability by retrieving and installing application adapters from Adapter Agents. These adapters contain the application components, i.e. ontologies, protocols, application logic and user interfaces, required for an application. Service adapters may also be deployed in a similar manner. Different Service Agents can offer the same service by retrieving and installing the service adapter. Service replication increases fault tolerance and allows for load balancing. Upgrades to existing user and service adapters can be dynamically deployed through the system. This eases post deployment maintenance of applications.

An important aspect of maintenance not currently supported is the versioning of application components. In a real world system different versions of application components may exist. The functionality of the Adapter Agent can be extended to store and manage different versions of application components. The application catalogue can be used to capture the dependencies between applications and application components.

This chapter describes a flexible, adapter based IWMAS infrastructure and middleware platform that explicitly addresses runtime application deployment. However, an application framework is required to develop specific classes of applications. The next chapter describes the Sensor Web Agent Platform, an application framework for designing and developing complex earth observation applications.

Chapter 4

DESIGN OF THE SENSOR WEB AGENT PLATFORM

This chapter describes an Ontology Driven MAS for the Sensor Web, the Sensor Web Agent Platform (SWAP). SWAP builds on the middleware services provided by MASII to provide an application infrastructure for developing complex earth observation applications. The vision and technical challenges for creating a single worldwide Sensor Web is first reiterated and an application framework that attempts to address these challenges is then described [1].

4.1 OUR VISION OF THE SENSOR WEB

Our vision of the Sensor Web is to create a worldwide computing platform that allows end users to dynamically access multiple sensor sources, and extract and use appropriate information from these sources. Three broad technical challenges are apparent:

1. Creating a publicly accessible, open distributed computing infrastructure where heterogeneous sensor data services, data processing services and complex end-user applications can be deployed, dynamically discovered and accessed.
2. Integrating data from different sensors that have different temporal resolutions, spatial resolutions, data models and data formats. A higher spatial coverage and temporal resolution is achieved by integrating data from different sensors.
3. Performing context-based information extraction. The technical skill and time required to extract appropriate information from sensor data forms a barrier to a potentially large end-user community who could benefit from this data. Depending on their needs (context), users may require different

[1]Parts of this chapter were previously published in [137, 138, 139].

aspects of the sensor data. The same data may be used for different applications. Users must not be overwhelmed by the complexity and scale of sensor data. They should be presented with just the information that they require for their task. Constructing and maintaining customized real-time monitoring applications, especially remote sensing applications, is tedious and requires personnel with advanced remote sensing and programming skills.

The Sensor Web Agent Platform (SWAP) aims to alleviate these challenges. SWAP builds on the core infrastructure services provided by MASII (see section 3.2). It provides an application framework that consists of an abstract architecture, an ontological infrastructure and an internal agent architecture for designing and developing Sensor Web applications.

4.2 THE SWAP ABSTRACT ARCHITECTURE

The SWAP abstract architecture (figure 4.1) is a layered architecture [39] that provides logical agent abstractions for designing and deploying Sensor Web applications.

It consists of three layers. Sensor Agents in the *Sensor Layer* encapsulate individual sensors, sensor systems and archived observations. They expose sensor data in a uniform way and deal with any sensor-dependant processing. Tool, Modeling and Workflow Agents in the second layer, the *Knowledge Layer*, retrieve and process data from Sensor Agents. Tool Agents provide data processing services such as feature extraction. Modeling Agents store real-world models and can provide projections and analysis of data. Workflow Agents retrieve data from Sensor Agents, passes this data through a combination of Tool and Modeling Agents, and aggregates the results. The results are stored by Workflow Agents and used by Application Agents in the third layer, the *Application Layer*. Application Agents combine higher level features provided by Workflow agents and provides different alerting and monitoring functionality to different end users. User Agents allows end users to access the functionality from multiple Application Agents. Users can register for and receive alerts from different Application Agents via their User Agent.

4.2.1 Sensor Layer

Sensor Agents represent either data access or sensor control services or a combination of both. Sensor Agents access physical or virtual sensors directly or via intermediary services such as an Open Geospatial Consortium (OGC) Sensor Observation Service or a Web Coverage Service. Sensor Agents handle

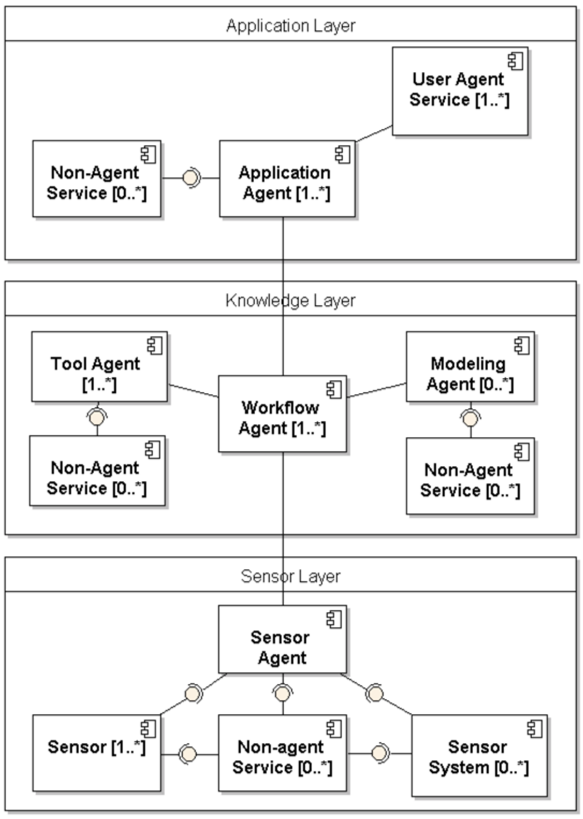


Figure 4.1: Three layered SWAP architecture

all sensor specific operations internally. This includes error correction, calibration and decoding proprietary data formats. This allows other agents to access data from different sensors without requiring knowledge of individual sensor calibration and data encoding formats. All Sensor Agents exposes sensor data or sensor operations via a uniform interface using uniform data formats. Descriptions of the service interfaces and the sensor data they provide are specified in the SWAP ontologies (described in section 4.3).

4.2.2 Knowledge Layer

The Knowledge Layer provides for the orchestration of complex processing chains that incorporate reusable modeling and processing components. Services offered by agents in the Knowledge Layer attempt to be sensor independent. Interactions with Sensor Agents are based on a common understanding of exchange formats and concepts that are specified in the SWAP ontologies. Tool Agents provide well defined and deterministic processing services, such as data transformation, data analysis or feature extraction. Tool Agents always produce a meaningful result if valid data is provided as input. Modeling Agents represent non-deterministic prediction models that may never complete or may provide multiple outcomes for a single request. Modeling Agents provide complex processing capacities that may be long running, may require additional input during processing, or may even fail to produce a usable result. Workflow agents capture and store expert knowledge in the system in the form of predefined processing chains or workflows. Workflows combine Tool and Modeling agents in specific sequences to solve specific problems. Typically, a Workflow Agent retrieves (raw) data from Sensor Agents and invokes a number of Tool and Modeling agents to run predefined processing steps on the data. After each step the processed data is passed back to the Workflow Agent and passed on to the next Tool or Modeling Agent in the workflow. After all processing steps have been completed the results are stored and can be retrieved by Application Agents in the Application Layer.

4.2.3 Application Layer

The Application Layer exposes Sensor Web applications to end users. It filters and aggregates data from the two lower layers to provide end user alerting and monitoring applications. Two types of agents are provided at the Application Layer, viz. User Agents and Application Agents. An Application Agent uses the results provided by one or more Workflow Agents to provide specific monitoring applications. A User Agent is installed and maintained by each end user. It allows a user to access the applications offered by different Application Agents. Users can select the combination of applications that they require. The User Agent can be configured to provide custom or integrated views of alerts received from these applications. The User Agent also informs users about new applications and services in the system. It can be continuously reconfigured to reflect the changing requirements of the user.

4.2.4 Incorporating OGC services

Open Geospatial Consortium (OGC) web services can be incorporated at each layer of the architecture. At the Sensor Layer, Sensor Agents can retrieve sensor data from existing Sensor Observation Services. At the Knowledge Layer, Tool Agents can redirect requests to Web Processing Services (WPS). At the Application Layer an Application Agent can use a Web Notification Service to transmit alerts using different transport protocols and data formats, e.g. SMS or pager.

4.3 OVERVIEW OF THE SWAP ONTOLOGICAL INFRASTRUCTURE

The aim of the SWAP ontological infrastructure is to provide consistent semantics to facilitate the discovery, reuse and integration of Sensor Web data and processing services. Ontologies, unlike other formal modeling languages, must be understood and managed by non computer science end-users while sufficiently formal to be interpreted by machines. Ontologies enable human users with limited technical expertise to explore and find relevant sensor data, services and applications that can aid them in their current task. They also provide sufficient technical detail for software agents to automatically interpret and process data. Additionally, ontologies should provide descriptions of agent services and interaction protocols. This is especially important to manage system dynamism on the Sensor Web where agents may change their service offerings or new agents may appear offering new services.

4.3.1 Rationale behind the SWAP ontology

Designing an ontology infrastructure that balances the requirements of both software agents as well as non technical human users is challenging. Ontologies can be split into two levels, a conceptual level and a technical level. The conceptual level ontologies supports the creation of conceptual descriptions of agent service offerings without the technical detail for invoking the services. This promotes conceptual interoperability between heterogeneous sensor resources, and dynamic extraction and integration of higher level features from sensor data. By providing good conceptual descriptions, scenarios for re-use can be inferred. However, agents still need to communicate and exchange data. This is supported at the technical level where specific data and agent message structures are specified. Technical ontologies must support data structures ranging from a single value at a specific time and space to multi-dimensional structures that hold values for specific spatial areas taken during specific time interval or at multiple time instants. Support must also be provided to represent the structure and types of messages exchanged between SWAP agents and to represent agent workflows for coordinating interactions between multiple agents. In this way heterogeneous agents can interpret and use data from other agents in its internal processing and know at which stage of its internal processing to incorporate this data.

The conceptual ontologies are based on the human cognitive system. As discussed earlier, in section 2.4.2, humans store knowledge in three separate cognitive systems [132]. The *what* system of knowledge operates by recognition, comparing evidence with a gradually accumulating store of known objects. The *where* system operates primarily by direct perception of scenes within the environment, picking up invariants from the rich flow of sensory information. The *when* system operates through the detection of change over time in both stored object and place knowledge, as well as sensory information. In addition, categorisation plays a key role in geographical cognition. Taxonomic and partonomic hierarchies facilitate the recognition of objects from sensory perception of the environment by providing schema for generic types of objects. Thus, when an object is encountered it is classified. This allows humans to reason at an abstract level, without worrying about the detail. This schema is constantly revised to represent the latest view of the world.

The SWAP upper ontologies are split between conceptual and technical level ontologies (see figure 4.2). At the conceptual level, there are four ontologies: *swap-theme* contains thematic concepts; *swap-space* contains spatial concepts; *swap-time* contains temporal concepts; and *swap-uncertainty* is an extension that allows for the representation of uncertainty. There are three ontologies at the technical level: *swap-data* contains concepts for representing different data types and data structures; *swap-task* provides concepts for representing atomic and composite processes; and *swap-agent* ontology contains descriptions for basic agent operations such as agent types, interaction protocols and agent actions. These

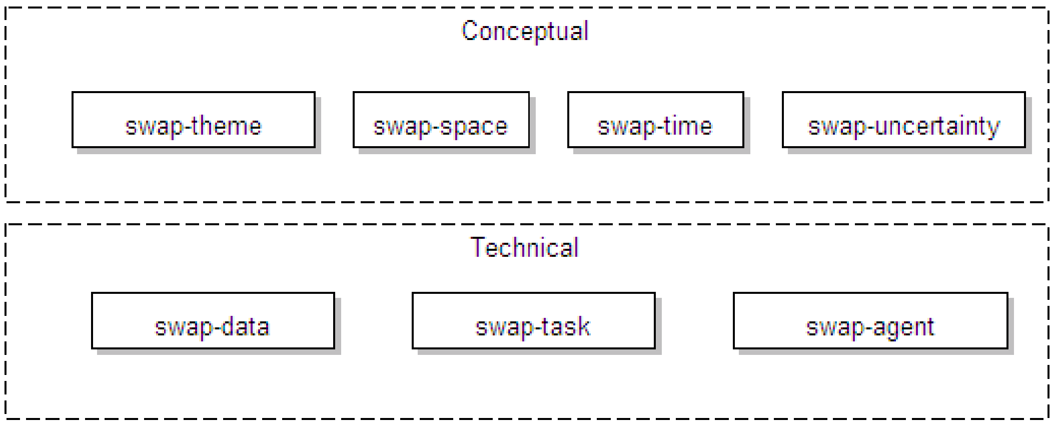


Figure 4.2: SWAP ontology levels

ontologies form the SWAP upper level ontologies. Domain ontologies for specific application domains are built by extending the *swap-theme* ontology. The *eo-domain* ontology extends the *swap-theme* ontology by adding concepts for building applications in the earth observation domain (figure 4.3). It currently links to concepts from the SWEET ontologies [167], an existing set of earth science ontologies.

Application ontologies specify concepts that are used for a specific application, e.g. wildfire detection. A *fire-detection* ontology will provide concepts required to model the wildfire application. The ontology reuses concepts from the domain and upper ontologies or extend these ontologies as required. The SWAP ontologies provide a uniform representation of sensor data. The *DataSet* concept in the *swap-data* ontology (figure 4.4) describes a sensor data set. A data set may have zero or more *thematic*, *spatial*, *temporal* and *uncertainty* properties. The structure used to hold and access the data value(s) is described by the *datatype* property.

4.3.2 Swap rules

Ontologies provide for a common interpretation of data and enable meaningful communication between agents. However, knowledge that enables agents to process current information and to respond to requests correctly must be provided. Rules can be used to encode this knowledge. Whereas ontologies provide the ”what”, i.e. knowing what is being meant by certain concepts, rules provide the ”how”, i.e. how to process and respond to information [181]. SWAP uses inference rules [32, 171] in this regard. Inference rules take the form of

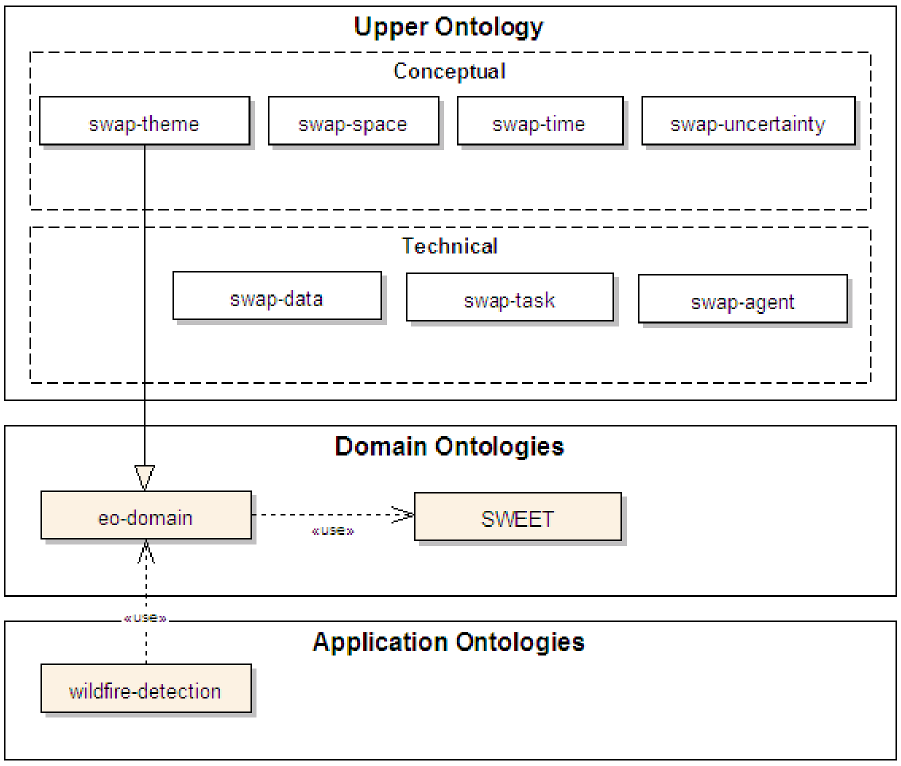


Figure 4.3: SWAP ontology structure

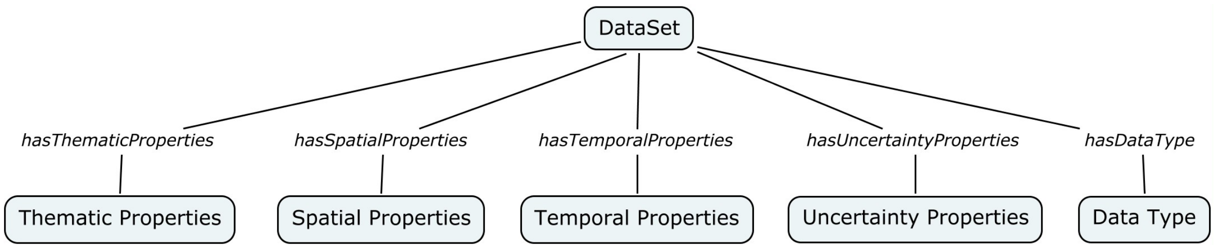


Figure 4.4: Representing a data set in SWAP

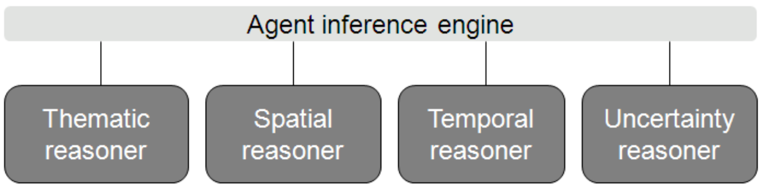


Figure 4.5: SWAP reasoning engine

if **antecedent** $then$ **consequent**

An inference rule allows for drawing a logical conclusion (consequent) if some premise (antecedent) holds. Inference rules can be run in the *forward* direction (if the stated antecedent holds, then the consequent can be inferred) or the *backward* direction (to infer a particular consequent, check that the antecedent holds). An inference engine executes these rules either as forward rules (forward chaining), as backward rules (backward chaining) or an hybrid of the two. From a given set of facts in a knowledge base, an inference engine can use a set of inference rules to draw additional logical conclusions, which result in new facts being added to the knowledge base.

SWAP uses the Java based Jena platform [2] for representing and executing rules. Jena provides a rule-based OWL reasoner that represents the OWL descriptional logics as a set of rules. These rules can be extended to specify additional domain or application specific rules. In this way, Jena bridges the gap between ontologies and rule-based systems. Jena also supports procedural attachments, i.e. rules may refer to external functions. Procedural attachments are represented as builtin Java classes in Jena. For example Jena provides the builtins $equals$, $lessThan$ and $greaterThan$ that take in two parameters (either numbers or dates) and perform the appropriate comparison, e.g. $lessThan(2, 5)$ returns $true$ and $greatherThan(6, 10)$ returns $false$.

The SWAP agent reasoning engine consists of a thematic, spatial, temporal and uncertainty reasoner as shown in figure 4.5. SWAP agents use these reasoners to interpret the contents of incoming messages and to process and respond to these messages.

4.4 THE SWAP CONCEPTUAL ONTOLOGIES

In this section the four SWAP conceptual representation and reasoning systems are described.

[2]http://jena.sourceforge.net

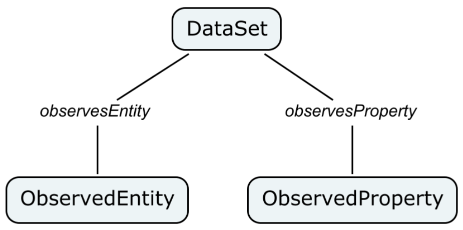


Figure 4.6: Thematic properties of a data set

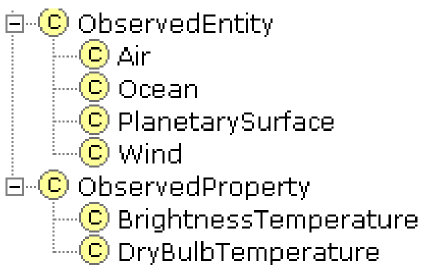


Figure 4.7: The *eo-domain* ontology

4.4.1 Thematic representation and reasoning

The *swap-theme* ontology contains two high level concepts, *observedEntity* and *observedProperty*, to represent observable entities and the observable properties of these entities respectively. There are two thematic properties of a *DataSet*, *observesEntity* describes the entity being observed, while *observesProperty* describes the property of this entity that is being measured. Thus the *observesEntity* property has *observedEntity* as its range, while the *observesProperty* has *observedProperty* as its range as shown in figure 4.6.

The *eo-domain* ontology (figure 4.7) links observable properties from the NASA SWEET [167] property ontology by making these properties a subclass of *observedProperty* such as *BrightnessTemperature*[3] and *DryBulbTemperature*[4]. Geographical entities from the SWEET earthrealm and SWEET phenomena ontologies are also linked by making these entities a subclass of *observedEntity*, e.g. *Air*, *Ocean*, *PlanetarySurface* and *Wind*.

[3]*brightness temperature* is the measure of the intensity of radiation thermally emitted by an object, given in units of temperature (*wikipedia*)

[4]*dry-bulb temperature* is the temperature of air measured by a thermometer freely exposed to the air but shielded from radiation and moisture (*wikipedia*)

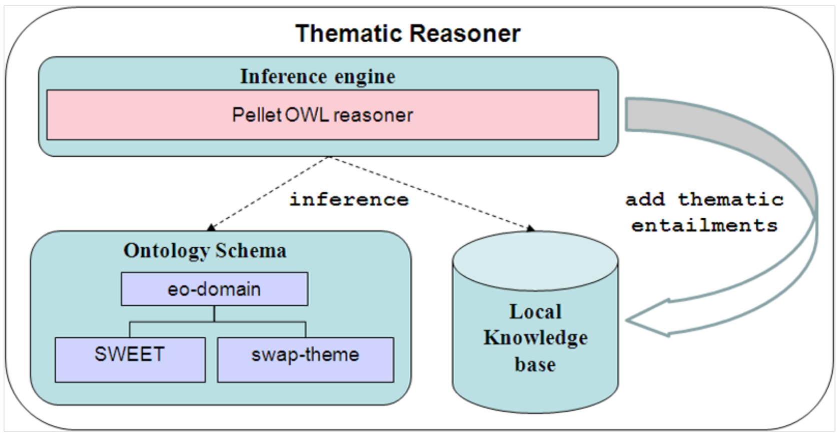


Figure 4.8: SWAP thematic reasoning

Consider a data set produced by the Meteosat Second Generation (MSG) satellite that contains measurements of brightness temperature of the earth's surface. An instance of the concept *PlanetarySurface* from the SWEET *earthrealm* ontology, *earthsurface*, is the *ObservedEntity*. An instance of the concept *BrightnessTemperature* from the SWEET *property* ontology, *msg-brightness-temperature*, is the *ObservedProperty*. A data set representing the dry bulb temperature measured by a specific weather station in South Africa can be similarly described. The *ObservedEntity* is *saweatherstation1-air* an instance of *Air*. An instance of *DryBulbTemperature*, *saws-DryBulbTemperature*, is the *ObservedProperty*.

An agent delegates thematic reasoning to the thematic reasoner (figure 4.8) to reason over the thematic properties of geographical data. An OWL reasoner preloads the ontology schema and applies the schema to instance data in the local knowledge base. The schema consists of the *eo-domain*, the *swap-theme* as well as the entire SWEET ontology, not just those concepts referenced in the *eo-domain* ontology. This allows the inference engine to detect relations with SWEET concepts not explicitly referenced in the *eo-domain* ontology, e.g. *BrightnessTemperature* and *DryBulbTemperature* are both subclasses of *Temperature*. Additional thematic entailments produced by the reasoner are added to the local knowledge base. Currently, only a standard OWL reasoner with no additional thematic rules are used for inferencing. As the default Jena OWL reasoner is not intended for inferencing over large ontologies, such as the SWEET ontologies, the Pellet OWL reasoner [5] is used to speed up the inferencing process.

[5]http://pellet.owldl.com

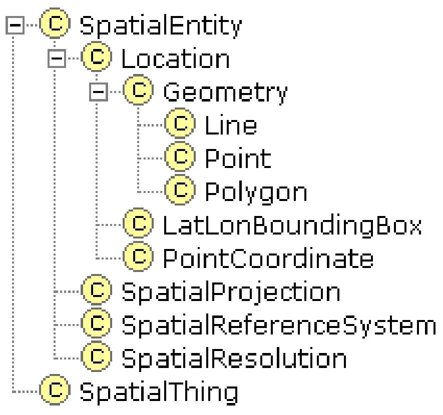


Figure 4.9: Part of the *swap-space* ontology

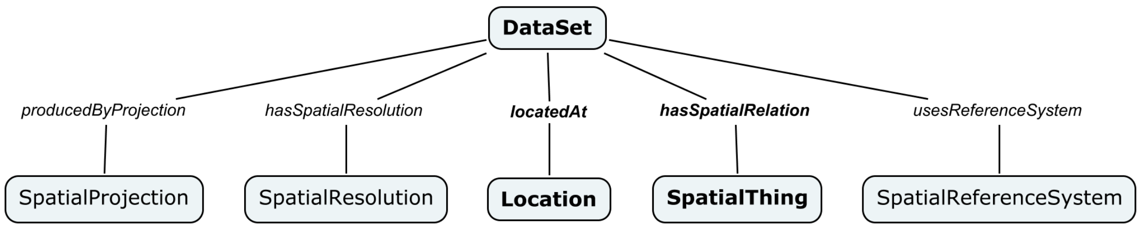


Figure 4.10: The spatial properties of a data set

4.4.2 Spatial representation and reasoning

The *swap-space* ontology provides concepts for representing the spatial aspects of data. A part of the *swap-space* ontology is shown in figure 4.10. The complete ontology is provided in Appendix A.2.1. A data set can have zero or more spatial properties (see figure 4.4). Spatial entities include spatial reference systems, spatial projection, spatial resolution and location (figure 4.9). Locations can be common descriptions such as a point coordinate or a bounding box, or well defined spatial geometries such as a point, line or polygon.

A $SpatialThing$ is defined as an entity that has a $Location$ and the spatial reasoner is used to determine how two $SpatialThings$ are spatially related. The OGC Simple SQL features spatial operators [49, 61, 150] define eight binary relations between two $SpatialThings$. The six relations *touches*, *crosses*, *overlaps*, *disjoint* and *within* and *equals* are shown between two spatial objects A and B in figure 4.11. In this notation all spatial objects have access to a boolean method for each spatial relation. Each method takes in another spatial object and checks whether the associated relation holds between the

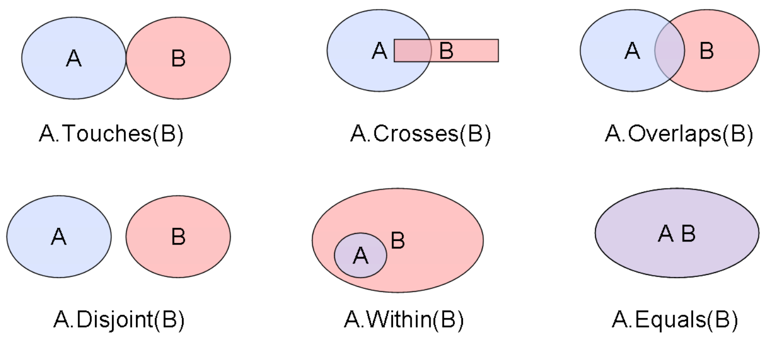


Figure 4.11: Spatial relations

two objects. For example, if A and B are spatial objects, then $A.Touches(B)$ returns *true* if A touches B. Two further relations *contains* and *intersects* are provided for convenience. These are defined as: A contains B if and only if B is within A (equation 4.1) and A intersects B if and only if A is not disjoint with B (equation 4.2).

$$A.Contains(B) \Leftrightarrow B.Within(A) \tag{4.1}$$

$$A.Intersects(B) \Leftrightarrow \,!\, A.Disjoint(B) \tag{4.2}$$

The spatial reasoner is used to determine which of the eight binary relations hold between two spatial things. Note that two spatial things can be related by more than one of these relations. The schema and the components of the spatial reasoner are shown in figure 4.12. It uses the Jena rule-based OWL reasoner. As OWL does not provide native support for spatial representation, additional rules are incorporated to determine the spatial relations between spatial things. Spatial relations are expressed as subproperties of $hasSpatialRelation$. For example, suppose that x and y are spatial things, then the reasoner might determine that x $intersects$ y where $intersects$ is a subproperty of $hasSpatialRelation$ representing the intersect relation. A set of Jena rules were formulated to infer relations between $SpatialThings$. The rules use special builtins that were created for each of the eight relations. For example, the rule used to determine whether two $SpatialThings$ intersect is:

```
(?x spc:intersects ?y) <-
    (?x rdf:type spc:SpatialThing) (?y rdf:type spc:SpatialThing)
    (?x spc:locatedAt ?xExt) (?y spc:locatedAt ?yExt)
    spatiallyIntersects(?xExt,?yExt).
```

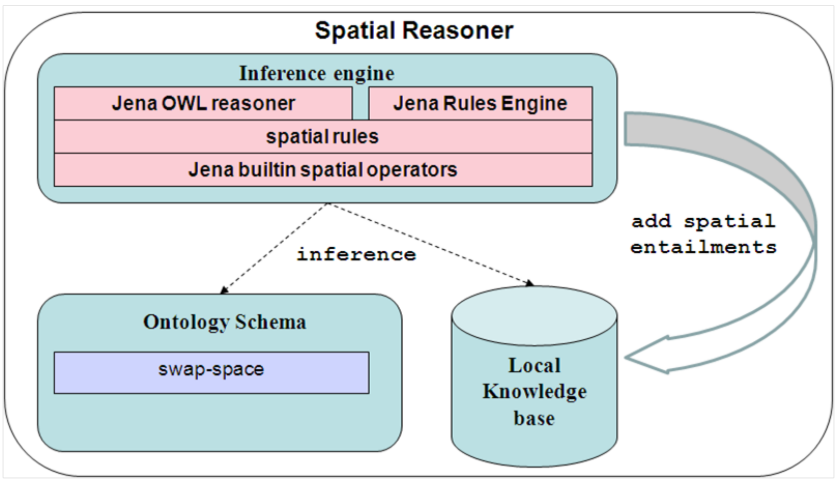


Figure 4.12: SWAP spatial reasoner

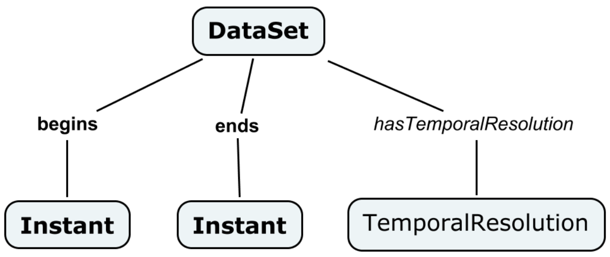


Figure 4.13: The temporal properties of a data set

where spc is the *swap-space* namespace and $spatiallyIntersects$ is a builtin class that determines whether the given spatial objects intersect. The builtins use the JTS topology suite [53] to determine if a specific relation holds between two spatial things. It first converts the spatial things into JTS geometry objects and then calls the appropriate method on the geometry objects to perform the check. The spatial inference rules are provided in appendix A.2.2.

4.4.3 Temporal representation and reasoning

The *swap-time* ontology (see Appendix A.3.1) is used to specify temporal entities and relations. It is based on and incorporates the *OWL-Time* [95, 96] ontology. *OWL-Time* is based on Allen's [20] representation of intervals and considers a temporal entity to be either a temporal *instant* or a temporal *interval*. An *instant* represents a specific point in time while an *interval* represents a time extent with a begin and an end time. A data set can have different temporal properties as shown in figure 4.13.

A $TemporalThing$ is either an $InstantThing$ or an $IntervalThing$. A data set can be viewed as an $IntervalThing$ if it specifies a $begin$ and an end time instant, which allows for other $InstantThings$ or $IntervalThings$ to be temporally related to it. For example if x is an $InstantThing$ and d is a $DataSet$ with a $begin$ and end time, then the binary relation x $inside$ d can either hold or not hold. *OWLTime* supports the binary relations $before$ and $after$ between time instants and the relation $inside$ between a time interval and an instant. The following binary relations are supported between time intervals:

$$intervalEquals,\ intervalBefore,\ intervalMeets,\ intervalOverlaps,\ intervalStarts,$$
$$intervalDuring,\ intervalFinishes$$

and their reverse interval relations:

$$intervalAfter,\ intervalMetBy,\ intervalOverlappedBy,\ intervalStartedBy,$$
$$intervalContainsand\ intervalFinishedBy.$$

A duration of an interval can also be represented using a $durationDescription$. An interval can have many duration descriptions, e.g. 1 day 2 hours, or 26 hours, or 1560 minutes, but only one duration. The $durationOf$ property is used to specify durations of temporal entities. The $inCalendarClockDataType$ property which has a range of *xsd:dateTime* is used to represent time instant values, while the $durationDescriptionDataType$ property which has a range of *xsd:duration* is used to represent time durations.

The *swap-time* ontology extends the *OWLTime* ontology to allow for representing other temporal entities not supported by OWLTime such as:

- Time resolution which represents a uniform gap between observations in a dataset. Currently this is a static value, e.g. 15 minutes describes a gap of 15 minutes between observations. Irregular gaps between observations are currently not supported.

- A temporal relation*follows* that, given a time resolution for a data set, is used to predict the time of the next observation. For example, suppose that a data set has a temporal resolution of 15 minutes. Given the time of an observation, e.g. 12:15, the $follows$ property relates the time of the next observation with the time of the current one. In this case the next observation will be at 12:30, and the follows relation relates these times as: $follows(12:30, 12:15)$

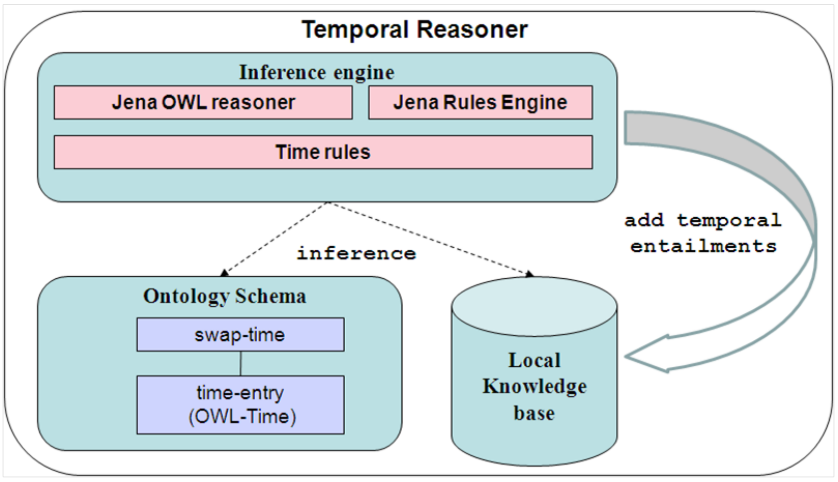


Figure 4.14: SWAP temporal reasoner

Since OWL and Jena do not support reasoning about time, a temporal reasoner (figure 4.14) was developed. The reasoner uses the Jena OWL rule based reasoner but also incorporates a set of additional temporal rules. The temporal rules, which are based on the COBRA temporal reasoner [46], specify the temporal relations in OWLTime. As the standard builtin comparison procedures provided by Jena, such as $equals$, $lessThan$ and $greaterThan$, support time values, no additional builtins were required. The reasoner applies these rules to the instance data found in the local knowledge base to determine which temporal relations hold between temporal entities. A temporal entity is considered to be any individual of type $IntervalThing$ or $InstantThing$.

For example, the two rules for determining whether a time instant is inside a time interval are:

```
(?x tme:inside ?y) <-
    (?x rdf:type tme:InstantThing),
    (?y rdf:type tme:IntervalThing),
    (?y tme:begins ?beginsY), (?y tme:ends ?endsY),
    (?beginsY tme:before ?x), (?x tme:before ?endsY).

(?x tme:before ?y) <-
     (?x rdf:type tme:InstantThing),
     (?x tme:inCalendarClockDataType ?timeX),
     (?y rdf:type tme:InstantThing),
     (?y tme:inCalendarClockDataType ?timeY),
     lessThan(?timeX,?timeY).
```

where tme is the name space of the OWLTime ontology. The first rule stipulates that a time $Instant$ x

is within a time interval y if the starting time of y is before x, and x is before the ending time of y. The second rule uses the $lessThan$ builtin to determine whether the time value of a time instant x is before the time value of another time instant y. The temporal inference rules are provided in appendix A.3.2.

4.4.4 Uncertainty representation and reasoning

The *swap-uncertainty* ontology and the uncertainty reasoner is used to represent and reason about uncertain knowledge. It takes a Bayesian approach to represent the uncertainty associated with observations as and the causal theories that are applied to observations. The uncertainty ontology and reasoner are described in chapter 5.

4.5 THE SWAP TECHNICAL ONTOLOGIES

The SWAP technical ontologies for representing data, agents and tasks are described in this section. The ontologies are provided in Appendix A.5.

4.5.1 Representing data

The *swap-data* ontology specifies data structures that allow agents to dynamically exchange and process data (figure 4.15). It contains descriptions to exchange simple numerical values ($SingleNumericValue$), numerical intervals ($NumericInterval$) and image files containing coverage data ($ImageFileValue$) with the corresponding file format ($ImageFileFormat$). Most data values have associated units of measure. The *swap-data* ontology specifies a general concept of $Units$ and different types of units (figure 4.16). $ThematicUnits$ are specified in domain ontologies, $TemporalUnits$ are specified in *swap-time* and $SpatialUnits$ are specified in *swap-space*.

Since OWL does not provide support for representing list structures, the linked list structure from the *ObjectList* ontology used in OWL-S [6] is used to represent lists. A $list$ contains a $first$ value that specifies the first data element and a $rest$ value that specifies another list containing the other elements in the list. A special list instance, nil, represents the empty list. A list that has a $rest$ value of nil represents the last element of a list.

[6]http://www.daml.org/services/owl-s/1.1/generic/ObjectList.owl

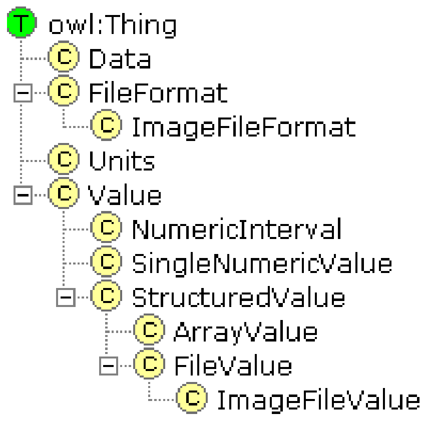


Figure 4.15: The *swap-data* ontology

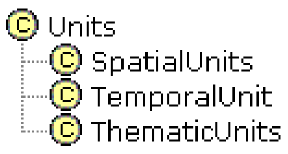


Figure 4.16: Representing units of measure in the SWAP ontology

4.5.2 Representing agents, services and interactions

The *swap-agent* ontology contains concepts to specify agents, the services they host, together with the interaction protocols required to invoke these services.

Figure 4.17 shows a fragment of the *swap-agent* ontology. An *Agent* provides one or more *Services*. A *Service* has a *ServiceDescription* that describes various properties of the *Service*, e.g. a description for a data service would be the details of its *DataSet* (figure 4.4). An agent action usually results in an exchange of messages between agents. Currently, only simple request-response interactions are supported. A message is either a type of $RequestAction$ or a $ResponseAction$, e.g. a $QueryAction$ message is a type of $RequestAction$. An instance of the action is contained in the body of the message. The request and response actions are specified in the *swap-agent* ontology (figure 4.18).

The *Protocol* specifies the type of request and response message required for a specific interaction. For example, the *DataRequest* protocol that is used for retrieving data from a Sensor Agent, specifies *QueryAction* and *InformDataAction* as the request and response messages respectively. For an agent to enact a *DataRequest* interaction it sends an instance of *QueryAction* to the Sensor Agent which responds with an instance of *InformDataAction*. A specific protocol is defined as a subclass of $Protocol$ with

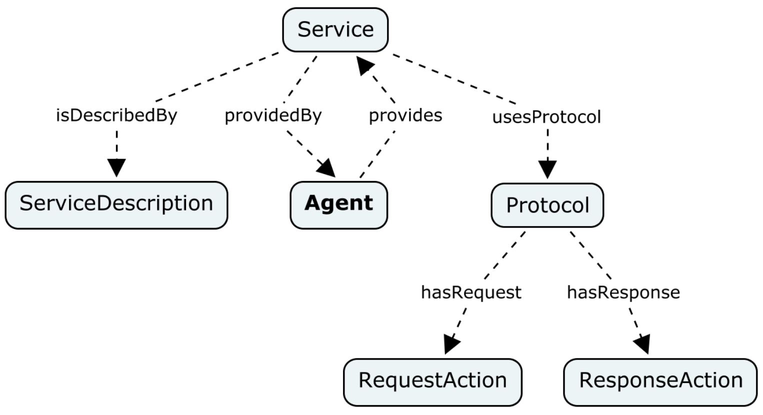


Figure 4.17: Part of the *swap-agent* ontology

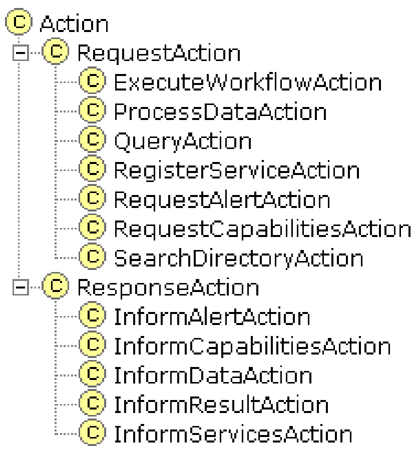


Figure 4.18: Representation of agent actions

restrictions on the message types that can be contained in the $hasRequest$ and $hasResponse$ properties. For the *DataRequest* protocol the request and response actions are restricted to a $QueryAction$ and a $InformDataAction$ respectively.

The *swap-agent* ontology also specifies the structure of an agent message (figure 4.19). A *Message* has: a *content* or body which is an instance of an *Action* (figure 4.18); an optional file *attachment*; a *type* which is the Uniform Resource Identifier (URI) of the *Action*; a *sender*; and a *receiver* which is the *Agent* that sent the message and the *Agent* to which the message was sent. Each message belongs to a conversation which is uniquely identified by a *conversationId*. Since message passing is asynchronous and an agent could have many outstanding request messages at any given time, the *conversationId* is used to map response messages to corresponding request messages.

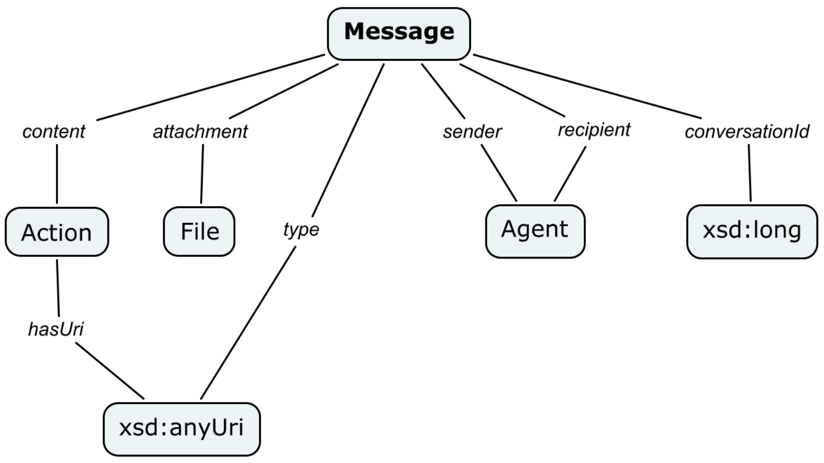


Figure 4.19: A representation of an agent message

4.5.3 Representing workflows using OWL-S

The *swap-task* ontology provides support for representing agent workflows. It is based on the Web Ontology Language for Services (OWL-S) [129, 191]. OWL-S is an extension of OWL that provides primitives for representing executable processes. A process in OWL-S specifies the way in which a client may interact with a service. An atomic process is a description of a service that expects a single message and returns a single message in response. A process can function in two ways. Firstly, it can generate and return some new information based on the world state and information that it is given. Information production is described by the inputs and outputs of the process. Secondly, it can produce a change in the world. This transition is described by the preconditions and effects of the process. A process (figure 4.20) can have zero or more inputs, which represent the information required to perform the process. It can also have zero or more outputs, i.e. the information that is produced by the process. There can be zero or more preconditions, all of which must hold for the process to be successfully invoked. The process can have zero or more effects which are communicated as results to the requestor. Outputs and effects can depend on conditions that hold true of the world state at the time the process is executed.

OWL-S uses the Semantic Web Rule Language (SWRL) [98] for representing variables. SWRL variables are used to represent the *Input* and *output* parameters of a process where each parameter has a type and a value. For object parameters, the type specifies an OWL class, as the URI of the class, and the value must contain an instance of the OWL class. XML data type parameters can also be used by specifying the URI of an XML data type and a corresponding literal value.

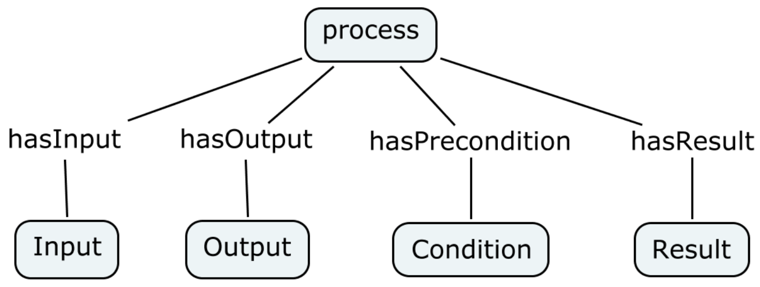


Figure 4.20: Part of the swap-task ontology

A composite process represents a composition of atomic processes. Each message the client sends advances it through the composite process. A composite process is used to represent workflows that combine interactions between multiple agents. The state (inputs, outputs, preconditions and results) after each processing step can be saved and used later in subsequent processing steps. A *CompositeProcess* must have a *composedOf* property which specifies the control structure (*ControlConstruct*). Each *ControlConstruct* is associated with a components property that indicates the nested control constructs from which it is composed and their ordering. To illustrate the OWL-S process representation, consider the composite process, $P3$, shown in figure 4.21. It is composed of two atomic processes $p1$ and $p2$. $P3$ has two input parameters, $p3 - input1$ and $p3 - input2$, and a single output parameter, $p3 - output1$. $P3$ is composed of two processing steps $P1$ and $P2$ which are performed in a simple sequence control construct, i.e. perform $P1$, then perform $P2$. $P1$ has a single input, $p1 - input1$ and a single output $p1 - output1$, while $P2$ has two inputs, $p2 - input1$ and $p2 - input2$, and one output, $p2 - output1$. Suppose, the input parameters for $P3$ are populated and $P3$ is invoked. As specified in the control construct (see figure 4.21), the value for $p3 - input1$, is used to populate $p1 - input1$, and $P1$ is then invoked, to produce $p1 - output1$ and this value is used to populate $p2 - input1$. The value of $p3 - input2$ is used to populate the second input for $p2$, i.e. $p2 - input2$, and $P2$ is invoked. $P2$ produces output $p2 - output1$. $p2 - output1$ is used to populate the only output for $p3$, i.e. $p3 - output1$ and the composite process $P3$ completes. Input and output parameters can be simple XSD literals, e.g. $XSD : Integer$ or OWL object types.

Besides the *Sequence* control construct described above, OWL-S also provides support for the following control constructs: *Split*, *Split + Join*, *Choice*, *Any-Order*, *Condition*, *If-Then-Else*, *Iterate*, *Repeat-While*, and *Repeat-Until*. The OWL-S Editor plugin [62] in Protege provides a graphical environment for constructing and visualising OWL-S processes and was used to construct the process in figure 4.21.

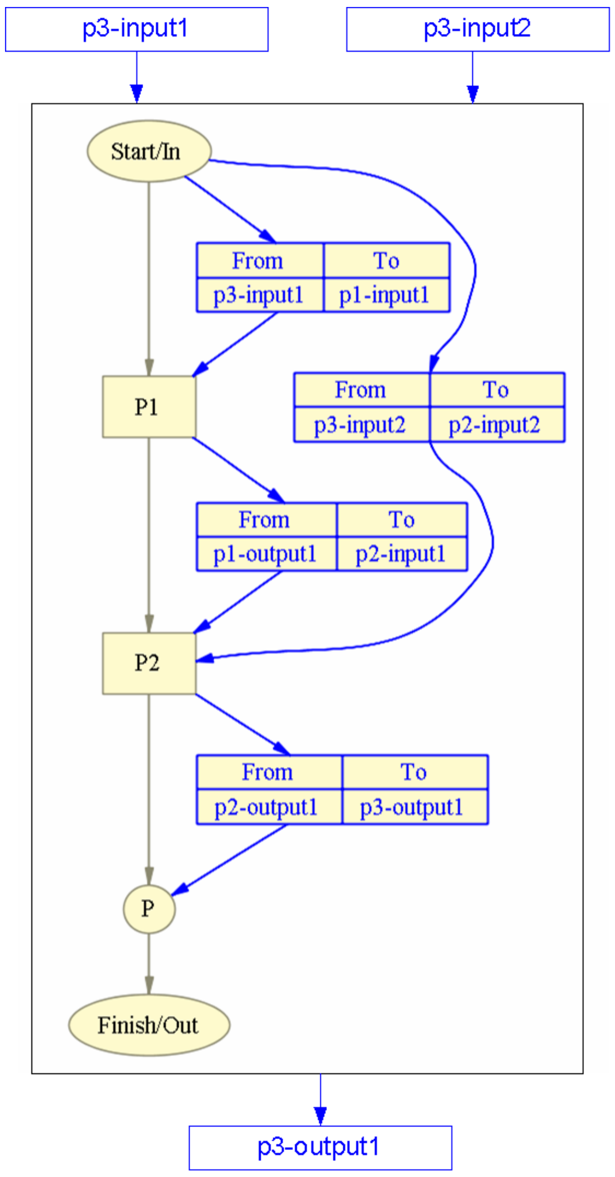


Figure 4.21: A sample composite process, $P3$, represented in OWL-S

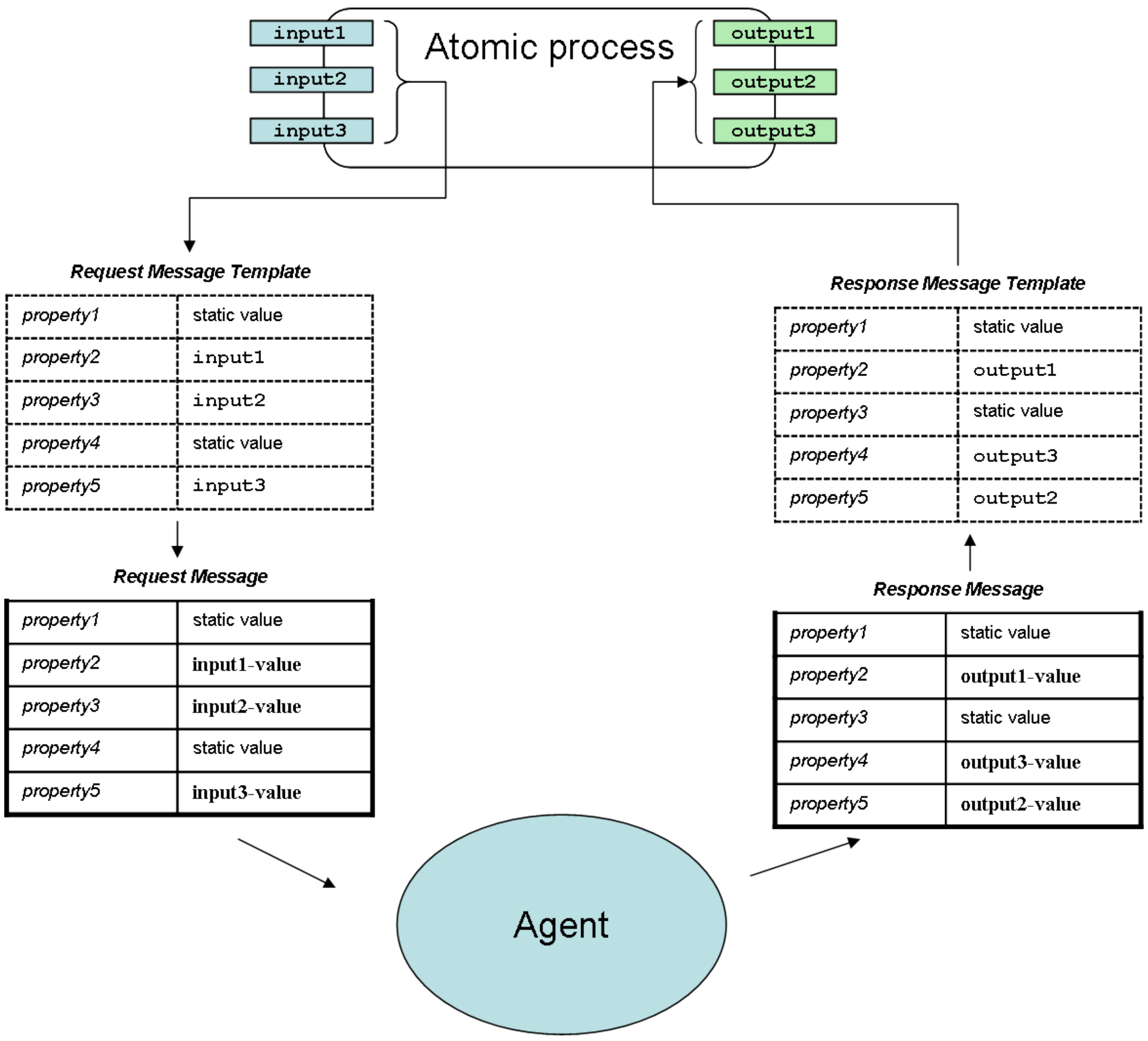
4.5.4 Incorporating agent services into OWL-S

While OWL-S supports the composition and execution of web services, it does not support agent composition and invocation. An *agent-to-process-mapping* was created, that extends OWL-S, and allows for representing executable agent workflows as OWL-S processes.

An OWL-S atomic process is used to represents an invocation of a specific agent service. The process to agent message mapping, shown in figure 4.22, uses the values of the input parameters to construct an agent request message and populates output parameter values with values extracted from the agent response message. It consists of an input mapping that binds input parameter values to variables in a *request message template* and an output mapping that binds values from a response message to variables contained in a *response message template*.

When an atomic process is executed, a copy of the *Request Message Template* is created. Static property values are copied as is, while properties with input variables are replaced with the values of the corresponding input parameters. This message is then sent to the appropriate agent which responds with a response message. The *Response Message Template* specifies which property values are to be extracted from the response message. When the response message is received, the template is searched for variable properties, i.e. properties that contain output variables. The process output parameters are then populated with the corresponding property values extracted from the response message. In this way values of input parameter values are used to construct a valid request message to invoke an agent service. Relevant values from the response message are extracted and used to populate the output parameters of the process.

One feature of this approach, as shown in figure 4.22, is the differentiation between static and variable values in the message templates. The request message template allows for specifying invocation parameters that may change for each invocation and that must be supplied as inputs to the process. The remaining invocation parameters take on default or static values that are specified in the template and remain constant for each invocation. For an agent that requires multiple invocation parameters, predefined or default values can be provided for certain parameters. Only the variable parameter values must be provided as inputs to the process. One current limitation of this approach is that only input variables can contain literals values. Output variables are restricted to object values.



Request Message Template

property1	static value
property2	`input1`
property3	`input2`
property4	static value
property5	`input3`

Request Message

property1	static value
property2	**input1-value**
property3	**input2-value**
property4	static value
property5	**input3-value**

Response Message Template

property1	static value
property2	`output1`
property3	static value
property4	`output3`
property5	`output2`

Response Message

property1	static value
property2	**output1-value**
property3	static value
property4	**output3-value**
property5	**output2-value**

Figure 4.22: Mapping OWL-S processes to agent services

The Mindswap OWL-S API [7] is used for the execution of OWL-S workflows. The API was extended to allow for execution of agent services using the *agent-to-process-mapping* described above.

4.6 AGENT DISCOVERY AND INVOCATION

The ontology infrastructure provides a common semantic framework for agents to interpret and reason about sensor data. However, agents must be able to discover and be able to send appropriate request messages to invoke these services. Agents that offer services must be able to receive, process and respond to these requests.

4.6.1 The SWAP Directory Agent

The SWAP Directory Agent provides a searchable repository of $Service$ instances that describe the services offered by agents in the system. SWAP agents register their services with the SWAP Directory Agent so that their services can be discovered and used by other agents.

4.6.1.1 External Representation

The SWAP Directory role is described in table 4.1 and the *SWAPServiceRegistration* and *SWAPDirectorySearch* protocols for registering and searching for a service are shown in table 4.2.

An agent registers its *Service* via a *RegisterServiceAction* message. A *RegisterServiceAction* instance has a *provides* property containing a *Service* instance (figure 4.17). The *Service* instance has a service description describing the service being offered, the details of the agent offering the service, and a protocol that provides request and response message templates (figure 4.22) for invoking the service and interpreting the response from the service. Different agents offer different types of services and accordingly have different *ServiceDescriptions*. Sensor Agents that offer access to data sets provide a description of their data set. Tool Agents that offer data processing services, Modeling Agents that offer prediction services and Workflow Agents that provide coordination services provide descriptions of their input and output parameters. Application Agents provide a description of the phenomena for which they are able to provide alerts. Services are searched by type, i.e. *DataSet*, *Tool*, *Modeling*, *Workflow* and *Application*,

[7] http://www.mindswap.org/2004/owl-s/api

Table 4.1: SWAP Directory role schema

Role Schema:	*SWAP Directory*
Description	Hosts a service repository containing service entries for all agents that offer a service in the system. Answers search requests for services. Allows agents to register a new service or to update a service entry for an existing service.
Protocols and Activities	*SWAPServiceRegistration*, *SWAPDirectorySearch*
Permissions	Delete, add, modify registry (local)
Responsibilities	• Maintain service registry • Respond to service query requests

and by specifying constraining values for properties specific to the service description for this service type. The SWAP Directory Agent responds with a *InformResultAction* with the results of the registration, i.e. whether the registration was successful or not.

Other agents are able to search the SWAP directory to discover these services. A search request for agent services is done via a *SearchDirectoryAction* message. A *SearchDirectoryAction* instance specifies the search criteria. It has a single property *hasServiceDescriptionValue* that specifies a *ServiceDescription* instance. Consider Sensor Agents that offer *DataSets*. *DataSets* can be restricted by specifying combinations of thematic, temporal and spatial properties. Suppose that *ds* is a *DataSet*, *st* is some *SpatialThing*, *ti* is a time *Instant* and *p* is some thematic *ObservedProperty*, then, *ds intersects st*, *ds intContains ti*, *ds observesProperty p* are examples of spatial, temporal and thematic constraints on *DataSet* services. A *SearchDirectoryAction* with value *ds* for *hasServiceDescriptionValue* will request all *DataSet Service* instances with descriptions that satisfy these constraints. Matching services are sent back via an *InformServicesAction* with the matching services specified by the *hasService* property.

4.6.1.2 Internal Operation

The SWAP Directory Agent handles two types of requests, service registrations and service search requests. Service registrations are processed by adding the service instance contained in the *RegisterServiceAction* to the local knowledge base or updating the entry if it already exists. The result of this operation is sent to the sender via an *InformResultAction* message.

Table 4.2: The service registration and directory search protocol schemas

Protocol	*SWAPServiceRegistration*
Initiator	Any agent
Participants	SWAP Directory Agent, any SWAP agent that offers a SWAP service
Message Types	*RegisterServiceAction*, *InformResultAction*
Inputs	Service instance
Outputs	Result of Registration
Description	An agent registers the service it provides by sending a *RegisterService-Action* to the SWAP Directory Provider, which responds with an *Inform-ResultAction* specifying whether the registration was successfull.
Protocol	*SWAPDirectorySearch*
Initiator	Any agent
Participants	Directory Agent, initiating agent
Message Types	*SearchDirectoryAction*, *InformServicesAction*
Inputs	Service criteria
Outputs	Zero or more services that match the input criteria
Description	An agent can search the service repository, by specifying thematic, spatial and temporal criteria. The SWAP Directory provider responds with services that match the criteria.

Search request messages, i.e. *SearchDirectoryAction* messages, are more complex to handle. Three separate queries for *Service* instances, using the SWAP reasoning engine, are performed on the local knowledge base (KB). Thematic constraints are identified and extracted from the search request and a thematic query is performed on the KB using the thematic reasoner. Spatial constraints are then identified and extracted from the search request and a spatial query is performed on the KB using the spatial reasoner. Finally the temporal constraints are identified and extracted from the search request and a temporal query is performed on the KB using the temporal reasoner. Those *Service* instances that appear in the results of all three queries form the final results of the search request. An *InformServicesAction* message containing these *Service* instances is composed and sent to the requesting agent.

4.6.2 Service Composition

Composing agent workflows consists of selecting one or more agent services and determining the sequence in which they must be executed. A composite OWL-S process, as described in section 4.5.3, is then constructed. An atomic process is created for each service instance (see figure 4.22). The *usesProtocol* property of a *Service* instance contains the request and response message templates that are required for creating the atomic processes. Since the automatic workflow composition is not supported workflows are composed manually. Workflow Agents are responsible for hosting and executing workflows.

4.7 SWAP INTERNAL AGENT ARCHITECTURE

This section describes an internal agent architecture that enables agents to receive, process and respond to incoming requests and information. The approach attempts to bridge the gap between the declarative programming approach used for programming ontology and agent systems and the imperative approach used in object oriented development. It uses a data mapping API to allow agent developers to incorporate and integrate existing programming libraries, sensor data stores and GUI components available to the geoinformatics community.

4.7.1 Internal Agent Architecture Overview

The internal architecture of an agent is shown in figure 4.23. An agent uses the shared ontologies and shared rules to interpret and reason about incoming messages. Custom rules and ontologies, which encode additional knowledge required for its internal operation, may also be specified. This allows for different agents to respond differently to request messages as determined by their custom ontologies and custom rules. The agent execution engine provides the runtime environment for an agent. It implements a communication infrastructure that provides a transport layer for message passing, message handlers to handle different messages and a message queue that redirects incoming messages to the appropriate message handlers. These message handlers use the reasoning engine to interpret, reason about and respond to these messages.

4.7.2 Incorporating GIS development libraries

Various initiatives are underway in the geoinformatics community to provide opensource development libraries to serve, access, process and visualise sensor data. The SWAP architecture attempts to leverage and reuse these libraries where possible. Some of the GIS Java based development libraries that are used within SWAP are described below.

52 North - OXFramework: 52 North's [8] OGC Web Service Access Framework (OX-Framework) [37] supports access to different kinds of OGC Web Services, and the visualisation and processing of queried

[8] http://52north.org

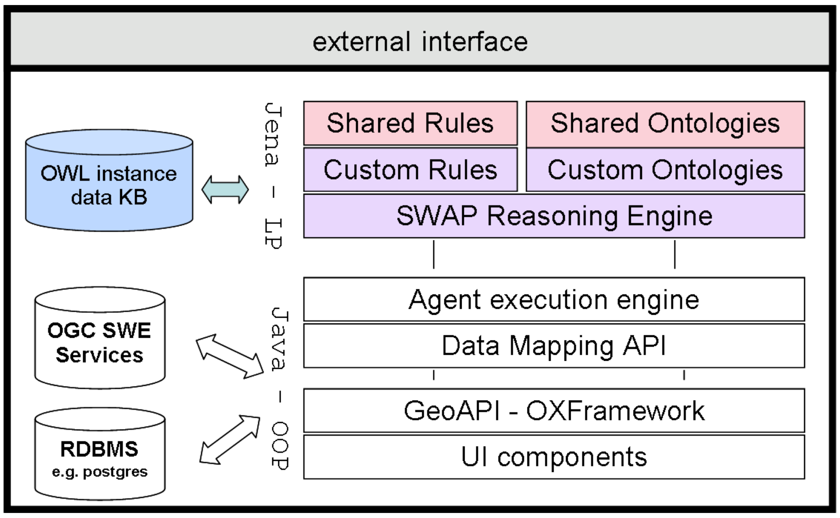


Figure 4.23: The SWAP internal agent architecture

data. It offers developers a customisable and extendable library of cooperating classes with the emphasis on reusability. It facilitates access to sensor data sources, such as SWE services and spatial databases, and provides Java-Swing based graphical components for building client applications for retrieving and visualising geographical data.

OpenGIS and GeoAPI: OpenGIS is a set of interface and encoding specifications produced by the Open Geospatial Consortium (OGC) [9] that supports transparent access to heterogeneous geodata and geoprocessing resources in a networked environment. It enables developers to write interoperating components that provide these capabilities. An OpenGIS compliant software product is a software product that follows the OpenGIS specifications and is able to interoperate with other OpenGIS implementations. The GeoAPI [10] project provides interface-only APIs derived from OGC and International Standards Organisation (ISO) standards. GeoAPI is implementation independent and is freely available as a set of Java interfaces from Sourceforge[11].

Java Topology Suite (JTS): The JTS Topology Suite[12] is an open source Java implementation of the OGC Simple Features Specification for SQL [150]. It is OpenGIS compliant and implements the two dimensional spatial algorithms required to calculate the spatial relations described in 4.4.2.

[9]http://www.opengeospatial.org
[10]http://docs.codehaus.org/display/GEO/Home
[11]http://geoapi.sourceforge.net
[12]http://www.vividsolutions.com/JTS/

Table 4.3: Mappings between SWAP ontology structures and OpenGIS Java class structures

Ontology	***Concept/Representation***	***Library***	***Interface/class***
swap-time	TemporalThing	GeoAPI	ITime
swap-time	InstantThing	GeoAPI	TimePosition
swap-time	IntervalThing	GeoAPI	TimePeriod
swap-space	Geometry	JTS	Geometry
swap-space	BoundingBox	GeoAPI	IBoundingBox
swap-space	PointCoordinate	JTS	Coordinate
swap-space	List of PointCoordinates	JTS	CoordinateList
swap-space	List of SingleValues with spatial extent	OXFramework	Collection<OXFFeature>
swap-data	ImageFileValue	core Java	File

4.7.3 Mapping between ontology data instances and OpenGIS data objects

SWAP provides a data mapping API that allows agent developers to incorporate functionality from existing Java based GIS libraries. The data mapping API, shown in figure 4.23, transforms spatial and temporal ontology data instances into corresponding OpenGIS data structures. For example, the Geometry class in the JTS Java libraries naturally maps to the *Geometry* concept defined in the *swap-space* ontology. The mappings that are currently provided are shown in table 4.3. For each mapping, the ontology concept, the corresponding Java representation class and the library to which the classes belong are shown.

The data mapping API is used within an agent to convert incoming ontology instance data to OpenGIS Java data objects. These objects can then be processed using the functionality provided within OpenGIS Java libraries. Once the processing is completed, the data mapping API is used to convert the results into ontology instances that can be exchanged with other agents.

Integration with SWE services The OXFramework provides client side APIs for accessing spatial databases and OGC SWE services. The data mapping API can be used to convert agent data requests to OXFramework objects. The appropriate OXFramework client side APIs can be used to access and request data from a relational database or a SWE service. The returned data can be converted into SWAP ontology instances. In this way SWAP agents can access and incorporate data from OGC SWE services and spatial databases.

User interface support The OXFramework also provides Java-Swing components to visualise sensor data in client applications. These components can now be incorporated within SWAP User Agents for data visualisation.

Integration with other Java development libraries: Since the GeoAPI and OpenGIS interfaces are standard, any OpenGIS compliant libraries can be incorporated in SWAP by using the data mapping API. For example, the open source GeoTools [13] Java library is used to read GeoTiff files.

4.8 SUMMARY

This chapter described the design and operation of the Sensor Web Agent Platform. The abstract architecture, described in section 4.2, guides the design of agent based Sensor Web applications. The architecture provides three abstraction layers and six abstract agent types that allows for a clear separation of concerns. Sensor specific functionality is restricted to Sensor Agents in the Sensor Layer. Data processing, data analysis steps and prediction models are applied to sensor data by agents in the Knowledge Layer. Results from the Knowledge Layer form the basis for alerts that are delivered to end users via alerting applications offered by Application Agents at the Application Layer.

The SWAP ontology infrastructure is described in sections 4.4 and 4.5. The conceptual ontologies (section 4.4) separate the representation of real world entities into four aspects or dimensions, i.e. spatial, temporal, thematic and uncertainty. The approach is based on the human cognitive system. It aims to ease the development, management and sharing of conceptual models for complex real world observations. Having four separate representation and reasoning systems also increases the flexibility and speed of the reasoning process. Since the reasoners operate independently, each reasoner can be customised, extended or exchanged without affecting the others. The different reasoners currently use different inferencing engines: the thematic reasoner uses a Pellet reasoner; the temporal and spatial reasoners use a Jena rule-based engine; and the uncertainty reasoner (discussed in the next chapter) uses a Bayesian inference engine. The technical ontologies (section 4.5) are used to specify data structures which are used by agents to access and interpret incoming data. The technical ontologies provide support for describing agents, the services they host, and the interaction protocols for invoking these services. Agents register their services with a SWAP Directory Agent (section 4.6). Services are discovered by querying the SWAP Directory Agent. Agent services can be assembled into executable agent workflows. Section 4.5.4 describes how OWL-S is extended and used for representing executable workflows.

[13] http://geotools.codehaus.org/

The internal agent architecture, described in section 4.7, facilitates the development of individual ontology driven agents. A key feature is the data mapping API which allows for converting between ontology instance data and OpenGIS Java objects. This allows agent developers to incorporate open source OpenGIS libraries or even remote OGC services to perform the internal processing of an agent.

The next chapter describes how Bayesian Networks are used to represent and reasoning about uncertain knowledge.

Chapter 5

INCORPORATING UNCERTAINTY INTO SWAP

Most geographical data contain some degree of uncertainty. This results from inaccuracies inherent in sensor readings or from inaccuracies of theories or algorithms used to produce the data. Representation and reasoning about uncertainty is required at all layers of the SWAP architecture. At the Sensor Layer, sensors measure entities in the real world. This is prone to a number of errors and inaccuracies including unpredictable changes in the physical environment, inaccuracies in spatial representation systems and degradation of sensors over time. In the Knowledge Layer, the results of individual Tool and Modeling agents also have an element of uncertainty. The accuracy of the results produced by Tool agents are influenced by the algorithms and configuration parameters being used as well as specific characteristics of the data, such as the spatial and temporal resolution. Modeling agents provide predictions that inherently have an element of uncertainty. Workflows combine results of Tool and Modeling agents to extract complex information that is used to produce alerts in the Application Layer. The uncertainty associated with component Sensor, Tool, Modeling and Sensor agents contribute to uncertainty in the results of the workflow. The nature and availability of sensor data, the accuracy and completeness of the theory that underpins the choice, and the sequence of the processing steps may contribute an additional element of uncertainty. The information produced by workflows are frequently approximations or best guesses.

This chapter describes the SWAP uncertainty ontology and reasoner. SWAP takes a Bayesian probability approach to represent and reason about uncertainty on the Sensor Web [159, 171]. Bayesian probability is well suited for dealing with uncertainty on the Sensor Web: where no complete theory is available; or where it exists it might be too tedious or complex to incorporate all the required observations; or where all the necessary observation data is not available [171].

5.1 BAYESIAN PROBABILITY

The classical ”frequentist” approach to probability is based on measuring the relative frequency of different outcomes based on previous random observations of some event, e.g. the chance of having an outcome of heads when tossing a coin. When the number of random observations of a certain event is very large, the relative frequency of the observations is a near exact estimate of the probability distribution of the event. Since frequencies can be measured, this frequentist interpretation of probability is deemed to be an objective measure for dealing with random phenomena.

However, many events have no historical precedent, may occur infrequently or are difficult to observe. Examples of such events are a tsunami occurring on the east coast of South Africa or the temperature of the earth's core being greater than 5700 Kelvin. The Bayesian approach to probability differs from the classical frequentist approach. The Bayesian probability of an event [171] represents the degree of belief that the event occurred given the occurrence of other events. Events that are easily observable often cause or influence the occurrence of other events which may not be as easily observable. Beliefs are governed by a probability distribution that can be updated by making use of the observed data. The personal belief of the occurrence of an event starts with a given distribution and is known as the prior distribution. Bayesian Networks capture dependency relations as well as the degree of the dependency between events within conditional probability distributions. By recording the occurrence of certain events, which are readily observable, a Bayesian Network can be used to determine the probability of the occurrence of other events which are directly or indirectly affected by these events. In Bayesian networks these casual relationships are applied to observational data in order to obtain a posterior probability distribution by updating the prior belief.

5.1.1 Bayesian Networks

A Bayesian Network is represented as a directed acyclic graph. The nodes of the graph represent random event variables. Influence or dependency relations between variables are represented by directed arcs between the nodes. The direct parents of a node are considered to be all nodes that have a direct influence on this node. The children of a node are considered to be all nodes on which this node has a direct influence.

Assume that all event variables $A, B, C \ldots$ are finite and discrete, i.e. each variable has a finite number of states. An event e is represented by assigning a state to a variable. Boolean variables take on one of two states, either $True$ or $False$, i.e. indicating whether the event occurred. Suppose that A and B are the only variables in a domain. If A has two states and B has three states then five possible atomic events, $E = \{a_1, a_2, b_1, b_2, b_3\}$, can be represented in this domain.

5.1.1.1 Prior probability

The prior (or unconditional) probability of an event is considered to be the probability that an event occurs given no other information about the occurrence of other events in the domain. For an event e the prior probability of the event is written as $P(e)$. For example, the prior probability that event a_1 (event variable A taking on its first state) will occur is $P(a_1)$. A prior probability distribution is specified for all independent variables, i.e. variable that have no parents or direct influences. If event variables A and B (from the example above) are independent, then the prior probability distribution of A is $\mathbf{P}(A) = \{P(a_1), P(a_2)\}$ and for B is $\mathbf{P}(B) = \{P(b_1), P(b_2), P(b_3)\}$. The joint probability distribution, $\mathbf{P}(A, B, C, \ldots)$, specifies all the combinations of all values on a set of random variables, i.e. it represents a complete set of the probabilities of all possible states of the domain. In this example $\mathbf{P}(A, B)$ is a $6x2$ table, where each row represents one possible state of the domain. In this example, the domain can be in one of six states at any given time.

5.1.1.2 Conditional probability

Once some evidence has been observed about the states of certain variables that influence other previously random variables, prior probabilities are no longer appropriate. Instead, conditional (or posterior) probabilities are used. Suppose that the state of B has an influence on the state of A then the conditional probability of a_1 given that B is in state b_1 is written as $P(a_1|b_1)$. Conditional probabilities can be defined in terms of unconditional probabilities:

$$P(a_1|b_1) = \frac{P(a_1 \wedge b_1)}{P(b_1)} \tag{5.1}$$

where $P(b_1) > 0$

This equation can also be written as $P(a_1 \wedge b_1) = P(a_1|b_1)P(b_1)$ or because of the communicativity of conjunction as $P(a_1 \wedge b_1) = P(b_1 \wedge a_1) = P(b_1|a_1)P(a_1)$. Equating the two right hand sides gives: $P(a_1|b_1)P(b_1) = P(b_1|a_1)P(a_1)$. Then dividing both sides by $P(a_1)$ results in:

$$P(b_1|a_1) = \frac{P(a_1|b_1)P(b_1)}{P(a_1)} \tag{5.2}$$

Equation 5.2 is known as Bayes' theorem.

An entry in the joint probability distribution is one particular assignments to each variable X_i in the Bayesian Network or $P(X_1 = x_1 \wedge \ldots \wedge X_n = x_n)$. This can also be written as $P(x_1, \ldots, x_n)$ and the value is calculated by

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n}(P(x_i|parents(X_i)) \tag{5.3}$$

where $parents(X_i)$ is the values of the variables that are direct parents of X_i.

5.2 BAYESIAN NETWORKS FOR THE SENSOR WEB

The occurrence of natural phenomena are sometimes difficult to detect. Certain phenomena exhibit consistent symptoms that are more easily detected. These symptoms can serve as an indicator for the occurrence of the phenomena. The analysis of observations from multiple sensors may be required to determine the existence of the symptoms of specific phenomena. A Bayesian Network can be used to determine the probability of the occurrence of a phenomena given one or more observable symptoms. In such a Bayesian Network two types of discrete random variables are required:

Observable event variables represents the occurrence of a symptom of a phenomena and is a qualitative measure for an observation. The variable must specify the entity, the characteristic of the entity being observed, as well as the property that contains the numerical value for the observation. The states are predefined numerical ranges, corresponding to qualitative descriptions. For example, wind speed is often used as an indication of the extent of a storm: from 6 to 49 km/hr is a breeze; 50 to 89 km/hr is a gale; 90 to 117 km/hr is a storm and speeds greater than 118 km/hr is indicative of a hurricane [1]. Observation instance data can be used to populate observable event variables..

Inferred event variables represents the occurrence of a phenomena, e.g. a hurricane. A phenomena is represented as a subclass of *Phenomenon* in the *swap-theme* ontology. When a phenomena is

[1]Using the Beaufort scale from http://en.wikipedia.org/wiki/Beaufort_scale

detected, an instance of the appropriate *Phenomena* class is created. These events are inferred from observable events or other inferred events. Even though these variables are intended for representing the occurrence of a phenomena, they can be used to represent any event that is not easily or directly measurable.

An occurrence of an observable event is determined by evaluating measurements of some observed property of an observed entity, e.g. the speed of the wind above a certain threshold results in the occurrence of a ”strong wind” event. These observable events are used to infer the probability of the occurrence of other events, e.g. a very strong wind is a symptom of a hurricane event. Thus, by analysing one or more measurements certain phenomena can be detected, e.g. a wind speed above 118 km/hr and an air pressure lower than 97.7 kPa can be considered to be symptoms of a hurricane event [2].

The proposed Bayesian Network model assumes that all variables are discrete and represent events that occur at the same time and space. The current model does not cater for the influence of past or future events, or the influence of events occurring at different locations. An investigation can be conducted into how the model can be extended in this regard, but is out of the scope of this work.

5.2.1 An ontology for Bayesian Networks

The approach builds on the ontology from the BayesOWL [57, 58] approach. BayesOWL proposes five classes (see figure 5.1) to represent a Bayesian Network (BN) within an ontology, i.e. *ProbObj*, which could either be a *CondProb* or a *PriorProb*, *Variable* and *State*. It also provides four properties, i.e. *hasClass*, *hasProbValue*, *hasVariable* and *hasCondition*. The structure of the BayesOWL ontology is shown in figure 5.2. A *ProbObj* has a probability value (*hasProbValue*) of some variable (*hasVariable*) being true. A Variable represents whether an instance is a member (*rdf:type*) of the specified class (*hasClass*). A variable can be in one of two states, either *True* or *False*, i.e. whether the instance is a member of the class or not.

The *swap-uncertainty* ontology, shown in figure 5.3, extends the BayesOWL ontology. The structure of the ontology is shown in figures 5.4 and 5.5. The complete ontology is given in Appendix A.4. The following extensions were made to the BayesOWL ontology:

[2]http://hypertextbook.com/facts/StephanieStern.shtml

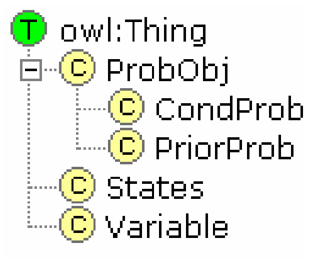


Figure 5.1: Classes in the BayesOWL ontology

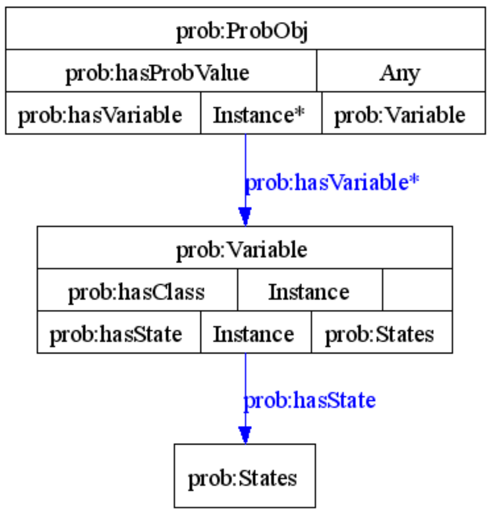


Figure 5.2: The BayesOWL ontology

- Specialising the State class to allow for user defined *DiscreteStates* rather than just Boolean states (True, False). The *DiscreteRangeState* could be a numeric interval for numerical data type properties or a *SingleNumericState* for single numeric values.

- BayesOWL caters only for instance classification. The *ProbObj* class in BayesOWL only provides the properties *hasProbValue* and *hasVariable*. This implicitly assumes a state of *True*. To cater for other states, the *hasState* property is added to *ProbObj*. In this way the state is made explicit and other discrete states can be represented.

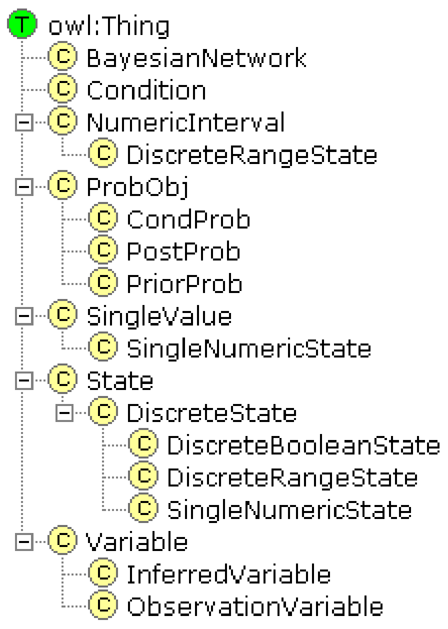
- BayesOWL supports only boolean states. The *CondObj* class has a single property, *hasCondition*, with range *Variable*, which implicitly represents a state of *True*. A new class, *Condition*, is introduced with properties *hasVariable* and *hasState* to allow for multiple states of influencing variables. Thus, a conditional probability may have one or more conditions representing the different states of an influencing variable.

- The BayesOWL *Variable* class uses the *hasClass* property to represent whether an instance is an instance (*rdf:type*) of a specific class. A new property *hasProperty* is introduced so that the *Variable* class can represent other properties, besides class type (*rdf:type*).

- Manually creating each condition and conditional probability statement can be tedious. Another property, *influencedBy*, is added to represent the arcs between variable nodes in a BN graph. By adding this property, the influence relations between variables, and the structure of a BN can be generated automatically. It is populated with default values which can be modified to reflect the actual probability values.

5.2.2 Specifying Bayesian Networks

A Bayesian Network (BN) is represented using the *swap-uncertainty* ontology shown in figure 5.4. Each node in the BN represents either an observation or an inferred variable. An observation variable represents the observation value (*hasValueProperty*) for some observed property (*observesProperty*) of the observed entity (*observesEntity*). An inferred variable represents the occurrence of some phenomena (*hasClass*). The *influencedBy* property specifies the variables that influence the state of this variable.



```
owl:Thing
    BayesianNetwork
    Condition
    NumericInterval
        DiscreteRangeState
    ProbObj
        CondProb
        PostProb
        PriorProb
    SingleValue
        SingleNumericState
    State
        DiscreteState
            DiscreteBooleanState
            DiscreteRangeState
            SingleNumericState
    Variable
        InferredVariable
        ObservationVariable
```

Figure 5.3: Classes in the *swap-uncertainty* ontology

An example to determine the probability of the occurrence of a hurricane is described below to illustrate how the *swap-uncertainty* ontology is used to represent a BN. Figure 5.6 shows a fragment of the *swap-theme* ontology that incorporates selected concepts from the SWEET substance, property and phenomena ontologies to describe the thematic properties of air pressure, wind speed and the hurricane phenomenon. Figure 5.7 shows an air pressure observation with *Air* and *AtmosphericPressure* as the observed entity and observed property respectively. Figure 5.8 shows a wind speed observation with *Wind* and *Speed* as the observed entity and observed property respectively. Suppose that low air pressure and high wind speed are symptoms of a hurricane. Then the Bayesian Network graph model to determine the probability of the occurrence of a hurricane phenomenon is shown in figure 5.9. To facilitate mapping between the ontology and the graph model, the full URI is used to label the variables. The BN contains three variables:

- *wind_speed_var* represents wind speed observations (figure 5.8)
- *air_pressure_var* represents air pressure observations (figure 5.7)
- *is_hurricane_var* represents instances of the *Hurricane* class (figure 5.6)

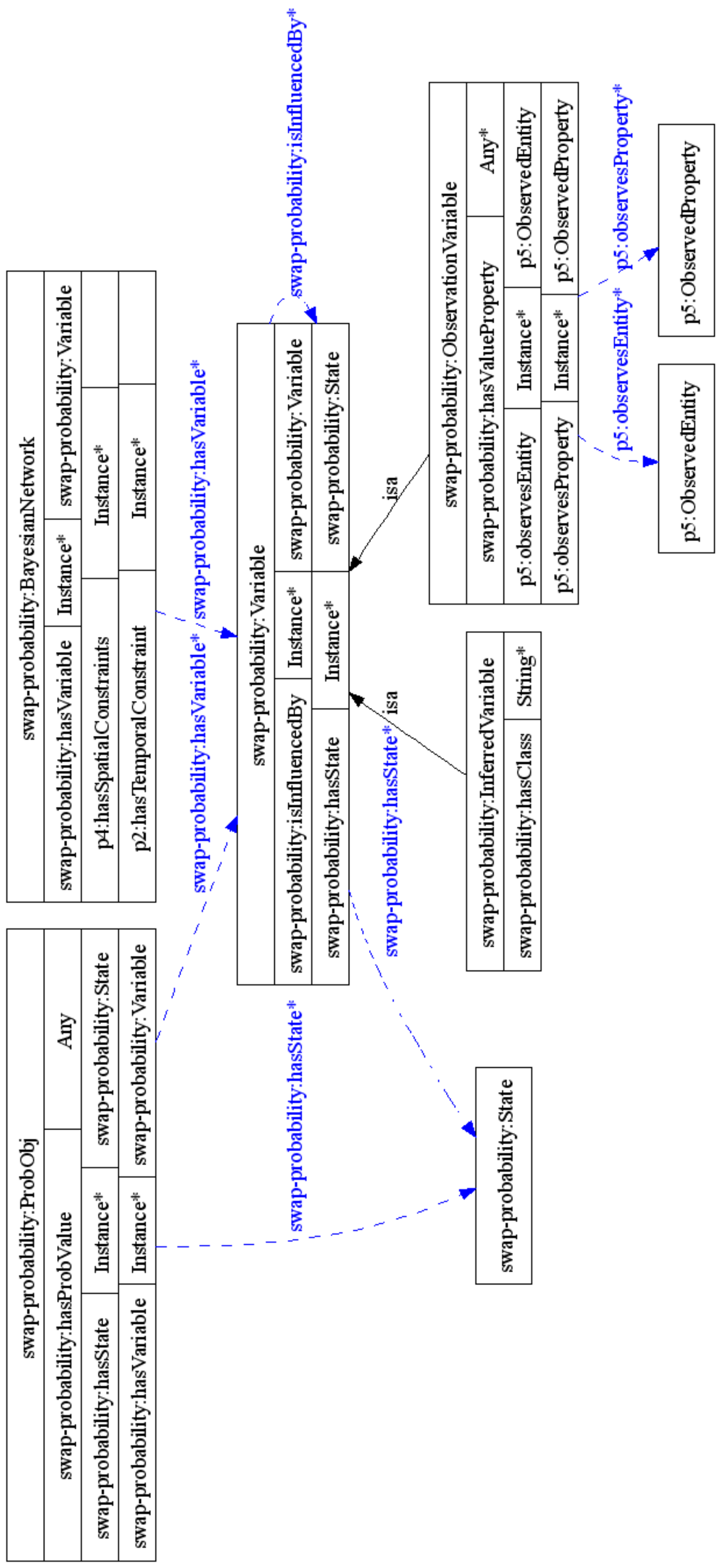

swap-probability:BayesianNetwork
swap-probability:hasVariable
Instance*
swap-probability:Variable
p4:hasSpatialConstraints
Instance*
p2:hasTemporalConstraint
Instance*
swap-probability:ProbObj
swap-probability:hasProbValue
Any
swap-probability:hasState
Instance*
swap-probability:State
swap-probability:hasVariable
Instance*
swap-probability:Variable
swap-probability:Variable
swap-probability:isInfluencedBy
Instance*
swap-probability:Variable
swap-probability:hasState
Instance*
swap-probability:State
swap-probability:ObservationVariable
swap-probability:hasValueProperty
Any*
p5:observesEntity
Instance*
p5:ObservedEntity
p5:observesProperty
Instance*
p5:ObservedProperty
swap-probability:InferredVariable
swap-probability:hasClass
String*
swap-probability:State
p5:ObservedEntity
p5:ObservedProperty
swap-probability:isInfluencedBy*
swap-probability:hasVariable*
swap-probability:hasVariable* / swap-probability:hasVariable*
swap-probability:hasState*
swap-probability:hasState*
isa
isa
p5:observesEntity*
p5:observesProperty*

Figure 5.4: The fragment of the *swap-uncertainty* ontology for representing a Bayesian Network

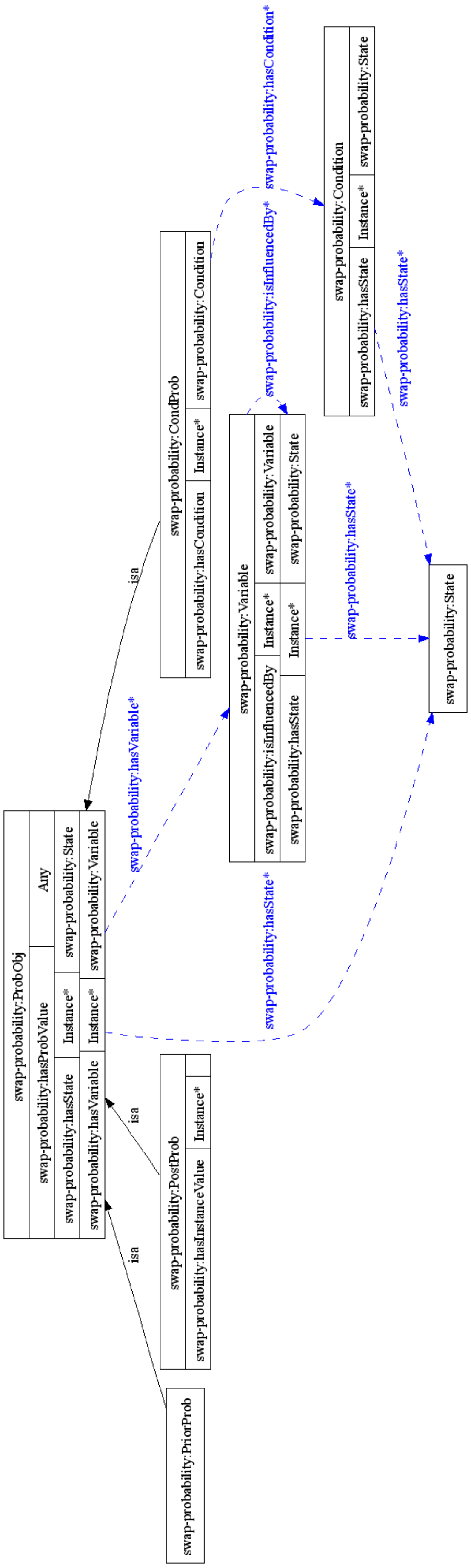

swap-probability:ProbObj
swap-probability:hasProbValue
Any
swap-probability:hasState
Instance*
swap-probability:State
swap-probability:hasVariable
Instance*
swap-probability:Variable
swap-probability:CondProb
swap-probability:hasCondition
Instance*
swap-probability:Condition
swap-probability:PostProb
swap-probability:hasInstanceValue
Instance*
swap-probability:PriorProb
swap-probability:Variable
swap-probability:isInfluencedBy
Instance*
swap-probability:Variable
swap-probability:hasState
Instance*
swap-probability:State
swap-probability:Condition
swap-probability:hasState
Instance*
swap-probability:State
swap-probability:State
isa
isa
isa
swap-probability:hasVariable*
swap-probability:hasState*
swap-probability:hasState*
swap-probability:hasState*
swap-probability:isInfluencedBy*
swap-probability:hasCondition*

Figure 5.5: The fragment of the *swap-uncertainty* ontology for representing probability statements

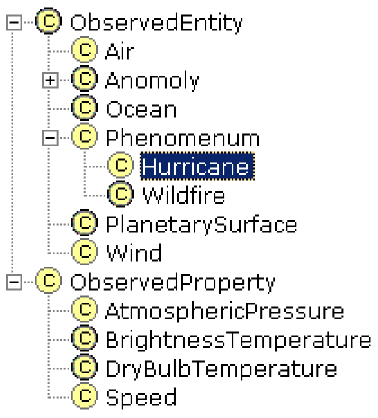


Figure 5.6: Concepts from the *swap-theme* ontology for representing wind speed and air pressure and hurricanes

Table 5.1: Prior Probabilities and Conditional Probability Tables (CPT) for detecting hurricanes

Variable	State	Prior Probability	
is_hurricane	*True*	0.05	
	False	0.95	
Variable	State	Conditional Probability	
wind_speed_var	**is_hurricane_var**	*True*	*False*
	gt_0_lt_50	0.02	0.98
	gt_49_lt_118	0.65	0.35
	gt_117	0.98	0.02
Variable	State	Conditional Probability	
air_pressure_var	**is_hurricane_var**	*True*	*False*
	lt_91d7	0.7	0.3
	gt_97d99	0.15	0.85
	gt_91d69_lt_98	0.6	0.4

The directed arcs from *is_hurricane_var* to *wind_speed_var* and *air_pressure_var* capture the influence of hurricanes on air pressure and wind speed, more specifically that hurricanes cause a lower air pressure and a higher wind speed. The states of the three variables together with fictitious values for prior and conditional probabilities are shown in table 5.1.

The BN described above can now be represented using the *swap-uncertainty* ontology by defining an instance of the *BayesianNetwork* class, *bn_detect_hurricane*, shown in figure 5.10. The three variables, *air_pressure_var*, *is_hurricane_var* and *wind_speed_var* are specified as the variables of the BN. The discrete numeric range states (*hasState*), parent nodes (*isInfluencedBy*) and thematic properties are specified for each variable. For the two observation variables, *air_pressure_var* and *wind_speed_var*, the *observesEntity*, *observesProperty* and *hasValueProperty* properties identify the observation instances to

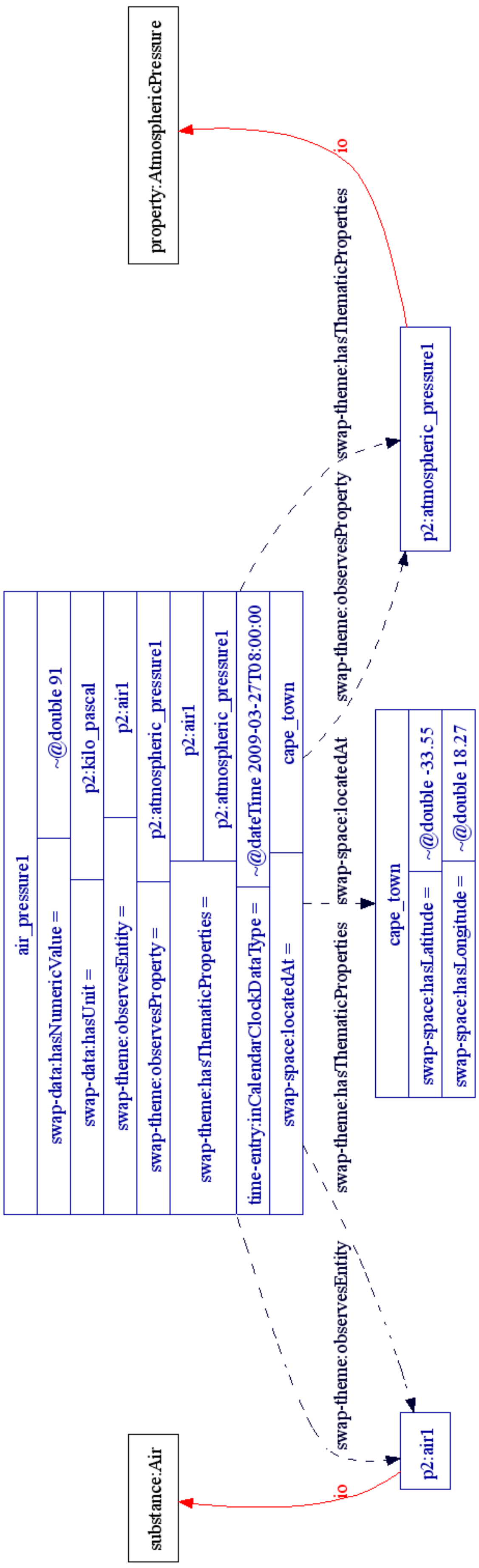

air_pressure1
swap-data:hasNumericValue = ~@double 91
swap-data:hasUnit = p2:kilo_pascal
swap-theme:observesEntity = p2:air1
swap-theme:observesProperty = p2:atmospheric_pressure1
swap-theme:hasThematicProperties = p2:air1
p2:atmospheric_pressure1
time-entry:inCalendarClockDataType = ~@dateTime 2009-03-27T08:00:00
swap-space:locatedAt = cape_town
cape_town
swap-space:hasLatitude = ~@double -33.55
swap-space:hasLongitude = ~@double 18.27
swap-theme:hasThematicProperties
swap-theme:observesProperty
swap-space:locatedAt
swap-theme:hasThematicProperties
swap-theme:observesEntity
p2:atmospheric_pressure1
p2:air1
io
io
property:AtmosphericPressure
substance:Air

Figure 5.7: An ontological representation of an air pressure measurement

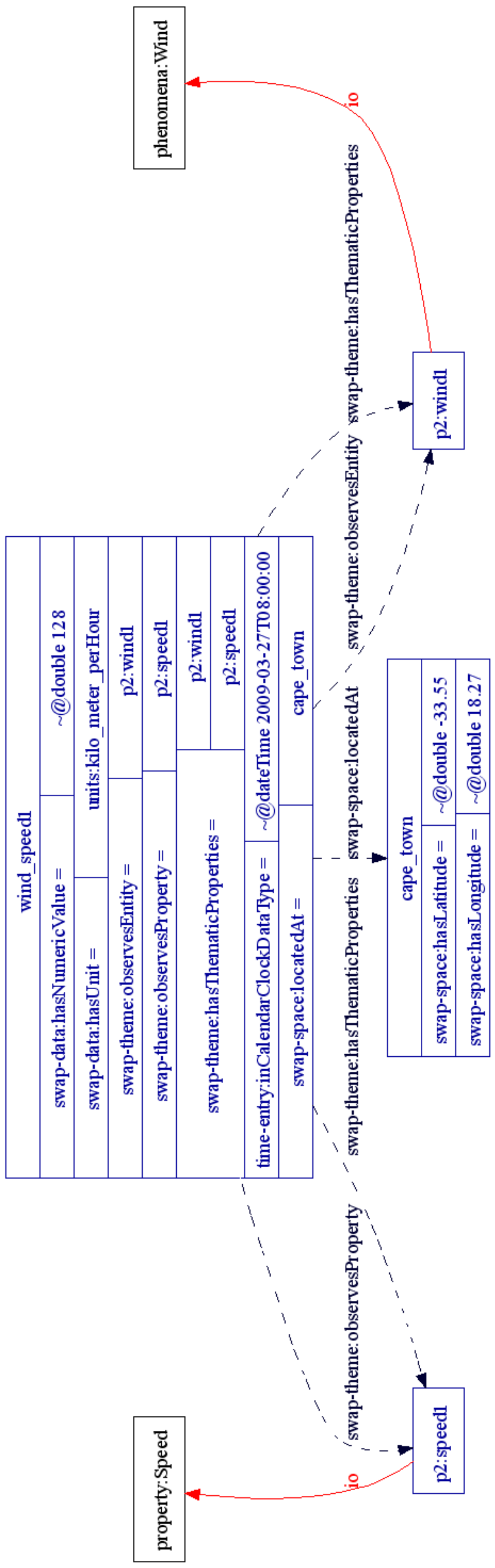

phenomena:Wind
property:Speed
wind_speed1
swap-data:hasNumericValue = ~@double 128
swap-data:hasUnit = units:kilo_meter_perHour
swap-theme:observesEntity = p2:wind1
swap-theme:observesProperty = p2:speed1
swap-theme:hasThematicProperties = p2:wind1 p2:speed1
time-entry:inCalendarClockDataType = ~@dateTime 2009-03-27T08:00:00
swap-space:locatedAt = cape_town
cape_town
swap-space:hasLatitude = ~@double -33.55
swap-space:hasLongitude = ~@double 18.27
p2:wind1
p2:speed1
swap-theme:hasThematicProperties
swap-theme:observesEntity
swap-space:locatedAt
swap-theme:hasThematicProperties
swap-theme:observesProperty
io
io

Figure 5.8: An ontological representation of a wind speed measurement

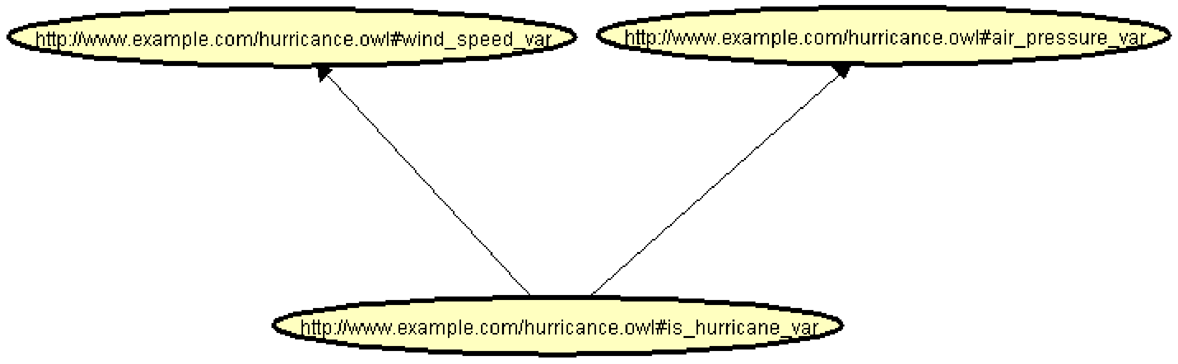

http://www.example.com/hurricance.owl#wind_speed_var
http://www.example.com/hurricance.owl#air_pressure_var
http://www.example.com/hurricance.owl#is_hurricane_var

Figure 5.9: A Bayesian Network to determine the occurrence of an hurricane from air pressure and wind speed observations

which these variables apply, i.e. air pressure (figure 5.8) and wind speed (figure 5.7) observations. For the inferred variable, *is_hurricane_var*, the *hasClass* property references the URI of the *Hurricane* class. When an hurricane is detected the location and time from the relevant observation instances are used to create an instance of the *Hurricane* class. The states for *air_pressure_var* and *wind_speed_var* are numeric intervals with upper and lower limits and are specified using the *NumericInterval* data type defined in the *swap-data* ontology (figure 5.11).

The probability statements from table 5.1 must also be specified for each variable. If a variable is not influenced by another variable, the conditional probability table consists of prior probability statements, one for each of the states of the variable. The *is_hurricane_var* variable has two prior probability statements and are represented as instances of *ProbObj* (see figure 5.5). A probability statement (*ProbObj*) represents the probability value of a given variable assuming a certain state. A prior probability instance representing a prior probability of 0.95 that *is_hurricane_var* assumes state *False* is shown in figure 5.12. Another instance of *PriorProb* is created to define the prior probability of *is_hurricane_var* being *True*.

For *wind_speed_var* three conditional probability statements, one for each of the probabilities in table 5.1, must be constructed. The first entry, 0.02, specifies that: if there is an hurricane there is a 2% probability that the wind speed is between 0 and 50 km per hour. The conditional probability statement shown in figure 5.13 represents this. A *Condition* instance, *cond_is_hurricane_var_True*, representing *is_hurricane_var* taking on state *True*, is first specified. An instance of *CondProb* is then created. It specifies the *wind_speed_var* as the variable (*hasVariable*), the previously defined *cond_is_hurricane_var_True* condition as a single condition (*hasCondition*), and *gt_0_lt_50* as the state (*hasState*) of the variable, and

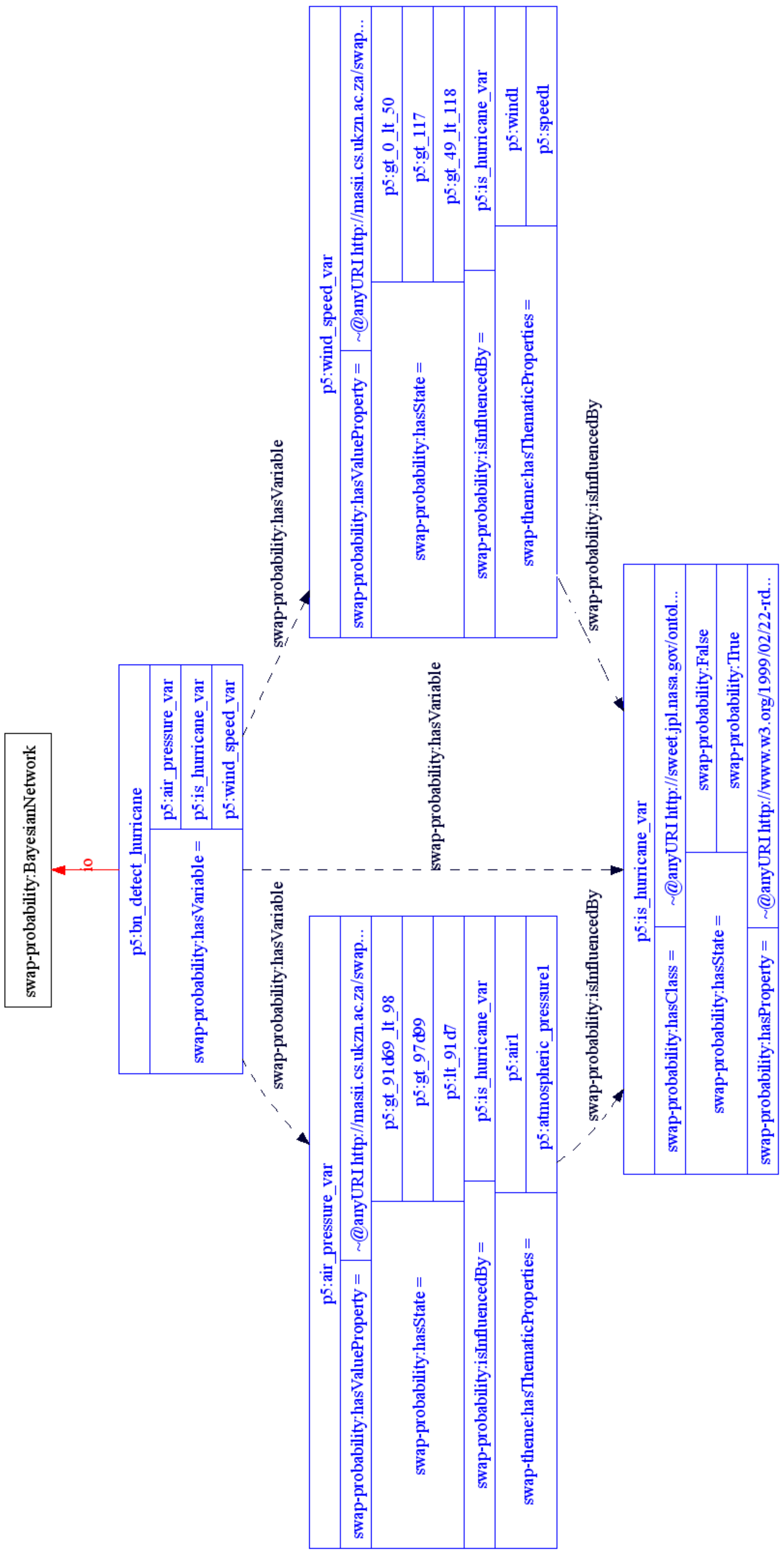

swap-probability:BayesianNetwork
io
p5:bn_detect_hurricane
swap-probability:hasVariable =
p5:air_pressure_var
p5:is_hurricane_var
p5:wind_speed_var
swap-probability:hasVariable
swap-probability:hasVariable
swap-probability:hasVariable
p5:wind_speed_var
swap-probability:hasValueProperty =
~@anyURI http://masii.cs.ukzn.ac.za/swap...
swap-probability:hasState =
p5:gt_0_lt_50
p5:gt_117
p5:gt_49_lt_118
swap-probability:isInfluencedBy =
p5:is_hurricane_var
swap-theme:hasThematicProperties =
p5:wind1
p5:speed1
p5:air_pressure_var
swap-probability:hasValueProperty =
~@anyURI http://masii.cs.ukzn.ac.za/swap...
swap-probability:hasState =
p5:gt_91d69_lt_98
p5:gt_97d99
p5:lt_91d7
swap-probability:isInfluencedBy =
p5:is_hurricane_var
swap-theme:hasThematicProperties =
p5:air1
p5:atmospheric_pressure1
swap-probability:isInfluencedBy
swap-probability:isInfluencedBy
p5:is_hurricane_var
swap-probability:hasClass =
~@anyURI http://sweet.jpl.nasa.gov/ontol...
swap-probability:hasState =
swap-probability:False
swap-probability:True
swap-probability:hasProperty =
~@anyURI http://www.w3.org/1999/02/22-rd...

Figure 5.10: An ontological representation of a Bayesian Network for detecting hurricanes

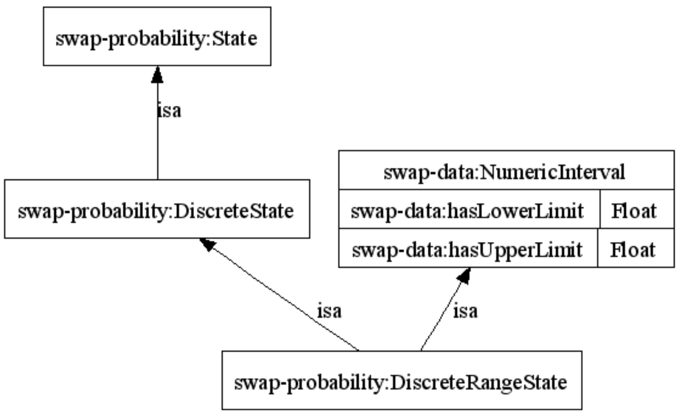

swap-probability:State
isa
swap-probability:DiscreteState
swap-data:NumericInterval
swap-data:hasLowerLimit
Float
swap-data:hasUpperLimit
Float
isa
isa
swap-probability:DiscreteRangeState

Figure 5.11: Representing discrete range states

```
<swap-uncertainty:PriorProb rdf:about="#pr_prob_is_hurricane_var_False">
  <swap-uncertainty:hasProbValue="0.95">
  <swap-uncertainty:hasState rdf:resource="&swap-uncertainty;False"/>
  <swap-uncertainty:hasVariable rdf:resource="#is_hurricane_var"/>
</swap-uncertainty:PriorProb>
```

Figure 5.12: Example of a prior probability statement

0.02 as the probability value (*hasProbValue*). Similarly, appropriate *Condition* and *CondProb* instances are created for the remaining five conditional probabilities for the *wind_speed_var* and the six conditional probabilities for *air_pressure_var* listed in table 5.1.

As illustrated in the example above the *swap-uncertainty* ontology can be used to specify the discrete random variables, the influence relations and the probability tables of a Bayesian Network (BN). The uncertainty reasoner is then used to perform inferencing on the BN, and to add the results or posterior probability statements to the local knowledge base.

```
<swap-uncertainty:Condition rdf:about="#cond_is_hurricane_var_True">
  <swap-uncertainty:hasVariable rdf:resource="#is_hurricane_var"/>
  <swap-uncertainty:hasState rdf:resource="&swap-uncertainty;True"/>
</swap-uncertainty:Condition>

<swap-uncertainty:CondProb rdf:about="#cd_prob_wind_speed_var_gt_0_lt_50_0">
  <swap-uncertainty:hasVariable rdf:resource="#wind_speed_var"/>
  <swap-uncertainty:hasState rdf:resource="#gt_0_lt_50"/>
  <swap-uncertainty:hasCondition rdf:resource="#cond_is_hurricane_var_True"/>
  <swap-uncertainty:hasProbValue="0.02">
</swap-uncertainty:CondProb>
```

Figure 5.13: Example of a conditional probability statement

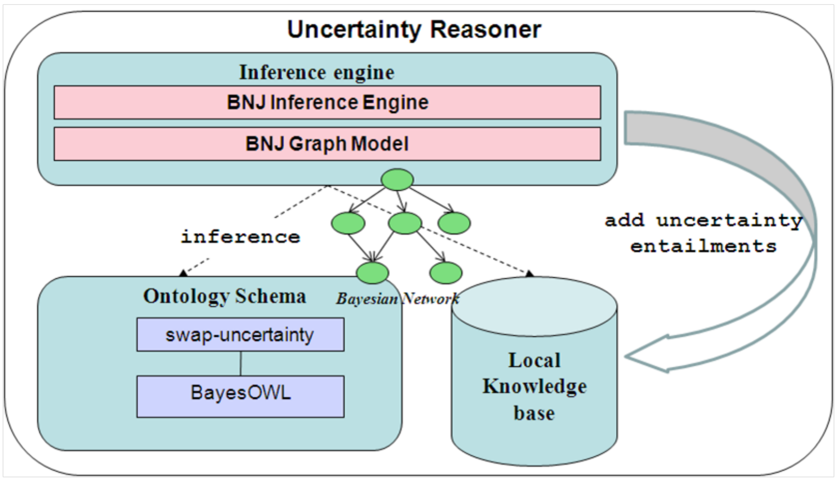


Figure 5.14: SWAP probability reasoner

5.2.3 The uncertainty reasoner

SWAP uses the BNJ toolkit for internal representation and inferencing. Bayesian Network tools in Java (BNJ) [2] is an open source Java toolkit for developing applications that use Bayesian Networks. It provides a visual Bayesian Network editor and viewer, a graph representation model for representing and manipulating a BN, a number of inference engines, as well as learning algorithms for constructing a Bayesian Network from data.

The schema and the components of the uncertainty reasoner are shown in figure 5.14.

Each BN (instances of *BayesianNetwork*) uses the states of observed variables (observation instances) to make inferences about whether a phenomena has occurred (inferred variables). If a phenomena has occurred then an instance of the corresponding phenomena, which contains the corresponding location and time of the observations, is created. A schema ontology containing the BN and observation instances from the local knowledge base are provided to the inference engine. The BN is first extracted from the schema ontology and used to create a BNJ graph model. The URIs of the variables and their states are used as the variable and state names in the BNJ graph model to ease the mapping of variables and states between the ontology and the graph model.

The following steps summarise the operation of the uncertainty reasoner:

- All observation variables are identified and added to a variable list. An instance list is then created for each variable in the variable list. The instance list contains all instances that match the *observesProperty* and *observesEntity* of the variable. If any general spatial and temporal constraints are specified for this BN then only those instances that satisfy these constraints are added to the instance list.

- Observation lists that comprise of observation instances for all variables that are made over the same time and location are then created. For each observation instance for a variable, iterate through all other variables and find observation instances for those variables that have matching location and time.

- For each observation list, determine and set the state of each observation variable that has a corresponding observation in the list using the *hasValueProperty* property from the variable to extract the observation value and corresponding state of the variable. The state of each observation variable is supplied as evidence values to these variables in the BNJ graph

- The BNJ inference engine is then applied to the graph.

- Posterior probabilities are extracted for the inferred variables and corresponding posterior probability statements are added to the local knowledge base, or could be stored separately in a temporary knowledge base.

- Using a predefined probability threshold, e.g. > 0.5, these statements are used to create new instances of the phenomena class to which these variables refer.

Consider the example BN to determine the occurrence of hurricanes described above. The air pressure (*air_pressure1*) and wind speed (*wind_speed1*) instances shown in figures 5.7 and 5.8) are used to populate the states of the observation variables, *air_pressure_var* and *wind_speed_var* to *lt_91d7* and *gt_117* respectively. The BNJ inference engine calculates a posterior probability of 0.8575 that *is_hurricane_var* is *True*. The result is used to generate and add the posterior probability statement shown in figure 5.15 to the local knowledge base. Since there is a 85.75% probability (above the predefined 50% classification threshold) that there is a hurricane the system creates an instance of *Hurricane* as shown in figure 5.16.

```
<swap-uncertainty:PostProb rdf:about=
              "#pp__is_hurricane_var_2009-03-27T08:00:00cape_town">
  <swap-uncertainty:hasProbValue rdf:datatype="&xsd;double">
              0.8575</swap-uncertainty:hasProbValue>
  <swap-uncertainty:hasState rdf:resource="&swap-uncertainty;True"/>
  <swap-uncertainty:hasVariable rdf:resource=
              "http://www.example.com/hurricane.owl#is_hurricane_var"/>
  <swap-uncertainty:inferredFromObservation rdf:resource="#air_pressure1"/>
  <swap-uncertainty:inferredFromObservation rdf:resource="#wind_speed1"/>
</swap-uncertainty:PostProb>
```

Figure 5.15: Example of a posterior probability statement

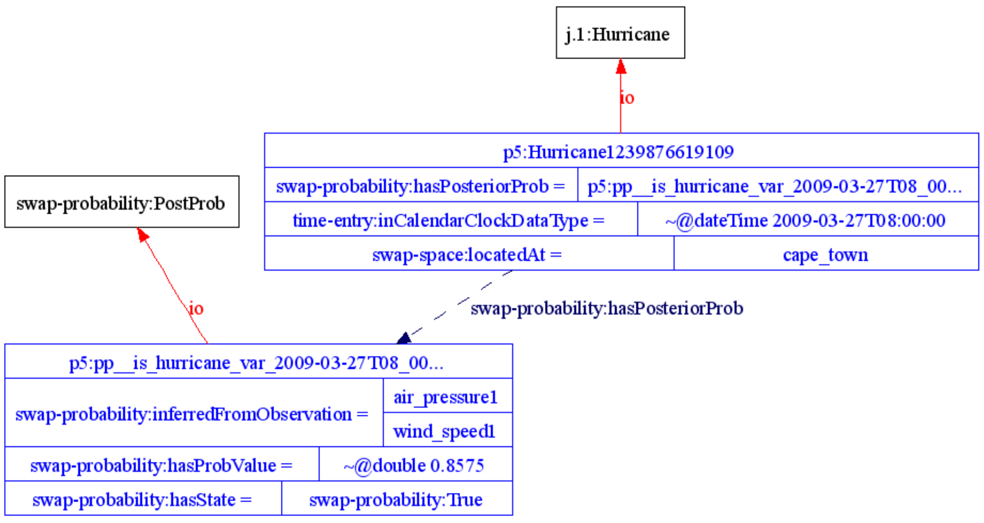

j.1:Hurricane
io
p5:Hurricane1239876619109
swap-probability:hasPosteriorProb =
p5:pp__is_hurricane_var_2009-03-27T08_00...
time-entry:inCalendarClockDataType =
~@dateTime 2009-03-27T08:00:00
swap-space:locatedAt =
cape_town
swap-probability:PostProb
io
swap-probability:hasPosteriorProb
p5:pp__is_hurricane_var_2009-03-27T08_00...
swap-probability:inferredFromObservation =
air_pressure1
wind_speed1
swap-probability:hasProbValue =
~@double 0.8575
swap-probability:hasState =
swap-probability:True

Figure 5.16: A *Hurricane* instance inferred using the SWAP probability reasoner

5.3 DISCUSSION AND SUMMARY

Representing and managing uncertain knowledge is an important aspect of the Sensor Web [25]. This chapter describes a practical approach to capture and use uncertain knowledge on the Sensor Web.

The *swap-uncertainty* ontology, described in section 5.2.1, allows for capturing and storing a complete Bayesian Network (BN) in an ontology. The variables of the BN correspond to either measurement observations (observable event variables) or phenomena (inferred event variables). These phenomena can be inferred by analysing measurements of the symptoms of the phenomena. The ontology allows for specifying and capturing multiple Bayesian Networks. BN variables are associated with observations and phenomena described in the *swap-theme* ontology. This is illustrated in section 5.2.2 using an example to detect hurricane phenomena. The uncertainty reasoner, described in section 5.2.3, provides functionality for: retrieving a BN from the knowledge base; dynamically populating the BN with available observations; determining the probability of inferred events or phenomena; and creating and adding inferred phenomena instances to the knowledge base.

Some of the benefits of the approach are:

Capturing, storing and sharing uncertain knowledge Scientists can capture and store uncertain knowledge or theories within one or more Bayesian Networks. Uncertainty resulting from weak theories, sensor inaccuracies as well as location specific anomalies can be incorporated in the BN. The thematic aspects of events are described using concepts from the SWAP ontologies and facilitate consistent Modeling of observed and inferred event variables. Theories reflecting causal relations between events and the degree of these relations are captured in the Conditional Probability Table (CPT). A BN can be published, shared and interpreted by agents as well as human users.

The CPTs of the event variables as well as the influence relations are stored in the knowledge base and can be easily updated in line with the continuous evolution of the beliefs and understanding of scientists. Multiple BNs can be stored for the same phenomena that reflect different theories held by different scientists. BNs can be easily changed to accommodate new observations as new sensors come online. Once the observable event variables are reconfigured, the reasoner is able to dynamically feed the new observations to the BN without additional changes. Similarly a BN can be accessed by other users and reconfigured by changing the CPT and structure accordingly. This can facilitate experimentation to better reflect local or alternative conditions.

Dynamic application of knowledge Bayesian Networks are dynamically incorporated into SWAP. As observations become available, they can be dynamically processed by the appropriate Bayesian Networks to detect phenomena events. Inferred phenomena instances including the details of the BN and the observation instances which resulted in its creation are recorded and disseminated in the system. When an inferred phenomena instance is encountered, the theory (BN) and the specific observations that resulted in its creation can be determined. The details of the creator or author of a BN may also be captured in the ontology. This provide a complete audit trail of the origin of phenomena instances, including the details of the domain expert who constructed the BN. This can assist in determining the quality and reliability of phenomena events and in taking decisions based on these events.

All BNs are specified uniformly and variables representing observations and phenomena are described using concepts from the *swap-theme* ontologies. The thematic reasoner can be used to detect related observations. This includes observations which measure the same or a similar observed entities and properties over different locations or using different sensors. A BN can be used to detect the same phenomena at other locations. If observations from the required sensors are not available for the location, related observations from other sensors can be identified and used. However, BNs may be sensor or location specific and not easily reusable. These related BNs may still be of benefit to scientists who can identify and correct the location and sensor differences. This facilitates the sharing of knowledge and experience between scientists.

Uncertain knowledge is not just a feature of the Sensor Web but also features more broadly within the Semantic Web [177]. As OWL does not provide support for representing uncertainty, approaches such as OntoBayes [207, 208] and BayesOWL [57, 58] propose extensions to OWL for incorporating Bayesian Networks. Our approach extends BayesOWL to incorporate observable measurement data and also places the variables and results of a BN within a comprehensive ontology infrastructure. This not only allows for evidence values to be dynamically populated, but also for posterior probabilities to be dynamically analysed, integrated and shared within the system.

This chapter described how Bayesian probability and Bayesian Networks can be used to represent and reason about uncertainty on the Sensor Web. In the next chapter the implementation of two prototype applications are described to illustrate the practical operation and use of the Sensor Web Agent Platform.

Chapter 6

IMPLEMENTING SWAP APPLICATIONS

This chapter describes how Sensor Web applications are developed and deployed on SWAP. The SWAP abstract architecture provides abstractions to guide the design of SWAP applications, while the ontological infrastructure provides a framework for defining agent interfaces and agent interactions in terms of message types and conversation protocols. Individual SWAP agents are implemented and executed using the MASII platform described in chapter 3.

Two application case studies are described. The first application for wildfire detection incorporates a single Sensor and Tool Agent. It is used as a running example to illustrate the operation of the different SWAP agents and various features of the framework such as the incorporation of uncertainty and the deployment of end user applications. The second application for monitoring informal settlements is a more complex application, which incorporates multiple data sets (Sensor Agents) and multiple processing steps (Tool Agents) as well as a supervised machine learning algorithm for feature classification.

6.1 CASE STUDY 1: WILDFIRE DETECTION

6.1.1 Application overview

The Advanced Fire Information System (AFIS) [63, 71] is a near real-time satellite-based fire monitoring system in Africa. Satellite image data measured by the Severi sensor on the MSG satellite provide brightness temperature measurements over specific spatial locations over the earth. These measurements are an indication of the temperature of the earth's surface over these locations. Abnormally high temperature values are used as a basis for detecting wildfires. The design of the wildfire detection application is based on the approach taken in the AFIS system.

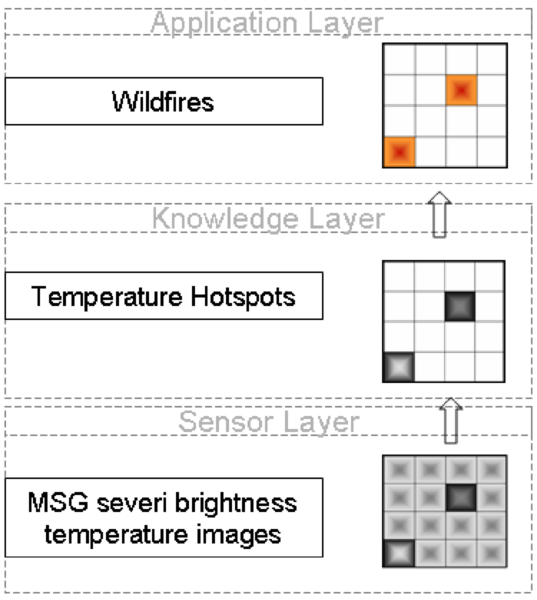


Figure 6.1: Extracting wildfires from satellite data

Each of the SWAP architectural layers provides different levels of abstractions to extract meaningful information from sensor data. Data transformations are required at each layer to detect wildfires. These transformations are shown in figure 6.1. Brightness temperature images at the Sensor Layer are used to extract thermal hotspots at the Knowledge Layer. These thermal hotspots are used to detect wildfires at the Application Layer.

A part of the wildfire ontology is shown in figure 6.2. The following classes are used to represent the concepts required for wildfire detection shown in (figure 6.1).

- *MSGThermalValue* instances represent individual values from the MSG brightness temperature (BT) images. It has an instance of *BrightnessTemperature* (referenced from the SWEET ontology) as its observed property and an instance of *PlanetarySurface* as its *ObservedEntity*.
- *MSGThermalHotspot* instances are *MSGThermalValue* instances that have abnormally high BT values. Hotspot instances that are also instances of *MSGThermalValue* are defined as *MSGThermalHotspots*.
- *Wildfire* instances are those *MSGThermalHotspot* instances that are deemed to be wildfires.

Agents are required at each layer to provide the required data and data transformations (see figure

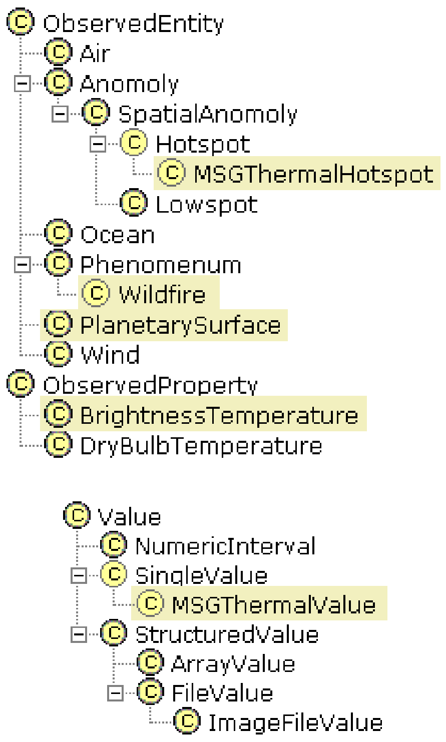


Figure 6.2: Thematic concepts for wildfire detection

6.3). At the Sensor Layer, the MSG Sensor agent offers brightness temperature data every fifteen minutes. The data is offered as georeferenced [1] GeoTiff images. At the Knowledge Layer three agents are deployed, a Contextual Algorithm (CA) Tool Agent, a Fire Spread (FS) Modeling Agent and a Hotspot Detector (HD) Workflow Agent. The CA Tool Agent takes as input georeferenced raster data, such as a GeoTiff file, detects pixels with spatial anomalies, and outputs the pixel values with their real world coordinates. In this instance the output is pixels with abnormally high values relative to neighbouring pixels. The HD Workflow Agent performs a composite process or workflow to detect temperature hotspots. It retrieves brightness temperature data from the MSG Sensor Agent and passes this data to the CA Tool Agent, which outputs abnormally high values in the data, i.e. the temperature hotspots. The FS Modeling Agent takes in a current wildfire and predicts the probability of the spread of the wildfire towards a specified location. In the Application Layer two agents are deployed, a Fire Detection (FD) Application Agent and a Fire Detection (FD) User Agent. The FD Application Agent receives and stores the temperature hotspots from the HD Workflow Agent and uses these hotspots to indicate possible wildfires. It also retrieves wildfire spread predictions for current wildfires from the FS Modeling Agent. The FD

[1] a pixel in the image can be mapped to a real world location

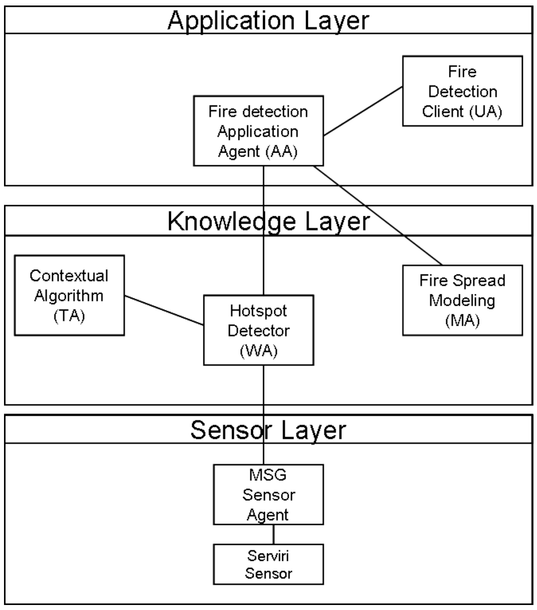

Application Layer
Fire Detection Client (UA)
Fire detection Application Agent (AA)
Knowledge Layer
Contextual Algorithm (TA)
Hotspot Detector (WA)
Fire Spread Modeling (MA)
Sensor Layer
MSG Sensor Agent
Serviri Sensor

Figure 6.3: Architecture of a wildfire detection application

User Agent registers alerts with the FD Application Agents, by providing a filter of features or areas of interest. The FD Application Agent responds with wildfire alerts and spread predictions whenever wildfires occur within these areas.

A description of how each of these concepts are used within the different agents is given later in section 6.2. The next section describes the use of a Bayesian Network to capture and reason about the causal relations between wildfires and MSG brightness temperature measurements.

6.1.2 Representing and reasoning about uncertainty for wildfire detection

Each pixel value in the MSG satellite image data represents Brightness Temperature (BT) values captured by the MSG Severi sensor. An instance of an *MSGThermalValue* is shown in figure 6.4. It has six properties:

- *hasNumericValue* represents the actual BT value at this location
- *hasSpatialVariance* represents the deviation of the pixel's value with that of its neighbours as a percentage calculated by $(val - (mean + std\_dev))/(mean\_std\_dev))$, where val is the pixel value, $mean$ is the mean value of the neighbouring pixels and std_dev is the standard deviation of the value from its neighbours

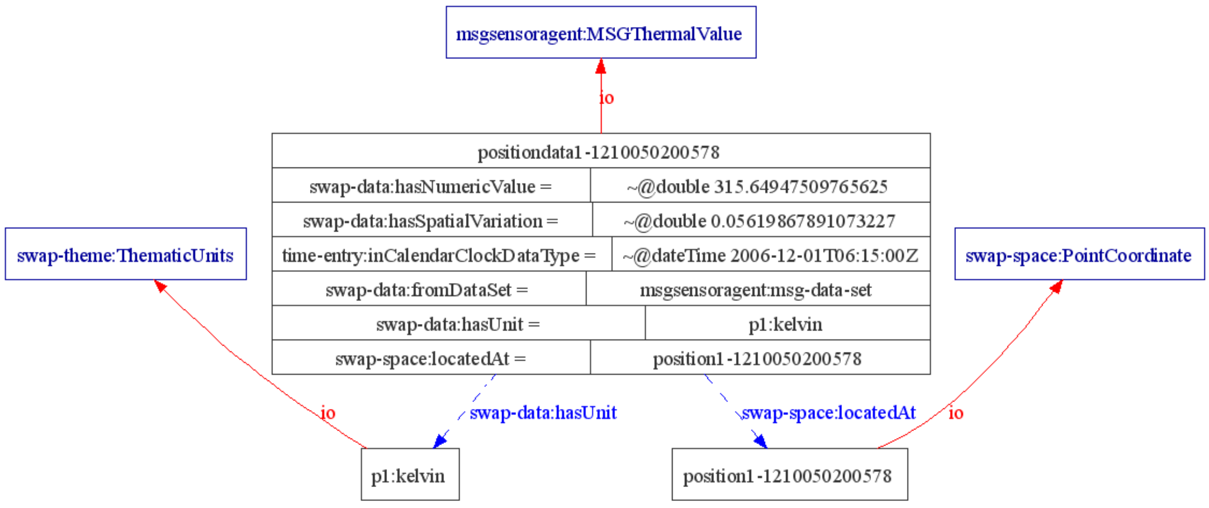

msgsensoragent:MSGThermalValue
io
positiondata1-1210050200578
swap-data:hasNumericValue =
~@double 315.64947509765625
swap-data:hasSpatialVariation =
~@double 0.05619867891073227
time-entry:inCalendarClockDataType =
~@dateTime 2006-12-01T06:15:00Z
swap-data:fromDataSet =
msgsensoragent:msg-data-set
swap-data:hasUnit =
p1:kelvin
swap-space:locatedAt =
position1-1210050200578
swap-theme:ThematicUnits
swap-space:PointCoordinate
io
swap-data:hasUnit
swap-space:locatedAt
io
p1:kelvin
position1-1210050200578

Figure 6.4: Representing an MSG brightness temperature measurement

- *inCalendarClockDataType* represents the observation time
- *fromDataSet* describes the data set to which this measurement belongs, in this case the *msg-data-set*
- *hasUnit* represents the unit of measure
- *locatedAt* represents the spatial coordinates of the location

Two key conditions must be satisfied for a BT value to be classified as a thermal hotspot (*MSGThermalHotspot*):

- The BT value (*hasNumericValue*) must be over 315K.
- The BT value measured at this pixel must be significantly higher than values of neighbouring pixels (*hasSpatialVariance*).

The values of the *hasSpatialVariance* and *hasNumericValue* properties of an *MSGThermalValue* instance therefore determine whether the *MSGThermalValue* is also a *MSGThermalHotspot*. The occurrence of a thermal hotspot is an indication of the possibility of a wildfire occurring at that location. Therefore, some hotspots may be classified as wildfires.

A Bayesian Network (BN) (see section 5.2.1), shown in figure 6.5, was created to determine the probability of an occurrence of a wildfire. The BN consists of four variables, i.e. *MSG_thm_btval_var*,

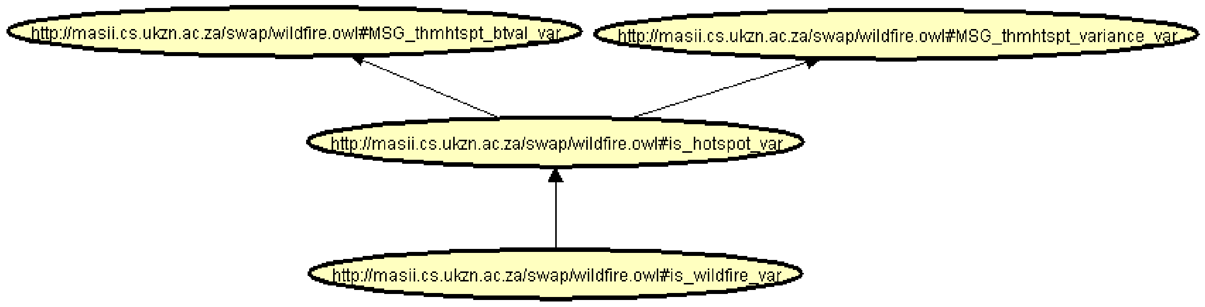

http://masii.cs.ukzn.ac.za/swap/wildfire.owl#MSG_thmhtspt_btval_var
http://masii.cs.ukzn.ac.za/swap/wildfire.owl#MSG_thmhtspt_variance_var
http://masii.cs.ukzn.ac.za/swap/wildfire.owl#is_hotspot_var
http://masii.cs.ukzn.ac.za/swap/wildfire.owl#is_wildfire_var

Figure 6.5: A Bayesian Network for wildfire detection

MSG_thm_variance_var, *is_hotspot_var* and *is_wildfire_var*. The first two represent the variance and the value of a MSG brightness temperature respectively. The latter two represent whether there is a hotspot and whether the hotspot is a wildfire. The directed arcs between variables represent cause and effect relations between events in the real world. Therefore a wildfire causes a thermal hotspot, a thermal hotspot in turn causes a high BT value and a high spatial variance of a *MSGThermalValue* relative to its neighbours.

Figure 6.6 shows the definition of the *msg_thm_btval_var* variable, which represents the BT value of an *MSGThermalValue*. Since the *hasNumericValue* property of an *MSGThermalValue* instance represents a BT value, *hasClass* is set to *MSGThermalValue* and *hasProperty* is set to *hasNumericValue*. By interogating the *hasClass* and *hasProperty* properties, BT values can be extracted from *MSGThermalValue* instances and used to determine the state of the variable. The variable has three states: less than 315, greater than 315 and less than 318, and greater than 318. The states are defined as discrete ranges. One of the states, *gt_315_lt_318*, is shown in figure 6.6. The *isInfluencedBy* property indicates that this variable is influenced by the *is_hotspot_var* variable.

Figure 6.7 shows the definition of the *msg_thm_variance_var* variable, which represents the spatial variance of this BT value relative to its neighbours. The *hasSpatialVariance* property of an *MSGThermalValue* instance represents the variance. It has three states, i.e. less than 1, greater than 1 and less than 5, and greater than 5. This variable is also influenced by the *is_hotspot_var* variable.

Figure 6.8 shows the *is_hotspot_var* variable, which indicates whether an *MSGThermalValue* is a hotspot. The *rdf:type* property, a native OWL construct that specifies the class type, is used to determine whether an instance is a *MSGThermalHotspot*. It is influenced by the *is_wildfire_var* variable. Figure 6.9

```
<swap-probability:Variable rdf:about="#msg_thm_btval_var">
  <swap-probability:hasClass rdf:datatype="&xsd;anyURI">
      http://masii.cs.ukzn.ac.za/swap/wildfire.owl#MSGThermalValue
  </swap-probability:hasClass>
  <swap-probability:hasProperty rdf:datatype="&xsd;anyURI">
      http://masii.cs.ukzn.ac.za/swap/swap-data.owl#hasNumericValue
  </swap-probability:hasProperty>
  <swap-probability:hasState rdf:resource="#lt_315"/>
  <swap-probability:hasState rdf:resource="#gt_315_lt_318"/>
  <swap-probability:hasState rdf:resource="#gt_318"/>
  <swap-probability:isInfluencedBy rdf:resource="#is_hotspot_var"/>
</swap-probability:Variable>

<swap-probability:DiscreteRangeState rdf:about="#gt_315_lt_318">
  <swap-data:hasLowerLimit rdf:datatype="&xsd;double">
      315
  </swap-data:hasLowerLimit>
  <swap-data:hasUpperLimit rdf:datatype="&xsd;double">
      318
  </swap-data:hasUpperLimit>
</swap-probability:DiscreteRangeState>
```

Figure 6.6: Representing the MSG thermal BT value variable

```
<swap-probability:Variable rdf:about="#msg_thm_variance_var">
  <swap-probability:hasClass rdf:datatype="&xsd;string">
      http://masii.cs.ukzn.ac.za/swap/wildfire.owl#MSGThermalValue
  </swap-probability:hasClass>
  <swap-probability:hasProperty rdf:datatype="&xsd;string">
      http://masii.cs.ukzn.ac.za/swap/swap-data.owl#hasSpatialVariation
  </swap-probability:hasProperty>
  <swap-probability:hasState rdf:resource="#gt_1_lt_5"/>
  <swap-probability:hasState rdf:resource="#gt_5"/>
  <swap-probability:hasState rdf:resource="#lt_1"/>
  <swap-probability:isInfluencedBy rdf:resource="#is_hotspot_var"/>
</swap-probability:Variable>
```

Figure 6.7: Representing the MSG thermal variance variable

```
<swap-probability:Variable rdf:about="#is_hotspot_var">
  <swap-probability:hasClass rdf:datatype="&xsd;anyURI">
      http://masii.cs.ukzn.ac.za/swap/wildfire.owl#MSGThermalValue
  </swap-probability:hasClass>
  <swap-probability:hasProperty rdf:datatype="&xsd;anyURI">
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
  </swap-probability:hasProperty>
  <swap-probability:hasState rdf:resource="&swap-probability;False"/>
  <swap-probability:hasState rdf:resource="&swap-probability;True"/>
  <swap-probability:isInfluencedBy rdf:resource="#is_wildfire_var"/>
</swap-probability:Variable>
```

Figure 6.8: Representing the *is_hotspot* variable

```
<swap-probability:Variable rdf:about="#is_wildfire_var">
  <swap-probability:hasClass rdf:datatype="&xsd;anyURI">
      http://masii.cs.ukzn.ac.za/swap/eo-domain.owl#Wildfire
  </swap-probability:hasClass>
  <swap-probability:hasProperty rdf:datatype="&xsd;anyURI">
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
  </swap-probability:hasProperty>
  <swap-probability:hasState rdf:resource="&swap-probability;False"/>
  <swap-probability:hasState rdf:resource="&swap-probability;True"/>
</swap-probability:Variable>
```

Figure 6.9: Representing the *is_wildfire* variable

shows the *is_wildfire_var* variable, which indicates whether a *MSGThermalHotspot* is a *Wildfire*. It is not influenced by another variable.

The prior and conditional probabilities for the four variables are shown in table 6.1. Since the *is_wildfire_var* is not influenced by other variables, prior probabilities for each of its two states are specified (see figure 6.10).

The *is_hotspot_var* variable requires four conditional probability statements, one for each of the conditional probabilities in table 6.1. The first value represents the probability that *is_hotspot_var* is

```
<swap-probability:PriorProb rdf:about="#pr_prob_is_wildfire_var_True">
  <swap-probability:hasProbValue rdf:datatype="&xsd;double">
      0.05
  </swap-probability:hasProbValue>
  <swap-probability:hasState rdf:resource="&swap-probability;True"/>
  <swap-probability:hasVariable rdf:resource="#is_wildfire_var"/>
</swap-probability:PriorProb>

  <swap-probability:PriorProb rdf:about="#pr_prob_is_wildfire_var_False">
  <swap-probability:hasProbValue rdf:datatype="&xsd;double">
      0.95
  </swap-probability:hasProbValue>
  <swap-probability:hasState rdf:resource="&swap-probability;False"/>
  <swap-probability:hasVariable rdf:resource="#is_wildfire_var"/>
</swap-probability:PriorProb>
```

Figure 6.10: Prior Probability statements for the *is_wildfire_var* variable

Table 6.1: Prior Probabilities and Conditional Probability Table (CPT) for classifying wildfires

Variable	State	Prior Probability
is_wildfire_var	*True*	0.05
	False	0.95

Variable	State	Conditional Probability	
is_hotspot_var	**is_wildfire_var**	*True*	*False*
	True	0.9990	0.0200
	False	0.0010	0.9800

Variable	State	Conditional Probability	
msg_thm_variance_var	**is_hotspot_var**	*True*	*False*
	lt_1	0.0050	0.8000
	gt_1_lt_5	0.2450	0.1999
	gt_5	0.7500	0.0001

Variable	State	Conditional Probability	
msg_thm_btval_var	**is_hotspot_var**	*True*	*False*
	lt_315	0.0000	0.6700
	gt_315_lt_318	0.2000	0.3200
	gt_318	0.8000	0.0100

True given that *is_wildfire_var* is *True*. As shown in figure 6.11, a condition instance, for the condition *is_wildfire_var* is *True*, is first defined. Then two *CondProb* instances are created. The first is for the probability that *is_hotspot_var* is *True* when *is_wildfire_var* is *True*, and the second is for the probability that it is *False* when *is_wildfire_var* is *True*. Similarly, a *Condition* instance for *is_wildfire_var* taking on the *False* state, and two other *CondProb* instances are defined.

Similarly, the *msg_thm_variance_var* variable requires six conditional probability statements (see table 6.1). The first probability value represents the probability that *msg_thm_variance_var* is *lt_315* given that *is_hotspot_var* is *True*. As shown in figure 6.12 a condition instance representing the condition that *is_hotspot_var* is *True* is first defined. Then three *CondProb* instances are created, representing the three conditional probabilities for each of its three states when *is_hotspot_var* is *True*. Similarly, another *Condition* instance for *is_hotspot_var* taking on state *False* is defined and the other three *CondProb* instances are defined.

The six conditional probability statements for the *msg_thm_btval_var* are defined in a similar manner.

The uncertainty reasoner (see section 5.2.3) can then be applied to MSG thermal values. It generates posterior probability statements similar to the one shown in figure 6.13. When the posterior probability of a wildfire occurring is over a predefined threshold (e.g. over 0.5) then an instance of *Wildfire* is created.

```
<swap-probability:Condition rdf:about="#cond_is_wildfire_var_True">
  <swap-probability:hasState rdf:resource="&swap-probability;True"/>
  <swap-probability:hasVariable rdf:resource="#is_wildfire_var"/>
</swap-probability:Condition>

<swap-probability:CondProb rdf:about="#cd_prob_is_hotspot_var_True_0">
  <swap-probability:hasCondition rdf:resource="#cond_is_wildfire_var_True"/>
  <swap-probability:hasProbValue rdf:datatype="&xsd;double">
      0.999
  </swap-probability:hasProbValue>
  <swap-probability:hasState rdf:resource="&swap-probability;True"/>
  <swap-probability:hasVariable rdf:resource="#is_hotspot_var"/>
</swap-probability:CondProb>

  <swap-probability:CondProb rdf:about="#cd_prob_is_hotspot_var_False_0">
  <swap-probability:hasCondition rdf:resource="#cond_is_wildfire_var_True"/>
  <swap-probability:hasProbValue rdf:datatype="&xsd;double">
      0.001
  </swap-probability:hasProbValue>
  <swap-probability:hasState rdf:resource="&swap-probability;False"/>
  <swap-probability:hasVariable rdf:resource="#is_hotspot_var"/>
</swap-probability:CondProb>
```

Figure 6.11: A conditional probability statement for the *is_hotspot_var* variable

```
<swap-probability:Condition rdf:about="#cond_is_hotspot_var_True">
  <swap-probability:hasState rdf:resource="&swap-probability;True"/>
  <swap-probability:hasVariable rdf:resource="#is_hotspot_var"/>
</swap-probability:Condition>

<swap-probability:CondProb rdf:about=
      "#cd_prob_MSG_ThmHspt_Variance_Var_lt_1_0">
  <swap-probability:hasCondition rdf:resource="#cond_is_hotspot_var_True"/>
  <swap-probability:hasProbValue rdf:datatype="&xsd;double">
      0.005
  </swap-probability:hasProbValue>
  <swap-probability:hasState rdf:resource="#lt_1"/>
  <swap-probability:hasVariable rdf:resource="#MSG_thmhtspt_variance_var"/>
</swap-probability:CondProb>

<swap-probability:CondProb rdf:about=
      "#cd_prob_MSG_Thm_Htspt_val_var_gt_315_lt_318_0">
  <swap-probability:hasCondition rdf:resource="#cond_is_hotspot_var_True"/>
  <swap-probability:hasProbValue rdf:datatype="&xsd;double">
      0.2
  </swap-probability:hasProbValue>
  <swap-probability:hasState rdf:resource="#gt_315_lt_318"/>
  <swap-probability:hasVariable rdf:resource="#MSG_thmhtspt_btval_var"/>
</swap-probability:CondProb>

  <swap-probability:CondProb rdf:about=
      "#cd_prob_MSG_Thm_Htspt_val_var_gt_318_0">
  <swap-probability:hasCondition rdf:resource="#cond_is_hotspot_var_True"/>
  <swap-probability:hasProbValue rdf:datatype="&xsd;double">
      0.8
  </swap-probability:hasProbValue>
  <swap-probability:hasState rdf:resource="#gt_318"/>
  <swap-probability:hasVariable rdf:resource="#MSG_thmhtspt_btval_var"/>
</swap-probability:CondProb>
```

Figure 6.12: Conditional probability statements for the *msg_thm_variance_var* variable

```
<swap-uncertainty:PostProb rdf:about=
            "#pp_positiondata1-121005020000_is_wildfire_var">
  <swap-uncertainty:hasProbValue rdf:datatype="&xsd;double">
            0.7575</swap-uncertainty:hasProbValue>
  <swap-uncertainty:hasState rdf:resource="&swap-uncertainty;True"/>
  <swap-uncertainty:hasVariable rdf:resource=
            "http://masii.cs.ukzn.ac.za/swap/wildfire.owl#is_wildfire_var"/>
  <swap-uncertainty:inferredFromObservation rdf:resource="#positiondata1-121005020000"/>
</swap-uncertainty:PostProb>
```

Figure 6.13: A posterior probability statement for the *is_wildfire_var*

6.2 SWAP AGENT OPERATION AND IMPLEMENTATION

In this section the operation of the six different SWAP agent types is discussed. The external representation of each agent, i.e. the interaction protocol used to communicate with the agent, as well as the internal operation are described. An implementation of each agent within the context of the wildfire detection application is also described.

6.2.1 Sensor Agent

Sensor Agents, described in section 4.2.1, are responsible for serving semantically marked up sensor data in a sensor independent format.

6.2.1.1 External representation

Sensor Agents host the *Sensor Data* service and accept *QueryAction* request messages (representing a query for sensor data) and respond with *InformDataAction* messages containing data that matches the query. The role is described in table 6.2 and the $DataRequest$ protocol is described in table 6.2.

6.2.1.2 Internal operation

The Sensor Agent binds to a data source, e.g. a set of local image files, a database or an OGC Sensor Observation Service (SOS). A Java interface, the *DataAdapter* interface (figure 6.14), provides a standard interface to bind to and access different sensor data sources. It provides a generic way in which to access different data sets independently of the type or structure of the data source. The data source could be a spatial database, such as PostGIS or Oracle, an OGC Sensor Observation Service (SOS) or even

Table 6.2: The Sensor Data Provider schema

Role Schema:	*Sensor Data Provider*
Description	Hosts a sensor data repository or data set. Primarily answers queries for data from its data set. Registers and provides a $Service$ instance that describes its data set and interaction protocols.
Protocols and Activities	*DataRequest*, *CapabilityRequest*, *SWAPServiceRegistration*
Permissions	Update capabilities in registry
Responsibilities	• Register capabilities with Directory Agent • Answer query for capabilities • Answer query for data • Maintain its data set

Table 6.3: The *DataRequest* Protocol schema

Protocol	*DataRequest*
Initiator	Any agent e.g. workflow agent
Participants	Sensor Agent, initiating agent
Message Types	QueryAction, InformDataAction
Inputs	Data Query Criteria
Outputs	Sensor Data corresponding to query criteria
Description	An agent can request sensor data from the Sensor Agent based on criteria specified in a data query. The Sensor Agent responds with the data that satisfies the query criteria.

an image file store on a local hard disk drive. The interface provides two methods for data access, i.e. the *queryCoverageData* and the *queryFeatureData* methods to retrieve coverage and feature data respectively.

Three implementation classes of this interface are shown in figure 6.14. The first, the *BrightnesstemperatureDataAdapter*, provides access to an image store containing raw MSG brightness temperature images. The *WeatherDataAdapter* class provides access to a Weather SOS that serves weather data generated by the South African Weather Service [2], while the *HotspotDataAdapter* class provides data from a prototype AFIS Fire Detection SOS [3]. The *BrightnesstemperatureDataAdapter* provides coverage data while the other two adapters provide feature data. The input and output parameters of the query methods are standard GeoAPI data structures using the OXFramework implementation. The *queryCoverageData* method takes in a string representing the phenomena or observable, e.g. *msg-brightness-temperature*, the spatial extent in the form of an *IBoundingBox*, and the time as an *ITime* object. It returns a Java *File* object, which represents a GeoTiff file on the local file system. The *queryFeatureData* method takes as parameters the required observable as a *String*, a set of spatial coordinates as a Java *Set* of *Coordinate* objects and the time represented as an *ITime* object. It returns a Java *Collection* of features represented as *OXFFeature* objects. If the data adapter supports only one data set, such as the *BrightnesstemperatureDataAdapter* which only contains *msg-brightness-temperature* data, then the observable (*String*) parameter is not necessary. If the adapter supports more than one data set, such as the *WeatherDataAdapter* which supports dry-bulb temperature, wind speed and wind direction, then the observable parameter specifies the local name of the *observedProperty*. For example, if *saws-drybulb-temperature* is the local name of the requested *observedProperty*, then the *queryFeatureData* method of the *WeatherDataAdapter* returns dry-bulb temperature observations.

The Sensor Agent uses the data mapping API (see section 4.7.3) to convert between ontology representations of time, space and the corresponding GeoAPI data structures. Coverage data is returned as a georeferenced image file (in the GeoTiff format) and feature data is returned as a collection of OpenGIS features. The *DataAdapter* interface hides the implementation details of the data access methods. In this way the data source and implementation details to query and access data is hidden. Furthermore, the implementation class for the data adapter is specified as a runtime parameter for the agent and is only loaded and bound at runtime. In this way it is possible to change the sensor data source by specifying

[2]http://ict4eo.meraka.csir.co.za:8080/WeatherSOSSA/sos
[3]http://ict4eo.meraka.csir.co.za:8080/AFIS_SOS/sos

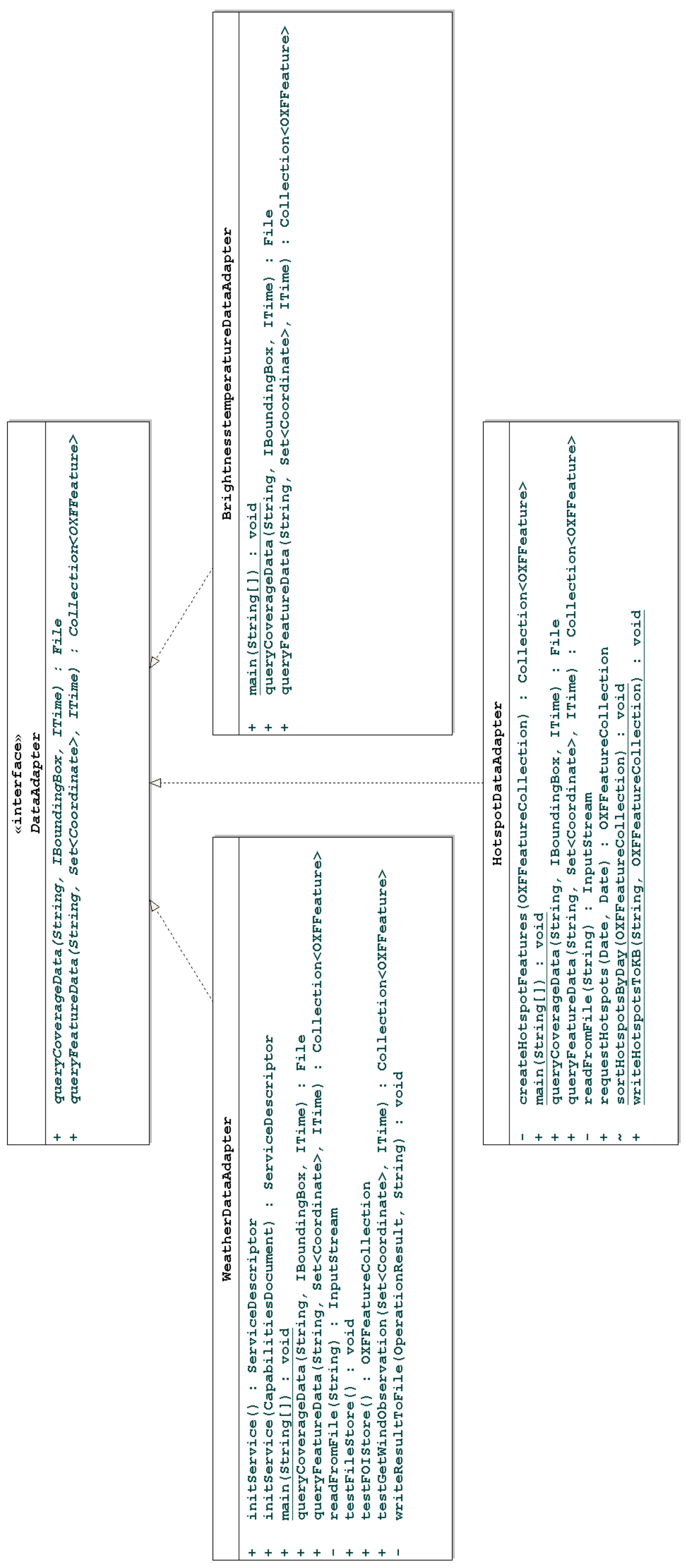

«interface»
DataAdapter
+ queryCoverageData(String, IBoundingBox, ITime) : File
+ queryFeatureData(String, Set<Coordinate>, ITime) : Collection<OXFFeature>
BrightnesstemperatureDataAdapter
+ main(String[]) : void
+ queryCoverageData(String, IBoundingBox, ITime) : File
+ queryFeatureData(String, Set<Coordinate>, ITime) : Collection<OXFFeature>
WeatherDataAdapter
+ initService() : ServiceDescriptor
+ initService(CapabilitiesDocument) : ServiceDescriptor
+ main(String[]) : void
+ queryCoverageData(String, IBoundingBox, ITime) : File
+ queryFeatureData(String, Set<Coordinate>, ITime) : Collection<OXFFeature>
- readFromFile(String) : InputStream
+ testFileStore() : void
+ testFOIStore() : OXFFeatureCollection
+ testGetWindObservation(Set<Coordinate>, ITime) : Collection<OXFFeature>
- writeResultToFile(OperationResult, String) : void
HotspotDataAdapter
- createHotspotFeatures(OXFFeatureCollection) : Collection<OXFFeature>
+ main(String[]) : void
+ queryCoverageData(String, IBoundingBox, ITime) : File
+ queryFeatureData(String, Set<Coordinate>, ITime) : Collection<OXFFeature>
- readFromFile(String) : InputStream
+ requestHotspots(Date, Date) : OXFFeatureCollection
~ sortHotspotsByDay(OXFFeatureCollection) : void
+ writeHotspotsToKB(String, OXFFeatureCollection) : void

Figure 6.14: The *DataAdapter* interface with three implementation classes, including the *Hotspot-DataAdapter*

any valid implementation class of the *DataAdapter*.

6.2.1.3 Capability description

A Sensor Agent's service description describes the spatial, temporal and thematic aspects of its *DataSet* (see figure 4.4). *DataSet* descriptions can be searched by specifying combinations of thematic, spatial and temporal constraints as described in section 4.6.1.1.

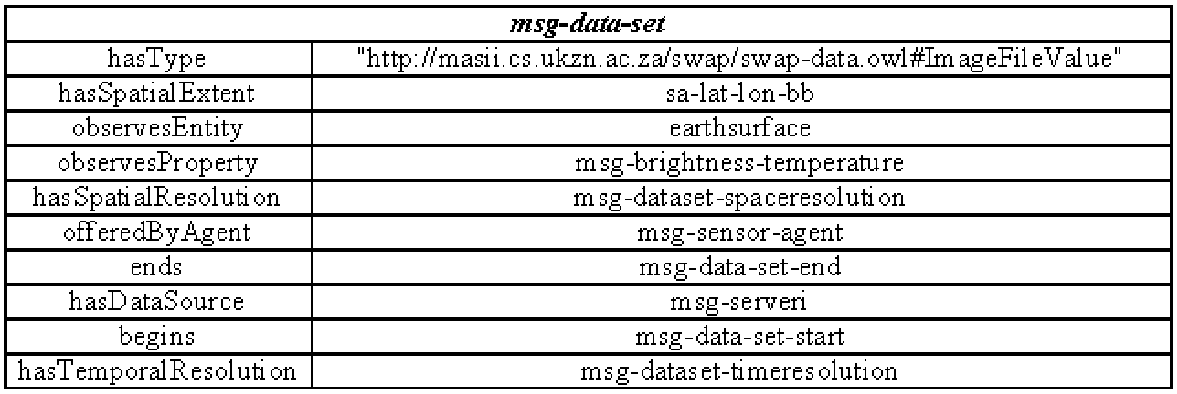
6.2.2 The MSG Sensor Agent

The MSG Sensor Agent is an instance of a Sensor Agent. It serves data from the MSG data set which is described by the *DataSet* instance, *msg-data-set*, shown in figure 6.15. The spatial property values are shown in figure 6.16 and the temporal property values are shown in figure 6.17. The *msg-data-request* protocol shown in figure 6.18 is used to query the MSG data set. It contains input and output message templates for constructing data request messages, and interpreting response messages containing the requested data. A sample *QueryAction* message to request brightness temperature data from the MSG data set is shown in figure 6.19. The request is for data observed at a specific time instant, i.e. on $1\ December$ 2006 at 06:15 and over a specific rectangular region on the earth's surface, i.e. between -17.229 and -34.448 degrees longitude and between 17 and 33 degrees latitude. The response message containing the requested data is shown in figure 6.20. The message also contains a GeoTiff file attachment. This illustrates how the MSG sensor agent provides MSG brightness temperature data to other agents.

6.2.2.1 Discovering the MSG Sensor Agent

The MSG Sensor Agent registers an entry for its sensor data set service with the SWAP Directory Agent. The *Service* instance, *msg-data-service*, is shown in figure 6.21.

For the prototype application, the data set hosted by the MSG Sensor Agent contained data with a spatial resolution of $6.8\ km^2$, observed over South Africa (figure 6.16), between $1\ Dec\ 06:00:00$ to $1\ Dec\ 06:22:00$ and with a temporal resolution of 15 minutes (figure 6.17). The protocol *msg-data-protocol* provides details for using the service. The request action instance, *bt-query-sensor-agent-input*



msg-data-set	
hasType	"http://masii.cs.ukzn.ac.za/swap/swap-data.owl#ImageFileValue"
hasSpatialExtent	sa-lat-lon-bb
observesEntity	earthsurface
observesProperty	msg-brightness-temperature
hasSpatialResolution	msg-dataset-spaceresolution
offeredByAgent	msg-sensor-agent
ends	msg-data-set-end
hasDataSource	msg-serveri
begins	msg-data-set-start
hasTemporalResolution	msg-dataset-timeresolution

Figure 6.15: The MSG DataSet instance

mapping, is a message template used to construct a request message to invoke the MSG data set service. The template contains fixed and variable property values. The fixed values must be copied as is, while the variable values are constrained by the service description. In the above example the fixed properties are $observesProperty$ and $observesEntity$ with values $msg - brightness - temperature$ and $earths - surface$, while the $locatedAt$ property must be a location inside South Africa, and the $hasInstant$ must be a time instant between $1\ Dec\ 06:00:00$ to $1\ Dec\ 06:22:00$ (see figure 6.19). The response message template, *bt-query-sensor-agent-output-mapping*, works in a similar way and provides a template of the expected response message from this agent. Expected values are represented as variables, which are populated by the agent in the response message (see figure 6.20).

The MSG Sensor Agent can be discovered by searching the SWAP Directory Agent for *DataSet* services. A *SearchDirectoryAction* instance, *request-dataset-services*, is shown in figure 6.22. The *hasServiceDescriptionValue* property take on an instance of *DataSet*, *temp-dataset*, so that only services with *DataSet* descriptions are searched. The properties of *temp-dataset* specify three criteria for *DataSets*: a temporal constraint, *intContains req-time-interval*, i.e. only *DataSets* that contain some data observed during the time interval *req-time-interval*; a spatial constraint, *intersects test-feature*, i.e. only *DataSets* that contain some measurements over the spatial location occupied by *test-feature*; and a thematic constraint *observesProperty temperature*, i.e. only data *DataSets* that measures some type of *Temperature*. All data sets that have some type of *Temperature* as the value of its *observesProperty* will match. The

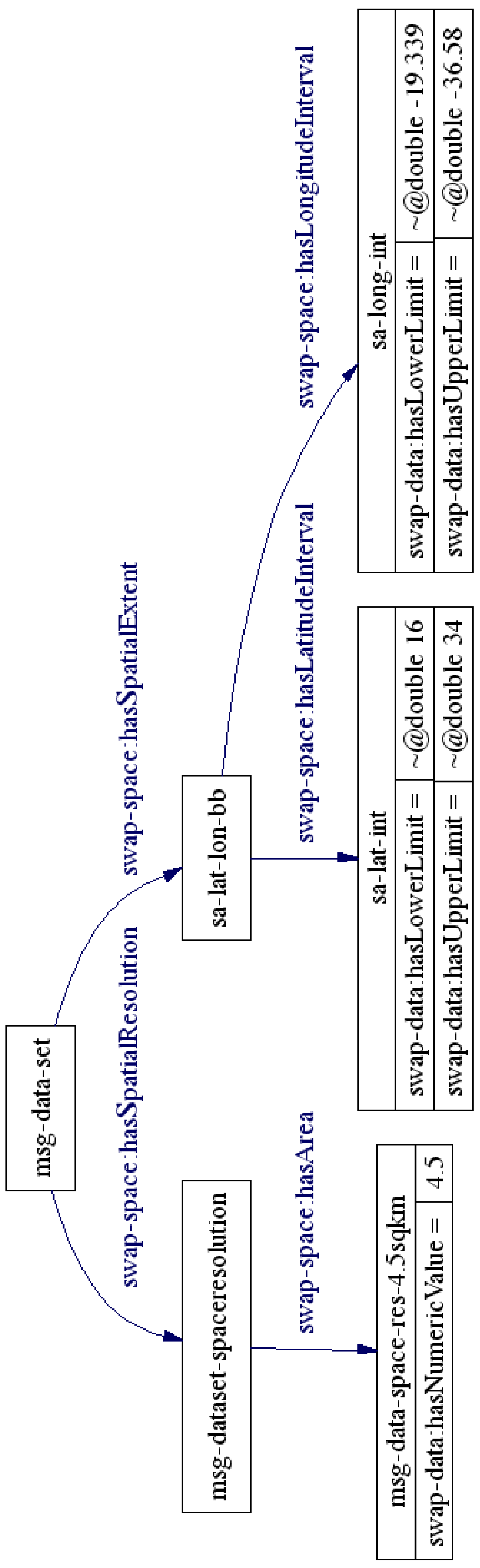

msg-data-set
swap-space:hasSpatialResolution
swap-space:hasSpatialExtent
msg-dataset-spaceresolution
sa-lat-lon-bb
swap-space:hasArea
swap-space:hasLatitudeInterval
swap-space:hasLongitudeInterval
msg-data-space-res-4.5sqkm
swap-data:hasNumericValue = 4.5
sa-lat-int
swap-data:hasLowerLimit = ~@double 16
swap-data:hasUpperLimit = ~@double 34
sa-long-int
swap-data:hasLowerLimit = ~@double -19.339
swap-data:hasUpperLimit = ~@double -36.58

Figure 6.16: Spatial properties of the MSG data set

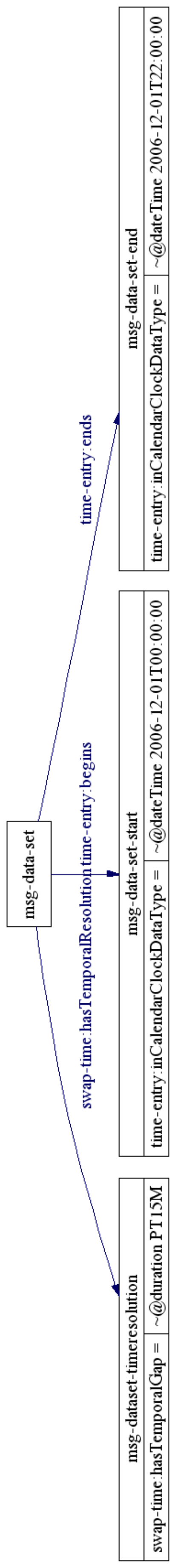

msg-data-set
swap-time:hasTemporalResolution
time-entry:begins
time-entry:ends
msg-dataset-timeresolution
swap-time:hasTemporalGap = | ~@duration PT15M
msg-data-set-start
time-entry:inCalendarClockDataType = | ~@dateTime 2006-12-01T00:00:00
msg-data-set-end
time-entry:inCalendarClockDataType = | ~@dateTime 2006-12-01T22:00:00

Figure 6.17: Temporal properties of the MSG data set

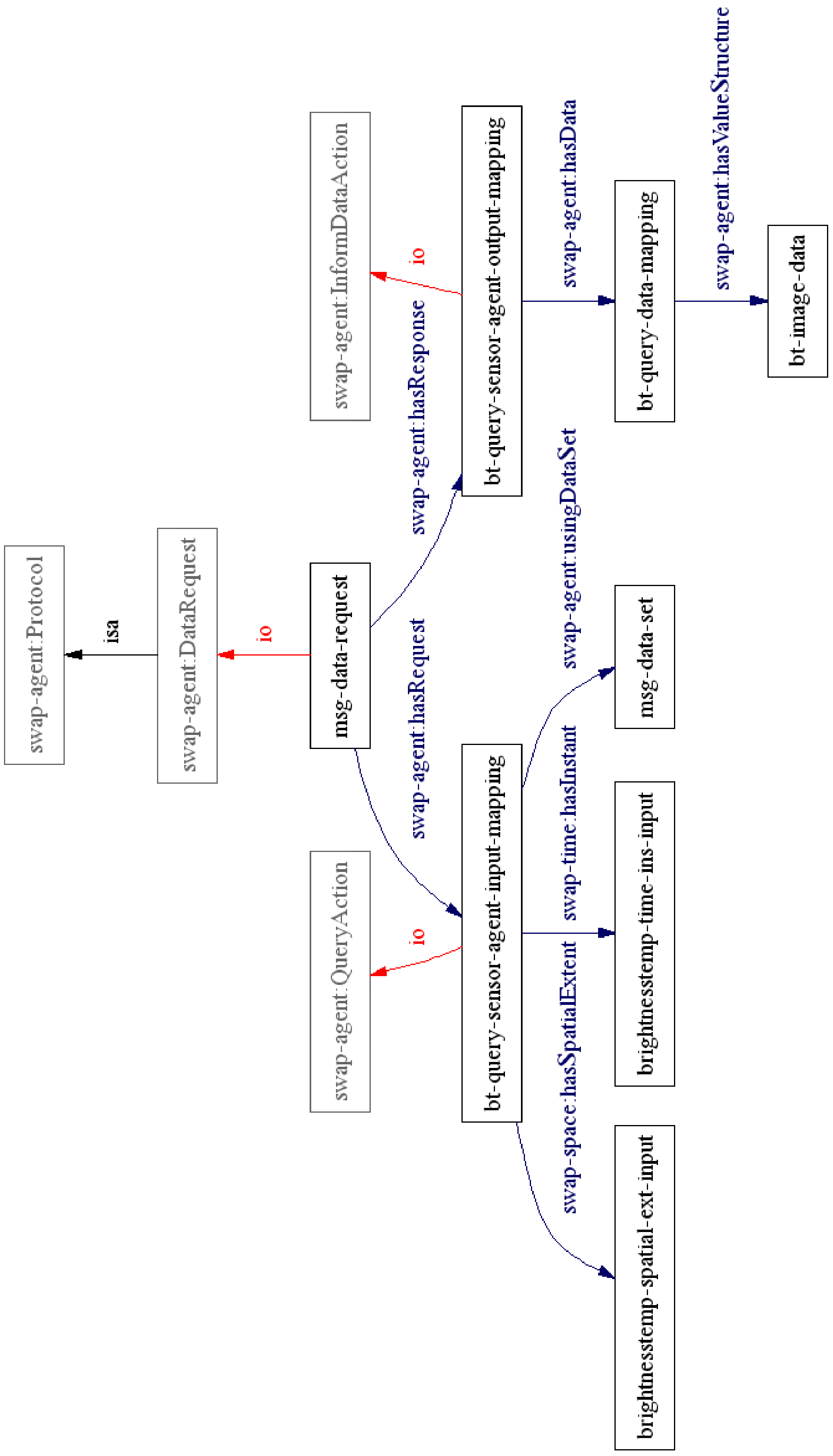


Figure 6.18: The data request protocol used to query the MSG data set

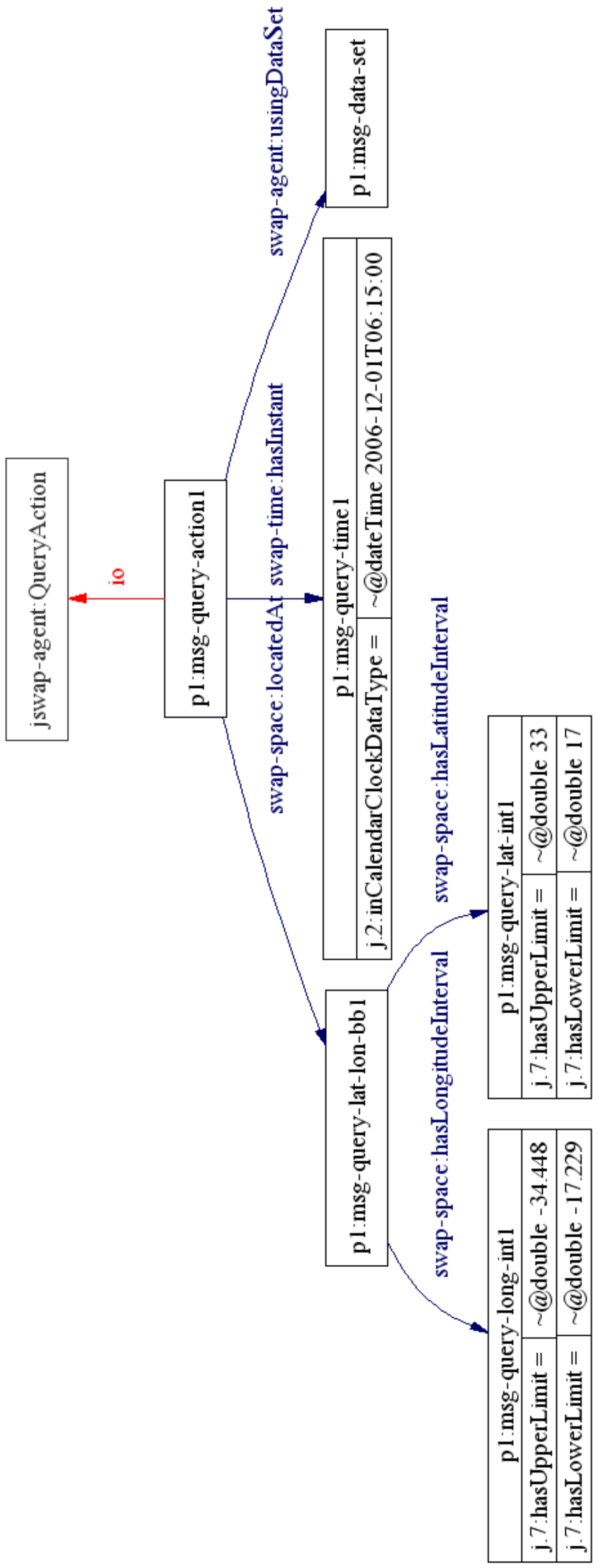

jswap-agent:QueryAction
io
p1:msg-query-action1
swap-agent:usingDataSet
p1:msg-data-set
swap-time:hasInstant
p1:msg-query-time1
j.2:inCalendarClockDataType = ~@dateTime 2006-12-01T06:15:00
swap-space:locatedAt
p1:msg-query-lat-lon-bb1
swap-space:hasLatitudeInterval
p1:msg-query-lat-int1
j.7:hasUpperLimit = ~@double 33
j.7:hasLowerLimit = ~@double 17
swap-space:hasLongitudeInterval
p1:msg-query-long-int1
j.7:hasUpperLimit = ~@double -34.448
j.7:hasLowerLimit = ~@double -17.229

Figure 6.19: A data request message for querying the MSG data set

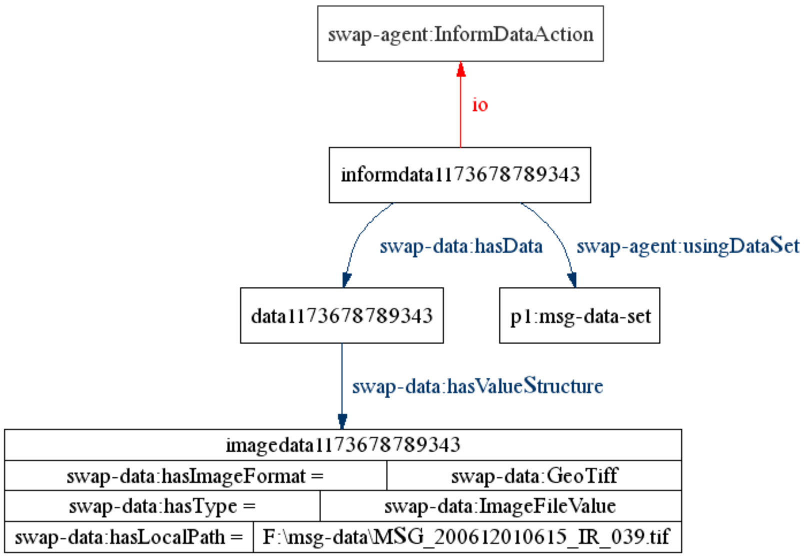


Figure 6.20: A data response message for querying the MSG data set

MSG data set offered by the MSG Sensor Agent in the wildfire detection application has an instance of *BrightnessTemperature* as its *observesProperty*, and since *BrightnessTemperature* is a type of *Temperature*, the MSG data set will match this query. There could also be other possible matches. Suppose that a Weather Agent existed in the system that served a *DataSet* for air temperature, i.e. has an instance of *DryBulbTemperature* as its *observesProperty*. Since *DryBulbTemperature* is also a type of *Temperature*, this data set will also match. In this case, since the MSG data service satisfies all three constraints, it is sent back to the requesting agent in an *InformServicesAction* as shown in figure 6.23. The service instance, *msg-data-service*, is specified using the *hasService* property and contains all the information required to locate (*msg-sensor-agent*) and invoke (*bt-query-sensor-agent-input-mapping*) the MSG Sensor Agent and to interpret its response (*bt-query-sensor-agent-output-mapping*).

6.2.3 Tool Agent

Tool Agents take in data and perform predefined processes on the data. A process has at least one input, the data to process, and at least one output, i.e. the processed data or the results of the processing.

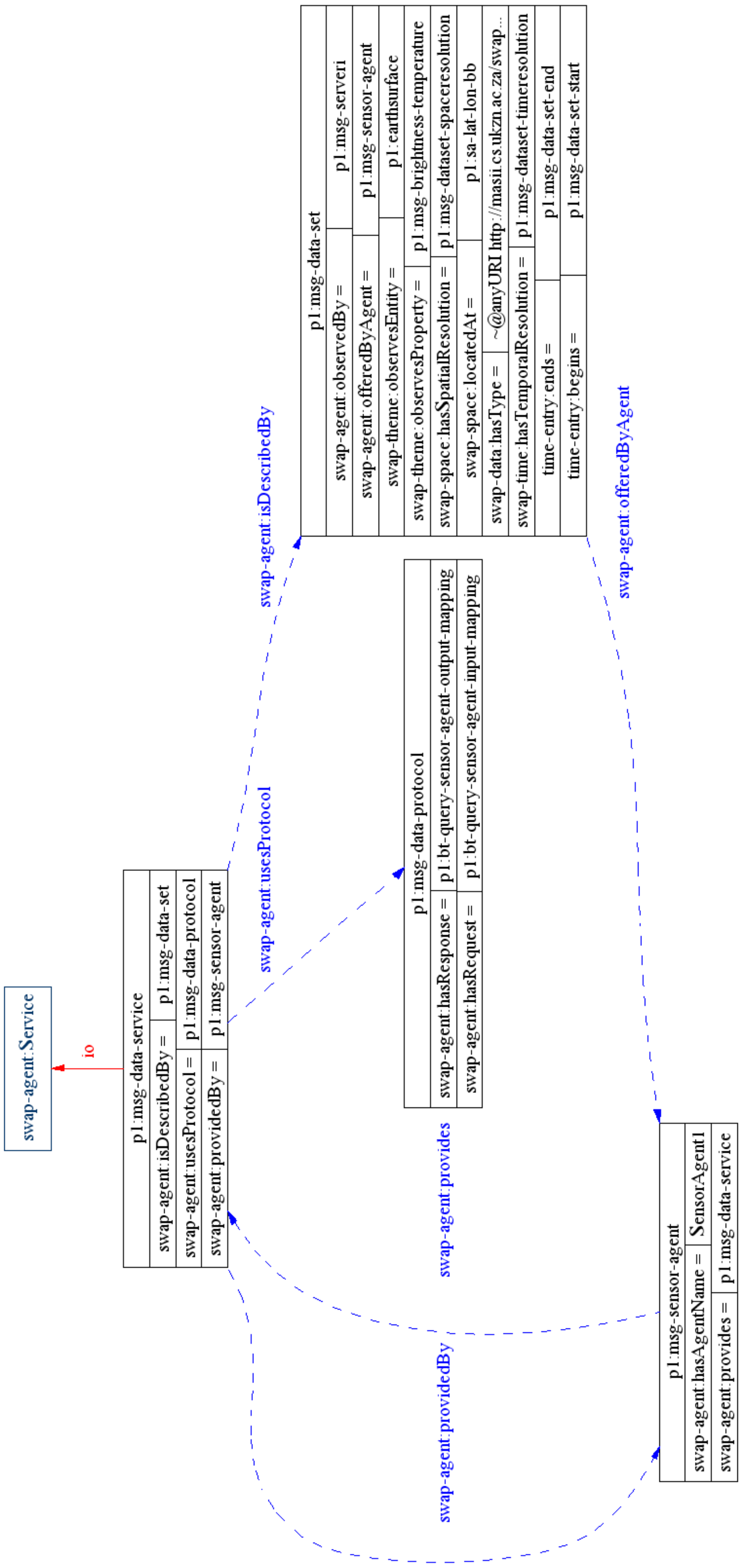

swap-agent:Service
io
p1:msg-data-service
swap-agent:isDescribedBy = p1:msg-data-set
swap-agent:usesProtocol = p1:msg-data-protocol
swap-agent:providedBy = p1:msg-sensor-agent
swap-agent:isDescribedBy
swap-agent:usesProtocol
p1:msg-data-set
swap-agent:observedBy = p1:msg-serveri
swap-agent:offeredByAgent = p1:msg-sensor-agent
swap-theme:observesEntity = p1:earthsurface
swap-theme:observesProperty = p1:msg-brightness-temperature
swap-space:hasSpatialResolution = p1:msg-dataset-spaceresolution
swap-space:locatedAt = p1:sa-lat-lon-bb
swap-data:hasType = ~@anyURI http://masii.cs.ukzn.ac.za/swap...
swap-time:hasTemporalResolution = p1:msg-dataset-timeresolution
time-entry:ends = p1:msg-data-set-end
time-entry:begins = p1:msg-data-set-start
p1:msg-data-protocol
swap-agent:hasResponse = p1:bt-query-sensor-agent-output-mapping
swap-agent:hasRequest = p1:bt-query-sensor-agent-input-mapping
swap-agent:offeredByAgent
swap-agent:provides
swap-agent:providedBy
p1:msg-sensor-agent
swap-agent:hasAgentName = SensorAgent1
swap-agent:provides = p1:msg-data-service

Figure 6.21: The *Service* instance for the MSG Sensor Agent

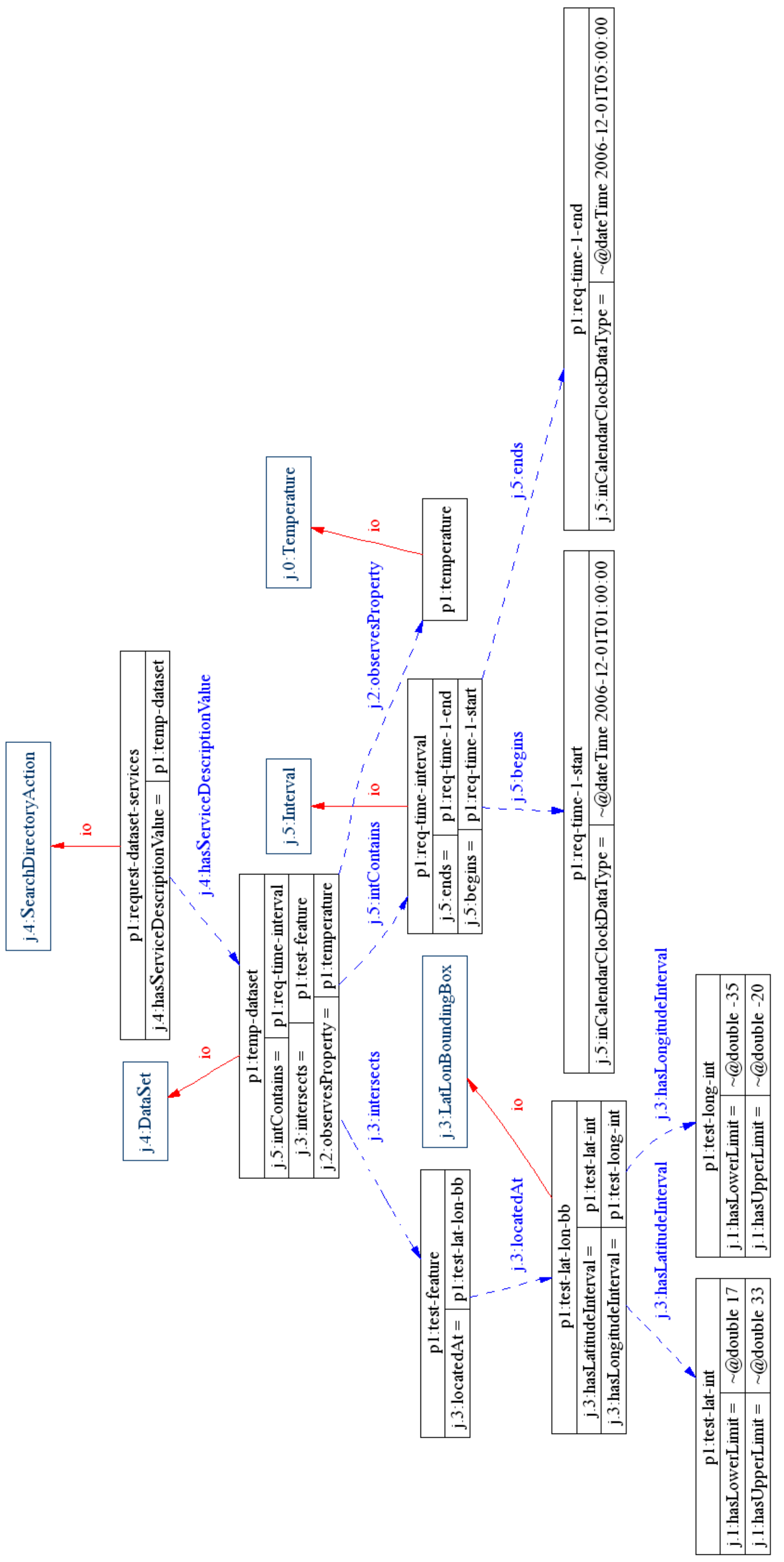

j.4:SearchDirectoryAction
io
p1:request-dataset-services
j.4:hasServiceDescriptionValue = | p1:temp-dataset
j.4:hasServiceDescriptionValue
j.4:DataSet
io
p1:temp-dataset
j.5:intContains = | p1:req-time-interval
j.3:intersects = | p1:test-feature
j.2:observesProperty = | p1:temperature
j.5:Interval
io
j.5:intContains
j.2:observesProperty
j.0:Temperature
io
p1:temperature
p1:req-time-interval
j.5:ends = | p1:req-time-1-end
j.5:begins = | p1:req-time-1-start
j.5:begins
j.5:ends
p1:req-time-1-start
j.5:inCalendarClockDataType = | ~@dateTime 2006-12-01T01:00:00
p1:req-time-1-end
j.5:inCalendarClockDataType = | ~@dateTime 2006-12-01T05:00:00
j.3:intersects
p1:test-feature
j.3:locatedAt = | p1:test-lat-lon-bb
j.3:LatLonBoundingBox
io
j.3:locatedAt
p1:test-lat-lon-bb
j.3:hasLatitudeInterval = | p1:test-lat-int
j.3:hasLongitudeInterval = | p1:test-long-int
j.3:hasLatitudeInterval
j.3:hasLongitudeInterval
p1:test-lat-int
j.1:hasLowerLimit = | ~@double 17
j.1:hasUpperLimit = | ~@double 33
p1:test-long-int
j.1:hasLowerLimit = | ~@double -35
j.1:hasUpperLimit = | ~@double -20

Figure 6.22: A search request message for data set services that, observe any temperature property, in any part of the given location (intersects with) and during the given time interval

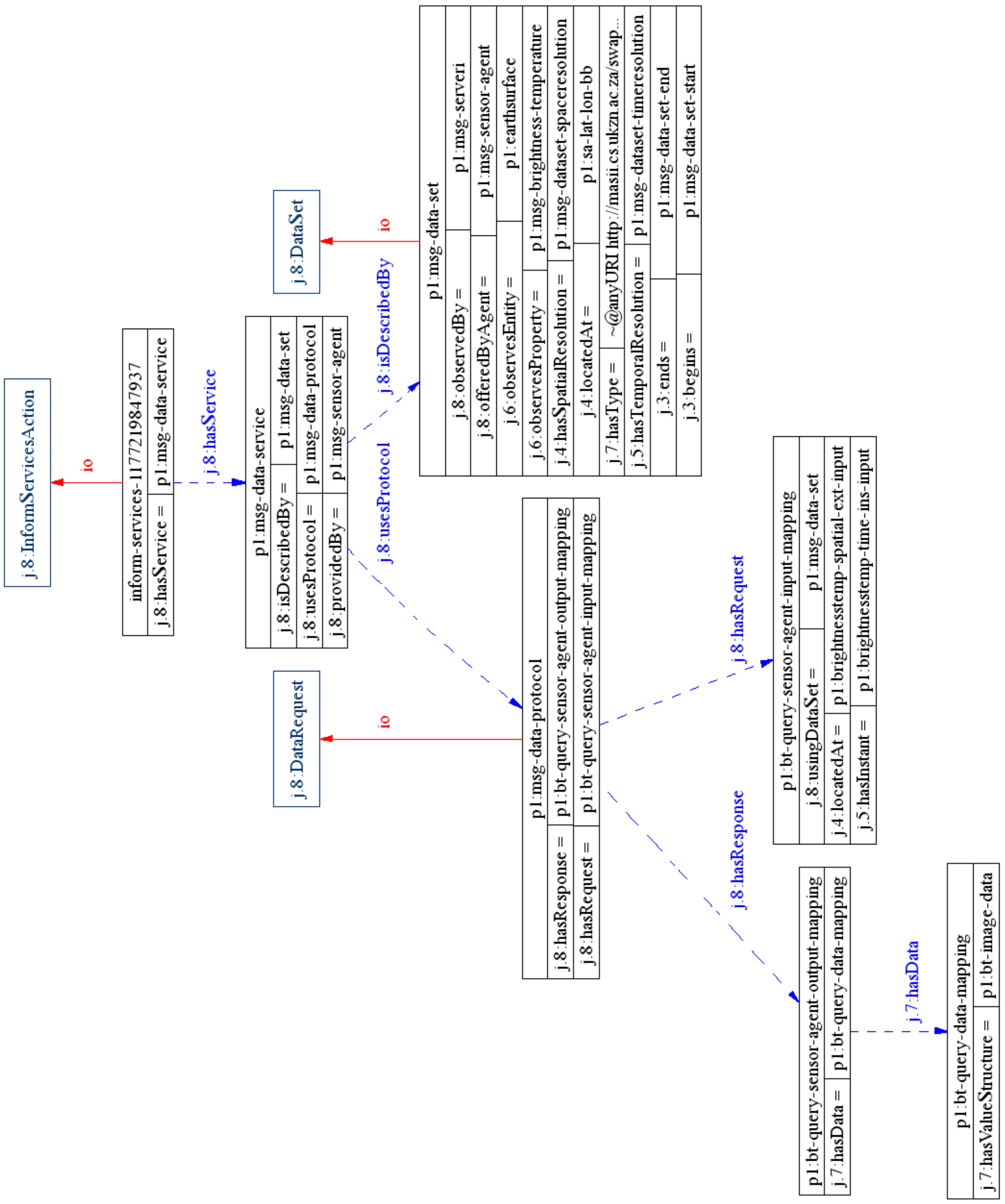

j.8:InformServicesAction
io
inform-services-1177219847937
j.8:hasService = p1:msg-data-service
j.8:hasService
p1:msg-data-service
j.8:isDescribedBy = p1:msg-data-set
j.8:usesProtocol = p1:msg-data-protocol
j.8:providedBy = p1:msg-sensor-agent
j.8:DataSet
io
j.8:isDescribedBy
p1:msg-data-set
j.8:observedBy = p1:msg-serveri
j.8:offeredByAgent = p1:msg-sensor-agent
j.6:observesEntity = p1:earthsurface
j.6:observesProperty = p1:msg-brightness-temperature
j.4:hasSpatialResolution = p1:msg-dataset-spaceresolution
j.4:locatedAt = p1:sa-lat-lon-bb
j.7:hasType = ~@anyURI http://masii.cs.ukzn.ac.za/swap...
j.5:hasTemporalResolution = p1:msg-dataset-timeresolution
j.3:ends = p1:msg-data-set-end
j.3:begins = p1:msg-data-set-start
j.8:usesProtocol
j.8:DataRequest
io
p1:msg-data-protocol
j.8:hasResponse = p1:bt-query-sensor-agent-output-mapping
j.8:hasRequest = p1:bt-query-sensor-agent-input-mapping
j.8:hasRequest
p1:bt-query-sensor-agent-input-mapping
j.8:usingDataSet = p1:msg-data-set
j.4:locatedAt = p1:brightnesstemp-spatial-ext-input
j.5:hasInstant = p1:brightnesstemp-time-ins-input
j.8:hasResponse
p1:bt-query-sensor-agent-output-mapping
j.7:hasData = p1:bt-query-data-mapping
j.7:hasData
p1:bt-query-data-mapping
j.7:hasValueStructure = p1:bt-image-data

Figure 6.23: A search response message, that contains a single matching service satisfying the search criteria specified in figure 6.22

Table 6.4: Tool role schema

Role Schema:	Tool Provider
Description	Provides a tool which performs some processing on the supplied data. Registers and provides a description of its process.
Protocols and Activities	DataProcessing, DescribeCapabilities, SWAPServiceRegistration
Permissions	Update capabilities in registry
Responsibilities	• Register and maintain capabilities with Directory Agent • Perform processing requests and respond with the corresponding results

Table 6.5: Process Data protocol schema

Protocol	Data Processing
Initiator	Any agent e.g. workflow agent
Participants	Tool Agent, initiating agent
Inputs	Process inputs
Outputs	Process outputs
Message Types	ProcessDataAction, InformResultAction
Description	An agent can request for the Tool Agent to perform some process given valid inputs. The Tool Agent responds with the outputs or results.

6.2.3.1 External representation

The Tool Agent encapsulates an executable process in the system. The process is executed by specifying values for the inputs of the process. After execution, the process produces values for its outputs. The process is invoked by sending the *ToolAgent* a *ProcessDataAction* message containing a *ProcessDataAction* instance. The *ProcessDataAction* instance has a *hasInput* property that is used to specify the inputs for the process. Once the process has completed, an *InformResultAction* instance is sent to the requesting agent. The *InformResultAction* instance has a *hasOutput* property that is used to specify the outputs of the process. The values of both the *hasInput* and *hasOutput* properties are represented as SWRL local variables (see section 4.5.4). SWRL variables have a type, i.e. the URI of the object or XML datatype, and a value which is either a literal or the URI of the object if it is an object type. The variable name is used to specify the name of the input or output parameter, and the variable value is used for specifying the corresponding value of the parameter. The Tool Provider role and *DataProcessing* protocol are shown in tables 6.4 and 6.5 respectively.

6.2.3.2 Internal operation

Internally, the Tool Agent accesses its process implementation via a Java interface, the *ToolAdapter* interface (figure 6.24), which provides standard methods for interacting with a process. Specific processes are implemented as *ToolAdapter* classes. The *ToolAdapter* interface specifies a process as having one or more inputs and one or more outputs. It provides standard methods for setting the inputs of the process, executing the process and retrieving the results of the process. A *ToolAdapter* implementation class uses two map data structures to store the names and corresponding values of input and output parameters. The input map stores input parameters, while the output map stores output parameters. The *setInput* method is used to set the values of input parameters, and returns an error if an incorrect input parameter is set. The $getOutputs$ method retrieves the expected outputs of this process and the $checkValidOutput$ method checks whether the given output parameter name is a valid output. The $process$ method is executed after all inputs have been set, and the $getOutput$ method is used to retrieve the value of each output parameter by supplying the parameter name. As with the *DataAdapter* used in the sensor agent, GeoAPI data structures are used for representing time, coverage and feature data values. This interface provides a standard internal representation for implementing and executing any process, which has one or more inputs and one or more outputs.

6.2.3.3 Capability description

Each Tool Agent registers a *Service* instance with the SWAP Directory Agent that describes its functionality. The service description contained in the *isDescribedBy* property is an instance of the *Tool* class. It has two properties *hasInputType* and *hasOutputType* that specify the inputs and outputs of specific *Tool* services. Other agents are able to search for *Tool* service descriptions that have specific inputs and outputs.

6.2.4 The Contextual Algorithm (CA) Tool Agent

The CA Tool Agent was developed for the wildfire detection application to detect abnormally high values or hotspots from brightness temperature raster data from the MSG Sensor Agent. The design of the CA Tool Agent is such that it is not tightly bound to brightness temperature data. It can be used in any

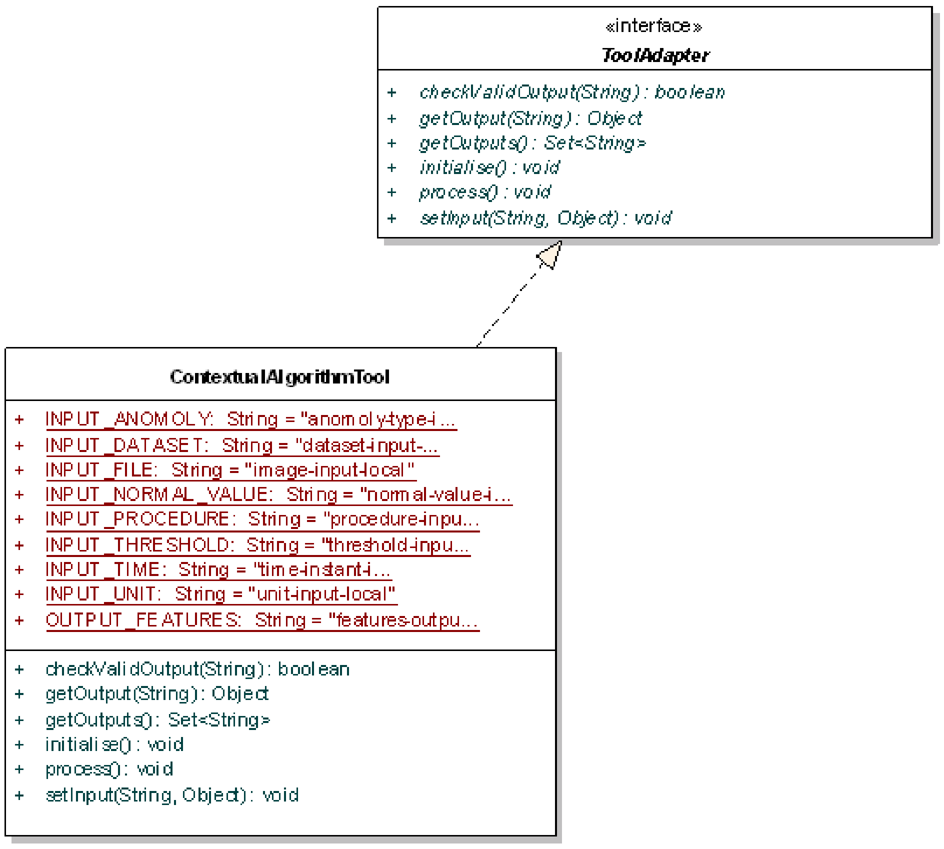

«interface»
ToolAdapter
+ checkValidOutput(String) : boolean
+ getOutput(String) : Object
+ getOutputs() : Set<String>
+ initialise() : void
+ process() : void
+ setInput(String, Object) : void
ContextualAlgorithmTool
+ INPUT_ANOMOLY: String = "anomoly-type-i...
+ INPUT_DATASET: String = "dataset-input-...
+ INPUT_FILE: String = "image-input-local"
+ INPUT_NORMAL_VALUE: String = "normal-value-i...
+ INPUT_PROCEDURE: String = "procedure-inpu...
+ INPUT_THRESHOLD: String = "threshold-inpu...
+ INPUT_TIME: String = "time-instant-i...
+ INPUT_UNIT: String = "unit-input-local"
+ OUTPUT_FEATURES: String = "features-outpu...
+ checkValidOutput(String) : boolean
+ getOutput(String) : Object
+ getOutputs() : Set<String>
+ initialise() : void
+ process() : void
+ setInput(String, Object) : void

Figure 6.24: Class diagram of the ToolAdapter interface

application that requires abnormally high value (hotspots) or abnormally low values (lowspots) to be detected from raster data.

Internally an implementation of the *ToolAdapter* interface is used to carry out the processing of the agent. The *ContextualAlgorithmTool* class, shown in figure 6.24, is an implementation of the *ToolAdapter* interface. It accepts eight inputs, which include a geo-referenced image file representing a coverage (*image-input-local*) and the observation time of the coverage (*time-instance-input-local*), and has one output, a Java collection of *OXFFeatures* (*feature-output-local*). These features represent specific filtered pixel values from the image, the real-world coordinates of the pixels, the observation time and other properties that describe this feature (see figure 6.26).

The *ContextualAlgorithmTool* implements a general contextual algorithm based on the contextual algorithm used in the AFIS application [71]. It takes as input a georeferenced image and performs a contextual search through the data to detect abnormally high or abnormally low values. The search involves comparing each pixel value with surrounding pixel values to detect abnormally high or abnormally low pixels in a given neighbourhood. The search algorithm is defined as:

$$anomoly\_Features = detectAnomolies(img, anom, thresh, avg\_val) \tag{6.1}$$

where img is a georeferenced image file containing floating point values, $anom$ is the anomaly to detect in img, either $Lowspot$ to detect abnormal lows, or $Hotspot$ to detect abnormal highs, $thresh$ is the minimum or maximum threshold i.e. only check those values that are higher or lower than the threshold, and avg_val is the average expected value of a pixel.

For detecting abnormally high values, the algorithm first scans the image, img, for a high pixel value, h, such that $h > thresh$. It then checks that a significant number of the surrounding pixels in a square window (e.g. 21x21 pixels) are less than the average expected value, avg_val. The mean, m, and standard deviation, s, of all pixel values in the window are then calculated. If $h > m + s$, then h is flagged as an abnormally high value. This process is repeated for all h, i.e. all pixels that are greater than $thresh$. Abnormally low values are detected in a similar manner, except that all greater than comparison operations would be replaced by less than operators.

For wildfire detection, the contextual algorithm is used to detect temperature hotspots with $anom = Hotspot$, $thresh = 315\ kelvin$ and $avg\_val = 312\ kelvin$. In this case each pixel value is checked for high values ($h > 315$). Then an expanding window with h at the centre, starting at 3x3 pixels and ending

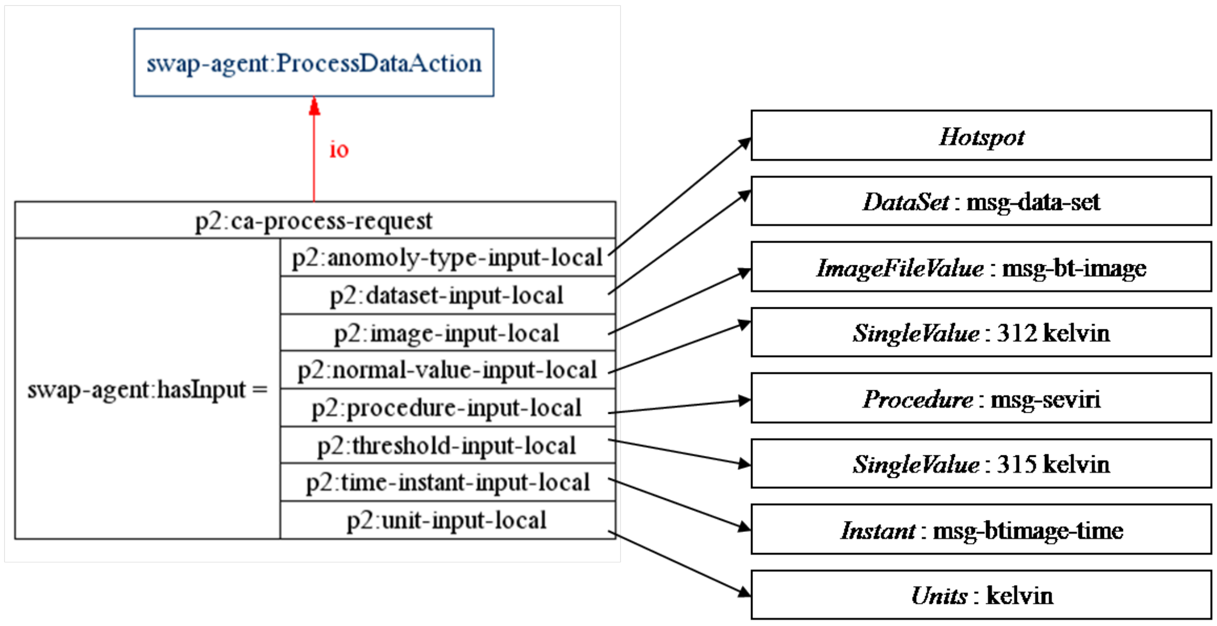


Figure 6.25: A process data request to invoke the CA Tool Agent

at 21x21 pixels, is created. If 33% of the pixels in the window are greater than 312 then the statistical check is done. The mean (m) and standard deviation (s) of the pixels in the window are calculated. If $h > m + s$ then the window is expanded. If $h > 315$ and $h > m + s$ in the 21x21 window then h is a temperature hotspot, i.e. a potential wildfire.

A *ProcessDataAction* instance, *ca-process-request*, is shown in figure 6.25. The *image-input-local* parameter contains: an instance of *ImageFileValue* (*msg-bt-image*); type of anomaly to detect, in this case hotspots represented by the *Hotspot* class; the threshold value contained in *threshold-input-local*; and the normal value contained in *normal-value-input*. Other ancillary data such as the observation time (*time-instant-input-local*), the procedure used to obtain these measurements (*procedure-input-local*) and the description of the dataset to which this data belongs (*dataset-input-local*) is also supplied and used to provide ancillary properties for the output features.

The CA Tool Agent responds with a *InformResultAction* message containing an *InformResultAction* instance (figure 6.26). The contextual algorithm has a single output, *feature-output-local* which is a list containing instances of *SingleValues* representing either *Hotspot* or *Lowspot* features. A sample hotspot feature, *positiondata1-1174198022281*, is shown in figure 6.26. The *hasNumericValue* property contains the temperature of this hotspot which is 315.65 kelvin for this particular hotspot. Other ancillary properties of this feature, such as the data set to which it belongs, are obtained from the input parameters

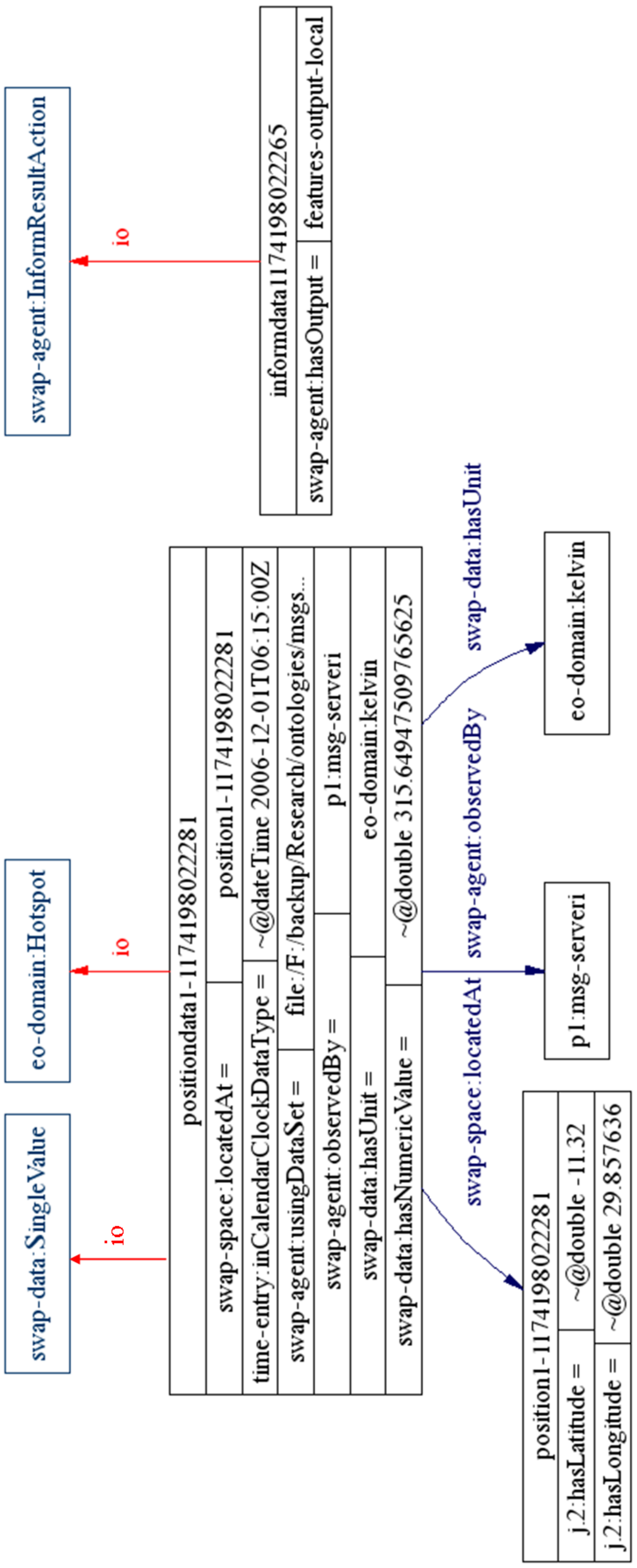

swap-data:SingleValue
eo-domain:Hotspot
swap-agent:InformResultAction
io
io
io
positiondata1-1174198022281
swap-space:locatedAt = | position1-1174198022281
time-entry:inCalendarClockDataType = | ~@dateTime 2006-12-01T06:15:00Z
swap-agent:usingDataSet = | file:/F:/backup/Research/ontologies/msgs...
swap-agent:observedBy = | p1:msg-serveri
swap-data:hasUnit = | eo-domain:kelvin
swap-data:hasNumericValue = | ~@double 315.64947509765625
informdata1174198022265
swap-agent:hasOutput = | features-output-local
swap-space:locatedAt
swap-agent:observedBy
swap-data:hasUnit
position1-1174198022281
j.2:hasLatitude = | ~@double -11.32
j.2:hasLongitude = | ~@double 29.857636
p1:msg-serveri
eo-domain:kelvin

Figure 6.26: A process data response from the CA Tool Agent

and provided as additional details for this feature.

6.2.4.1 Discovering the CA Tool Agent

The CA Tool Agent registers two service descriptions with the Directory Agent, one for calculating hotspots and the other for calculating lowspots as shown in figure 6.27. Both services take image data as input and output *SpatialAnomalies* that are either *Hotspots* or *LowSpots*.

These services can be discovered by searching for Tool Services that take as input an *ImageFileValue* and output either a *SpatialAnomaly* or more specifically a *Hotspot* or a *Lowspot*. An example search request is shown in figure 6.28. The search request specifies only Tool services that have an input of type *ImageFileValue* and an output of type *SpatialAnomaly*. Since the service descriptions of both CA Tool services, *ca-tool-lowspot-service* and *ca-tool-hotspot* service (figure 6.27) match, both services are returned.

6.2.5 Workflow Agent

The Workflow Agent coordinates activities between different agents to form processing chains or workflows. Internally, the workflow agent represents its interactions with other agents as an OWL-S composite process, consisting of atomic processes for each separate agent interaction. Externally the Workflow Agent is represented and invoked as a process, in a similar manner to the Tool Agent.

6.2.5.1 External representation

A workflow is represented as a process with input and output parameters. The invocation of the workflow is similar to that of the Tool Agent described above. An *ExecuteWorkflowAction* message containing a *ExecuteWorkflowAction* instance is used to invoke the Workflow Agent. The *ExecuteWorkflowAction* instance has a *hasInput* property that is used to specify values for all inputs for the workflow and initiates the execution of the workflow. Once the workflow has completed, an *InformResultAction* instance is sent to the requesting agent. The *InformResultAction* instance has a *hasOutput* property that is used to specify the outputs of the workflow. The workflow provider role and the workflow execution protocol are described in table 6.6 and table 6.7 respectively.

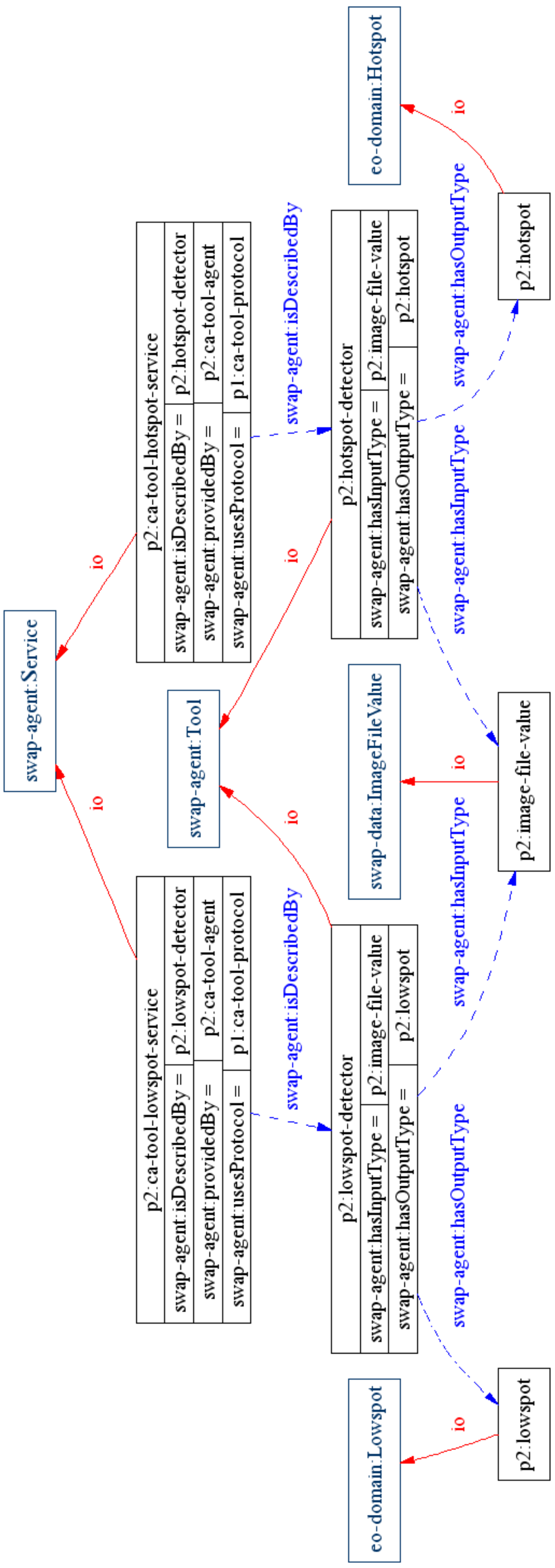

swap-agent:Service
io
io
p2:ca-tool-hotspot-service
swap-agent:isDescribedBy = p2:hotspot-detector
swap-agent:providedBy = p2:ca-tool-agent
swap-agent:usesProtocol = p1:ca-tool-protocol
p2:ca-tool-lowspot-service
swap-agent:isDescribedBy = p2:lowspot-detector
swap-agent:providedBy = p2:ca-tool-agent
swap-agent:usesProtocol = p1:ca-tool-protocol
swap-agent:Tool
io
io
swap-agent:isDescribedBy
swap-agent:isDescribedBy
p2:hotspot-detector
swap-agent:hasInputType = p2:image-file-value
swap-agent:hasOutputType = p2:hotspot
p2:lowspot-detector
swap-agent:hasInputType = p2:image-file-value
swap-agent:hasOutputType = p2:lowspot
eo-domain:Hotspot
swap-data:ImageFileValue
eo-domain:Lowspot
io
io
io
swap-agent:hasOutputType
swap-agent:hasInputType
swap-agent:hasInputType
swap-agent:hasOutputType
p2:hotspot
p2:image-file-value
p2:lowspot

Figure 6.27: Service entries for the CA Tool Agent

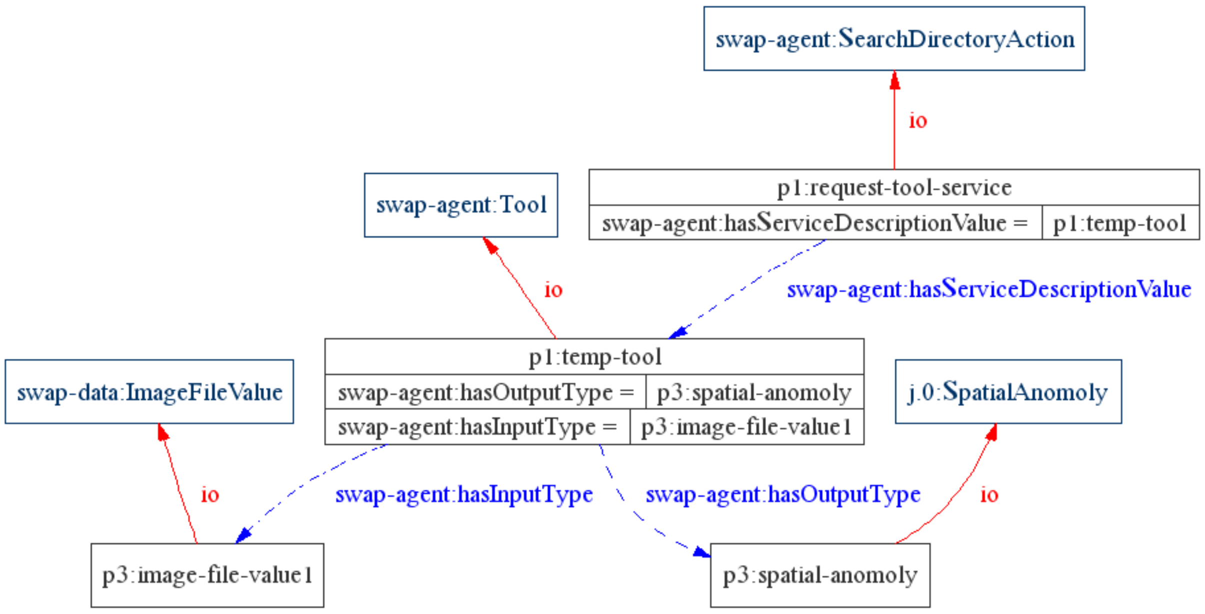

swap-agent:SearchDirectoryAction
io
swap-agent:Tool
p1:request-tool-service
swap-agent:hasServiceDescriptionValue = p1:temp-tool
io
swap-agent:hasServiceDescriptionValue
p1:temp-tool
swap-agent:hasOutputType = p3:spatial-anomoly
swap-agent:hasInputType = p3:image-file-value1
swap-data:ImageFileValue
j.0:SpatialAnomoly
io
swap-agent:hasInputType
swap-agent:hasOutputType
io
p3:image-file-value1
p3:spatial-anomoly

Figure 6.28: A search request that matches the CA Tool services

Table 6.6: The Workflow Role schema

Role Schema:	Workflow Provider
Description	Stores and executes a workflow. Registers and provides a description of its workflow.
Protocols and Activities	*WorkflowExecution*, *DescribeCapabilities*, *SWAPServiceRegistration*
Permissions	Update capabilities in registry
Responsibilities	• Register capabilities with Directory Agent • Perform workflow • Maintain workflow

Table 6.7: The *WorkflowExecution* Protocol schema

Protocol	WorkflowExecution
Initiator	Any agent but typically an Application Agent
Participants	Workflow Agent, initiating agent
Inputs	Workflow inputs
Outputs	Workflow outputs
Message Types	*ExecuteWorkflowAction*, *InformResultAction*
Description	An agent can request a workflow to be executed by the Workflow Agent based on the supplied inputs.

6.2.5.2 Internal operation

Internally, OWL-S composite processes are used for representing executable workflows. A composite process consists of two or more atomic processes, each with its own input and output parameters. The Maryland MindSwap OWL-S Java API [4] is used for executing OWL-S processes. The *ProcessExecution-Impl* class in the OWL-S API was extended to include a new method, *executeAgentAction* to implement the *AgentToProcessMapping* (described in section 4.5.4) which allows each atomic process to be mapped to an interaction with an agent. The *AgentToProcessMapping* is used to assemble an agent request message from the input parameters of a process, send the request message to the appropriate agent, receive the response message and extract the process output values from the response message. As opposed to other SWAP agents that use the Jena ontology API for accessing its ontology and knowledge base, the Workflow Agent uses the Mindswap API for accessing its ontology and knowledge base. Each atomic process instance has a *hasAgentMapping* property that contains a *ProcessAgentMapping* instance for this process. The *ProcessAgentMapping* contains input and output agent action message templates. The templates contain variable names for those values that need to be replaced by values from OWL-S variables. It also contains a *providedBy* property which specifies the details of the specific agent that will perform this process.

6.2.5.3 Capability Description

The Workflow Agent registers a *Service* instance with the SWAP Directory Agent that describes its functionality. The service description contained in the *isDescribedBy* property is an instance of the *Workflow* class. It is similar to the *Tool* description and has two properties *hasInputType* and *hasOutputType* that specify the inputs and outputs of specific *Workflow* services. Other agents are able to search for *Workflow* service descriptions that have specific inputs and outputs.

6.2.6 The Hotspot Detection Workflow Agent

The hotspot detection composite process or workflow used in the wildfire detection application is shown in figure 6.29. The process has two inputs, i.e. the spatial extent and the time at which to detect hotspots,

[4]http://www.mindswap.org

and consists of two processing steps or atomic processes. The *get-msg-bt-process* atomic process retrieves brightness temperature data from the MSG Sensor Agent and the *calc-hotspots-process* atomic process calculates hotspot features from this data.

The workflow is executed when the agent receives an *ExecuteWorkflowAction* containing the *hotspot-spatial-ext-input* and the *hotspot-time-ins-input* input parameters. The workflow execution engine works as follows. The *executeAtomic* action calls the *executeAgentAction* method. The message is sent to the agent specified by the *hasAgentName* property of the *Agent* instance in the *providedBy* property of the mapping template. The *ProcessAgentMapping* for the MSG Sensor Agent is shown in figures 6.30 and 6.31. Figure 6.30 illustrates the atomic process *get-msg-bt-process* for retrieving data from the MSG Sensor Agent and shows how this process is mapped to a *QueryAction* message which is sent to the MSG Sensor Agent. The process has two input parameters, i.e. *brightnesstemp-spatial-ext-input* and *brightnesstemp-time-ins-input*. The *bt-query-sensor-agent-input-mapping*, the input mapping of this process, is a well formed *QueryAction* message that can be sent to the MSG Sensor Agent. It has one static property value (*msg-data-set*) which is fixed for all queries and two variable property values (*brightnesstemp-spatial-ext-input* and *brightnesstemp-time-ins-input*). Note that these are named the same as the input parameters of the *get-msg-bt-process* process. When the input parameters of the process are populated, these values are used to populate the variable properties to form a valid *QueryAction* message. Figure 6.31 shows the output mapping which is used to extract values from the *InformDataAction* response message from the MSG Sensor Agent. The *bt-query-sensor-agent-output-mapping* depicts an expected response from the MSG Sensor Agent. The *hasData* property contains a value *bt-query-data-mapping* which has a variable property value *bt-image-data* in the *hasValueStructure* property. *bt-image-data* is also the output parameter of the *get-msg-bt-process* process. Thus, when the *InformDataAction* response message is received, it is parsed and the value of the *hasValueStructure* property is used to populate the *bt-image-data* output parameter.

Figures 6.32 and 6.33 shows the *calc-hotspots-process* and its associated input and output mappings. The *ca-input-mapping* depicts a template of an appropriate *ProcessDataAction* instance that can be used to invoke the CA Tool Agent. It has eight *local* variable values for its *hasInput* property. All input values are fixed except for *image-input-local* that specifies the image data and *time-instant-input-local* that specifies the observation time. The remaining six input parameters take on appropriate default values for calculating hotspots as shown in figure 6.25. The value of each local variable contains the URI of

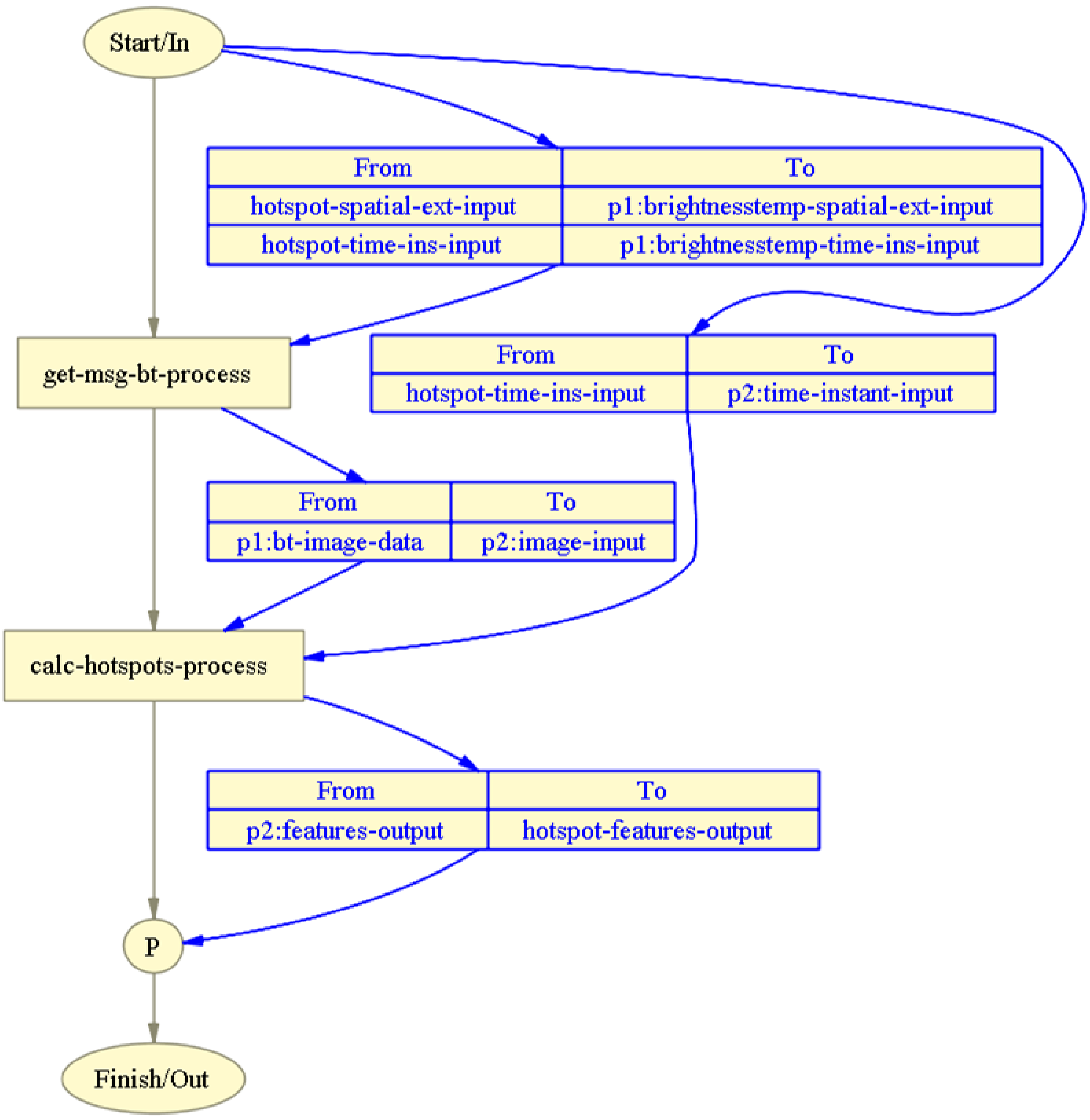


Figure 6.29: The OWL-S workflow for hotspot detection

the appropriate input parameter of the *calc-hotspots-process* and is used to link each variable value to the appropriate input parameter. In this case *image-input-local* has the URI of *image-input* while *time-instant-input-local* has the URI of *time-instant-input*. When the *calc-hotspots-process* is executed, the values supplied in *image-input* and *time-instant-input* are used to replace the value in *image-input-local* and *time-instant-input* respectively to form a *ProcessDataAction* message that is sent to the CA Tool Agent. The CA Tool Agent responds with a *InformResultAction* message containing the hotspots found in the supplied image data. The *ca-output-mapping* instance shown in 6.33 is an expected response message from the CA Tool Agent. The *hasOutput* property contains a variable property value *features-output-local* which has the URI of the *features-output* output parameter of the *calc-hotspots-process*. When the *InformResultAction* response message is received, it is parsed and the value of the *features-output-local* local variable is used to populate the *features-output* output parameter. This value is then used to populate the *hotspot-features-output* parameter, the only output parameter of the workflow, and the workflow completes. The agent composes an *InformResultAction* message with the *hotspot-features-output* output parameter and sends this to the invoking agent.

6.2.6.1 Discovering the Hotspot Detection Workflow Agent

The HDWA can be discovered in a similar manner to the discovery of the CA Tool Agent described in section 6.2.4.1. Its output is *Hotspots* (as for the *ca-tool-hotspot-service*) and the inputs are a time instant and a spatial extent.

6.2.7 Modeling Agent

The Modeling Agent maintains a prediction model and uses this model to service requests for predictions.

6.2.7.1 External Representation

The Modeling Agent is also represented as an executable process in the system and is invoked in a similar manner to the Tool Agent (see section 6.2.3). SWRL variables are used to represent input and output parameters for the prediction model. The agent is invoked by a *RequestPredictionAction* message, containing a *RequestPredictionAction* instance. The input parameters are specified as SWRL variables via the *hasInput* property and are used to specify the phenomenon to be modelled and optional spatial

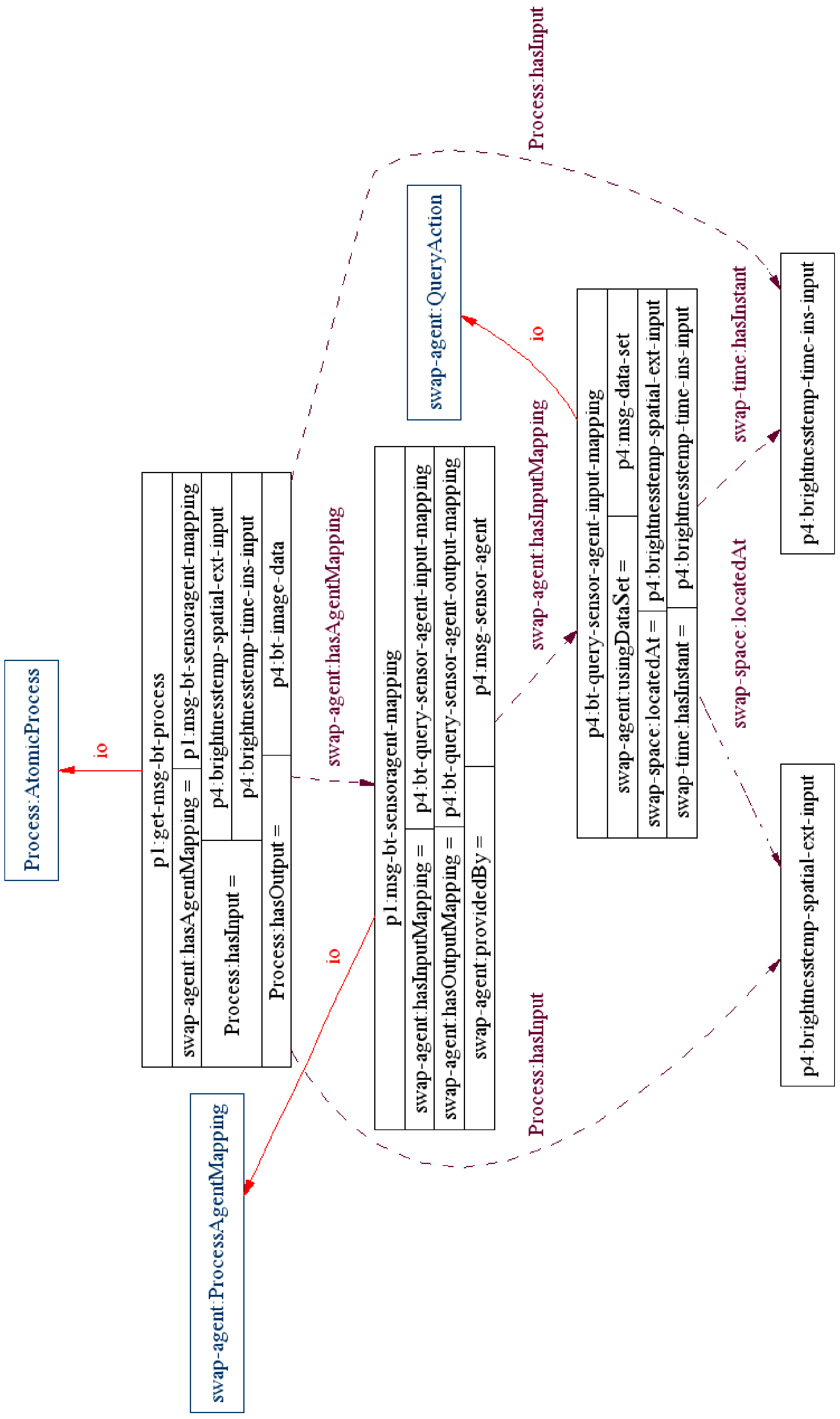

Process:AtomicProcess
io
p1:get-msg-bt-process
swap-agent:hasAgentMapping =
p1:msg-bt-sensoragent-mapping
Process:hasInput =
p4:brightnesstemp-spatial-ext-input
p4:brightnesstemp-time-ins-input
Process:hasOutput =
p4:bt-image-data
swap-agent:ProcessAgentMapping
io
swap-agent:hasAgentMapping
p1:msg-bt-sensoragent-mapping
swap-agent:hasInputMapping =
p4:bt-query-sensor-agent-input-mapping
swap-agent:hasOutputMapping =
p4:bt-query-sensor-agent-output-mapping
swap-agent:providedBy =
p4:msg-sensor-agent
swap-agent:QueryAction
io
Process:hasInput
swap-agent:hasInputMapping
Process:hasInput
p4:bt-query-sensor-agent-input-mapping
swap-agent:usingDataSet =
p4:msg-data-set
swap-space:locatedAt =
p4:brightnesstemp-spatial-ext-input
swap-time:hasInstant =
p4:brightnesstemp-time-ins-input
swap-time:hasInstant
swap-space:locatedAt
p4:brightnesstemp-time-ins-input
p4:brightnesstemp-spatial-ext-input

Figure 6.30: Process to Agent Mapping for querying the MSG Sensor Agent

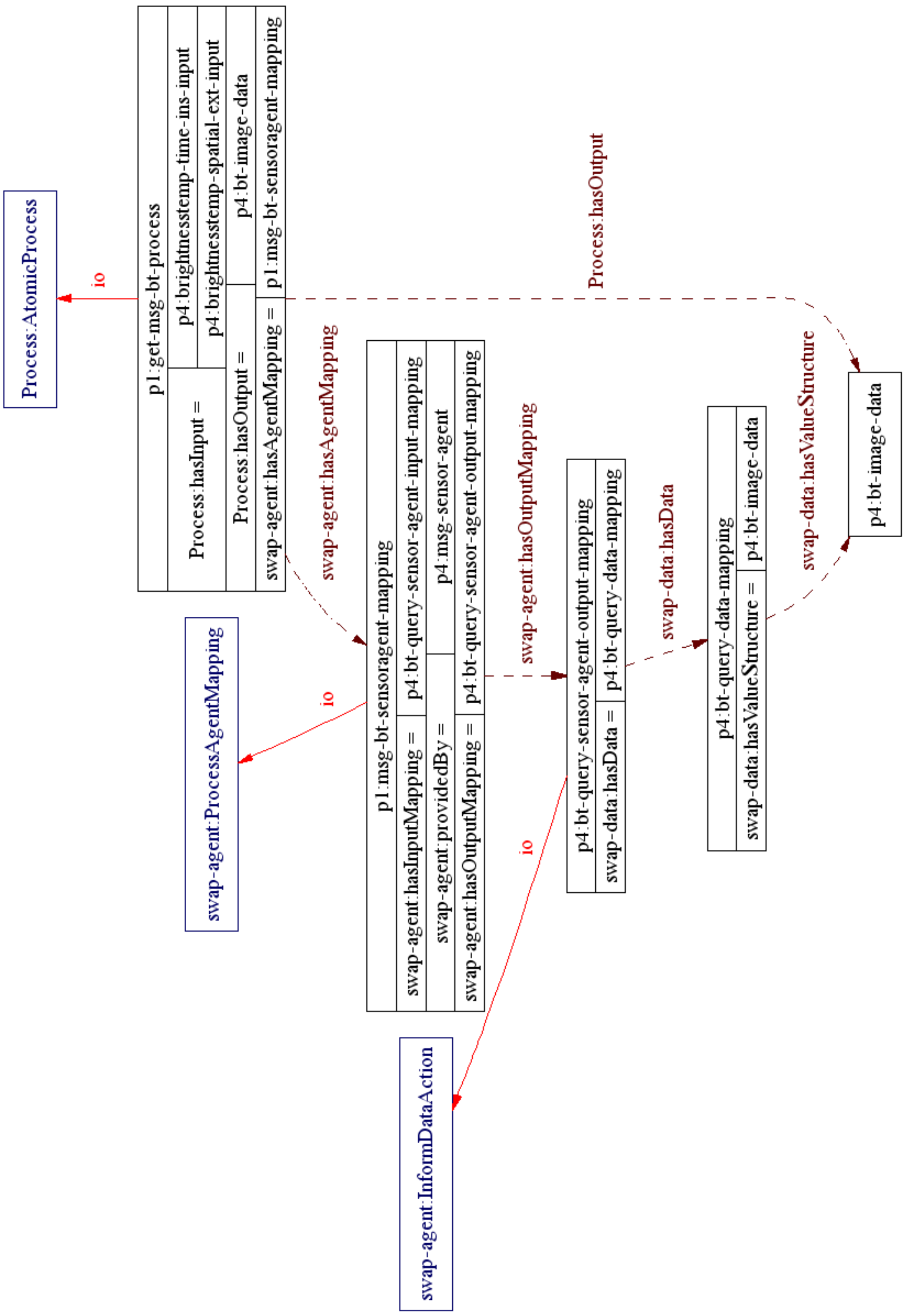

Process:AtomicProcess
io
p1:get-msg-bt-process
Process:hasInput =
p4:brightnesstemp-time-ins-input
p4:brightnesstemp-spatial-ext-input
Process:hasOutput =
p4:bt-image-data
swap-agent:hasAgentMapping =
p1:msg-bt-sensoragent-mapping
Process:hasOutput
swap-agent:hasAgentMapping
swap-agent:ProcessAgentMapping
io
p1:msg-bt-sensoragent-mapping
swap-agent:hasInputMapping =
p4:bt-query-sensor-agent-input-mapping
swap-agent:providedBy =
p4:msg-sensor-agent
swap-agent:hasOutputMapping =
p4:bt-query-sensor-agent-output-mapping
swap-agent:hasOutputMapping
io
swap-agent:InformDataAction
p4:bt-query-sensor-agent-output-mapping
swap-data:hasData =
p4:bt-query-data-mapping
swap-data:hasData
p4:bt-query-data-mapping
swap-data:hasValueStructure =
p4:bt-image-data
swap-data:hasValueStructure
p4:bt-image-data

Figure 6.31: Process to Agent Mapping for mapping the data response from the MSG Sensor Agent

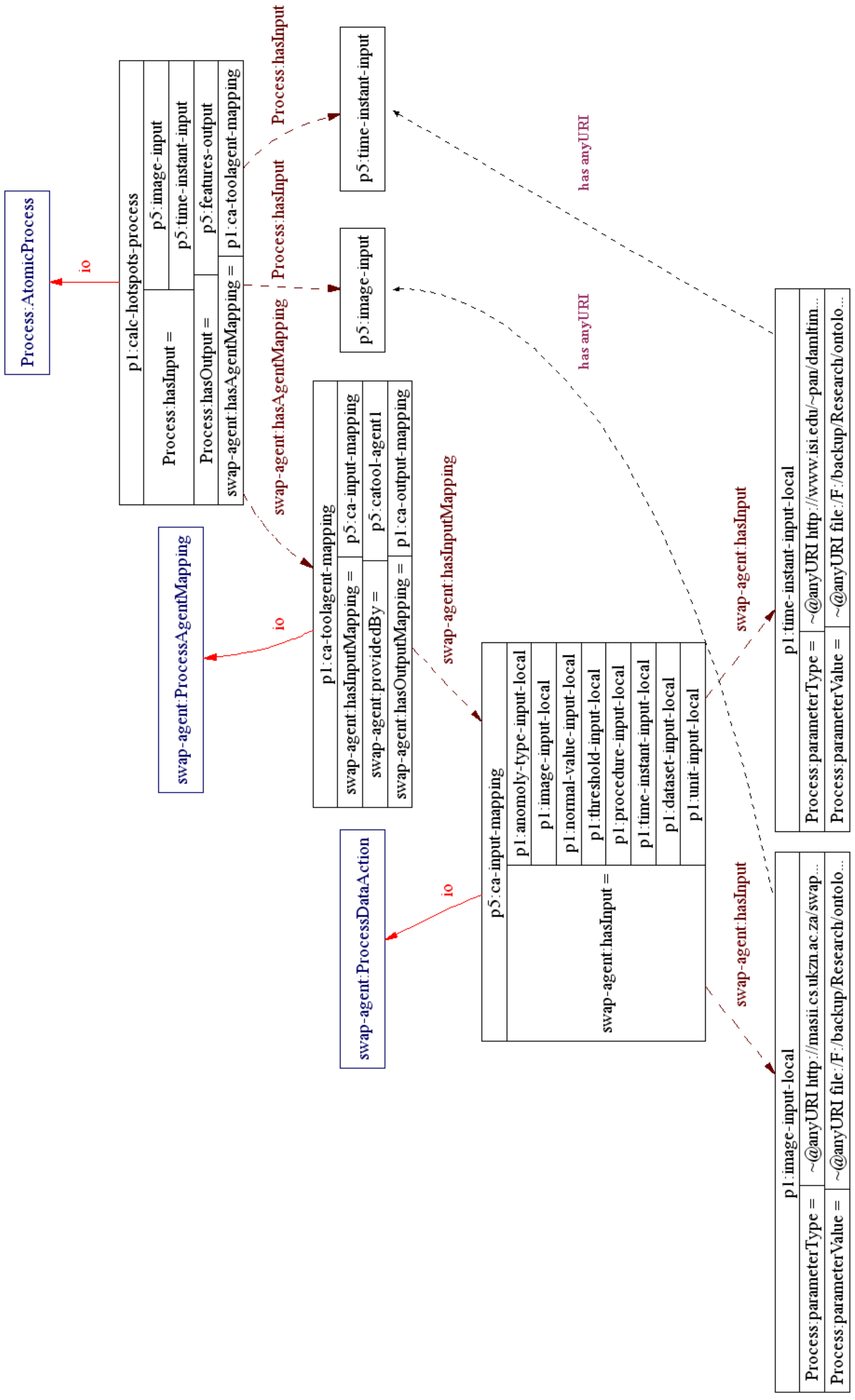


Figure 6.32: Process to Agent Mapping for a process request to the CA Tool Agent

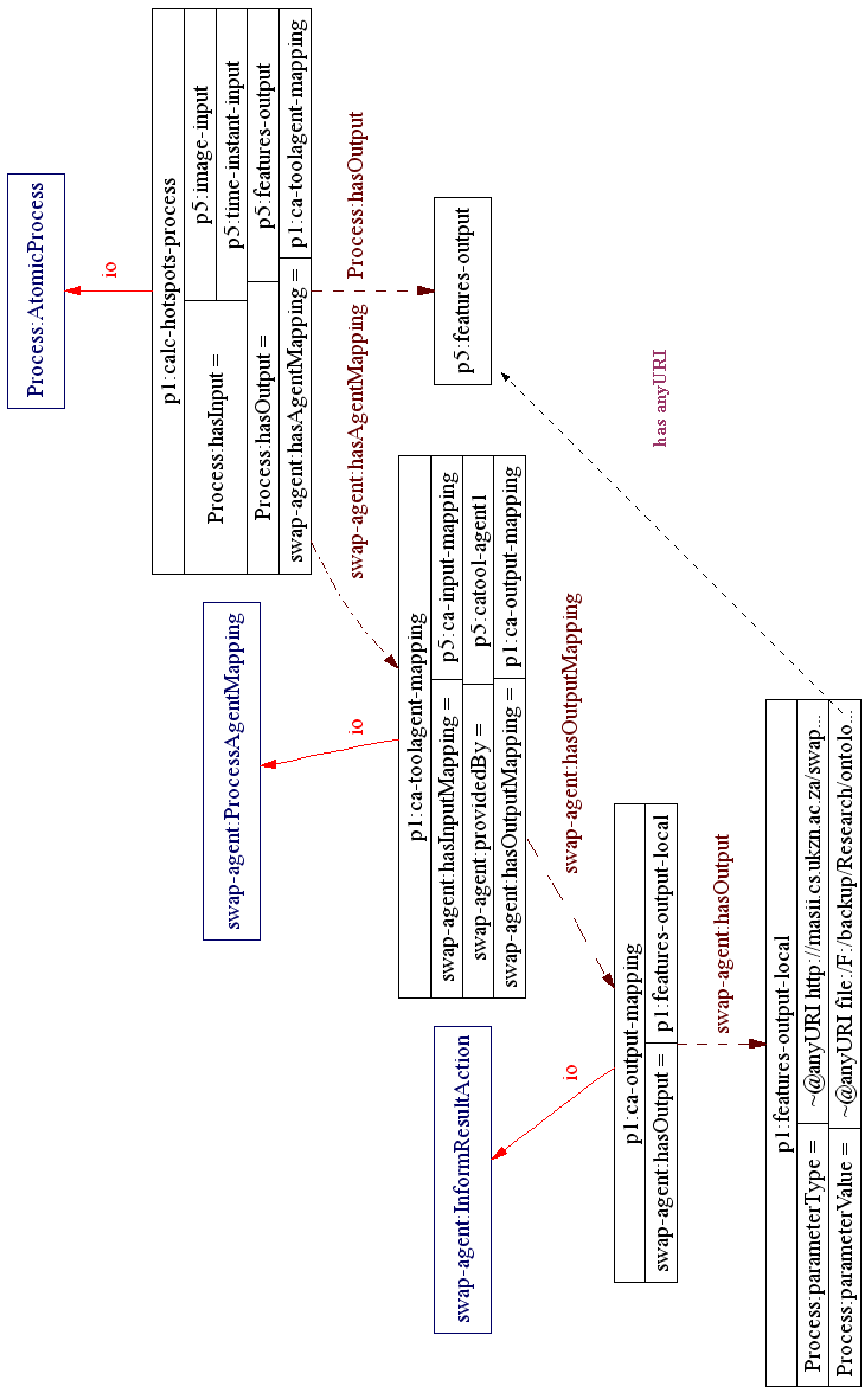


Figure 6.33: The Process to Agent Mapping for processing results from CA Tool Agent

Table 6.8: The Prediction Model Role schema

Role Schema:	Prediction Model
Description	Hosts a fire spread model and is able to predict, with an associated probability, the spread of the fire. Registers and provides a description of its capabilities with the Directory Agent.
Protocols and Activities	*PredictionRequest*, *DescribeCapabilities*, *SWAPServiceRegistration*
Permissions	Update capabilities in registry
Responsibilities	• Register capabilities with Directory Agent • Answer queries for capabilities • Respond to queries for fire spread predication • Maintain connections to appropriate data sources to update model

Table 6.9: The *PredictionRequest* Protocol schema

Protocol	Prediction Request
Initiator	Any agent but typically an Application Agent, or Workflow Agent
Participants	Modeling Agent, initiating agent
Inputs	Prediction inputs and criteria
Outputs	A prediction
Message Types	*RequestPredictionAction*, *InformPredictionAction*
Description	An agent can request for the Modeling Agent to make predictions about future events based on its internal model and supplied input data. The Modeling Agent responds with this prediction.

and temporal constraints, e.g. a future time and the location of interest. The agent responds with an *InformPredictionAction* message that contains the prediction values for this phenomenon, specified via the *hasOutput* property. The prediction model role and prediction request protocol that is used for requesting predictions are described in table 6.8 and table 6.9 respectively.

6.2.7.2 Internal Operation

A Modeling Agent's prediction model is represented using the Java *ModelingAdapter* interface shown in figure 6.34. The *ModelingAdapter* interface provides standard methods for interacting with a prediction model. Specific models are implemented as *ModelAdapter* classes. The *ModelAdapter* interface is designed similarly to the *ToolAdapter* interface described in section 6.2.3.2. It has a $setInput$ method that is used to set input parameters, a $process$ method that is executed after all inputs have been set, and a *getOutput* method for retrieving the values of output parameters. As with the *DataAdapter* and *ToolAdapter* interfaces, GeoAPI data structures are used for representing time, coverage and feature data values.

6.2.7.3 Capability Description

The Modeling Agent registers a *Service* instance with the SWAP Directory Agent that describes its functionality. The service description contained in the *isDescribedBy* property is an instance of the *Modeling* service class. It is similar to the *Tool* and *Workflow* service descriptions. It has two properties *hasInputType* and *hasOutputType* that specify the inputs and outputs of specific *Modeling* services. Other agents are able to search for *Modeling* service descriptions that have specific inputs and outputs.

6.2.8 The FireSpreadModeler Agent

The FireSpreadModeler Agent is a Modeling Agent used to predict the spread of wildfire. The agent was developed to illustrate the operation of the Modeling Agent in the system, more specifically how predictions are requested and communicated to agents. The current prediction model is a dummy model with mock data. A more realistic model would use current weather conditions such as: wind direction, wind speed, humidity and air temperature; and terrain characteristics such as slope and vegetation cover to determine fuel load.

The *FireSpreadModeler* class, shown in figure 6.34, is an implementation of the *ModelingAdapter* interface and is used internally to calculate wildfire spread. It takes as input a current wildfire and predicts the spread of this wildfire in the next hour, ten hours and one hundred hours. It accepts two inputs a wildfire feature that represents a current wildfire *wildfire-input-local* and the region of interest for predictions *location-input-local*. An example request message is shown in figure 6.35. It produces three output predictions, *wildfire-1hr-output-local*, *wildfire-10hr-output-local* and *wildfire-100hr-output-local*, that represent the wildfire prediction in 1 hour, 10 hours and 100 hours respectively (see figure 6.36).

To make predications the *FireSpreadModeler* should be able to query observations for current and future weather conditions at a specific area. The mechanism to retrieve this data is specified within the *FireSpreadModeler* class and does not affect the external operation of the agent. Various options

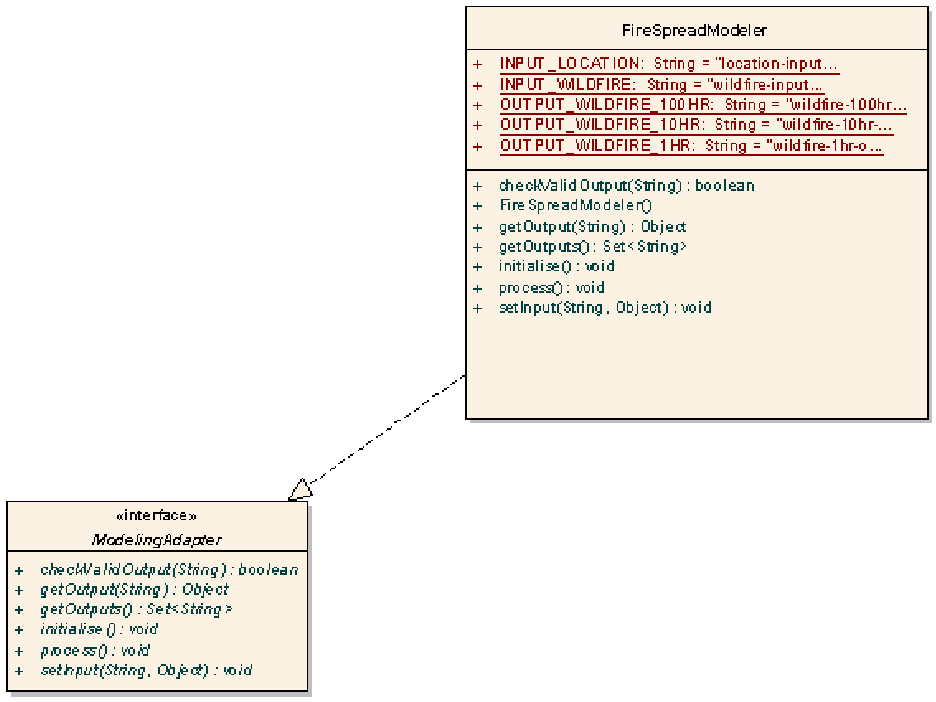

FireSpreadModeler
+ INPUT_LOCATION: String = "location-input...
+ INPUT_WILDFIRE: String = "wildfire-input...
+ OUTPUT_WILDFIRE_100HR: String = "wildfire-100hr...
+ OUTPUT_WILDFIRE_10HR: String = "wildfire-10hr-...
+ OUTPUT_WILDFIRE_1HR: String = "wildfire-1hr-o...
+ checkValidOutput(String) : boolean
+ FireSpreadModeler()
+ getOutput(String) : Object
+ getOutputs() : Set<String>
+ initialise() : void
+ process() : void
+ setInput(String, Object) : void
«interface»
ModelingAdapter
+ checkValidOutput(String) : boolean
+ getOutput(String) : Object
+ getOutputs() : Set<String>
+ initialise() : void
+ process() : void
+ setInput(String, Object) : void

Figure 6.34: Class diagram of the ModelingAdapter interface

are available. A Weather Sensor Agent can be deployed in the Sensor Layer which uses the *Weather-DataAdapter* shown in figure 6.14 to provide weather data, or it can query a weather Sensor Observation Service (SOS) directly using the OXFramework SOS client. Terrain and vegetation cover do not change as rapidly as weather observations. An agent serving terrain and vegetation cover data could be deployed in the Sensor Layer, or terrain and vegetation models could be provided locally and accessed from within the *FireSpreadModeler* class. It is easier to access a SOS or a local data store rather than deploy Sensor Agents to provide weather, terrain and vegetation cover data. However, by deploying Sensor Agents to provide this data, the data could be used within other SWAP applications.

6.2.8.1 Discovering the FireSpreadModeler Agent

The agent can be discovered in a similar manner as the CA Tool Agent described in section 6.2.4.1. The outputs are predicted wildfire features representing the wildfire spread at given times in the future, while the inputs are a current wildfire feature and a location of interest.

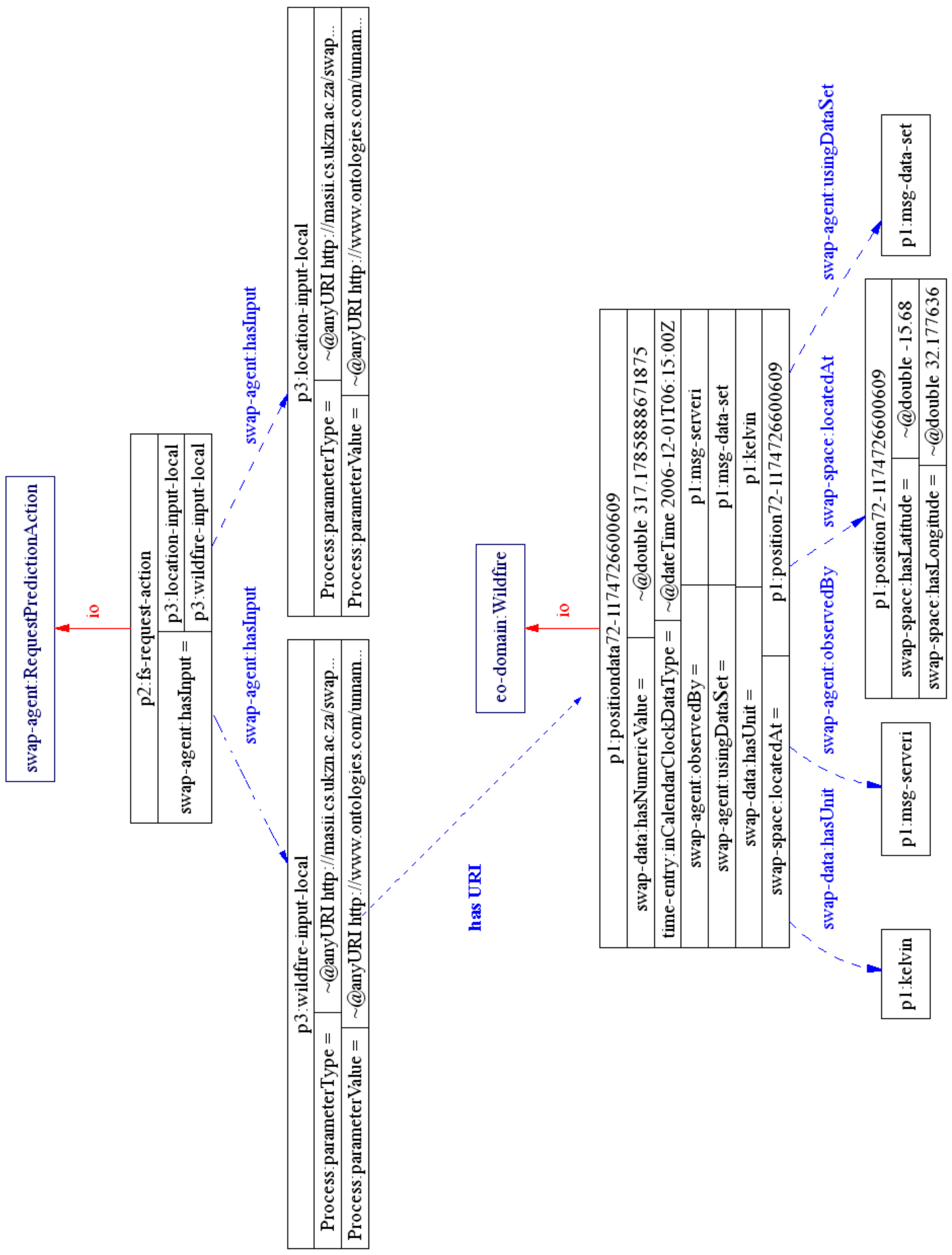

swap-agent:RequestPredictionAction
io
p2:fs-request-action
swap-agent:hasInput =
p3:location-input-local
p3:wildfire-input-local
swap-agent:hasInput
swap-agent:hasInput
p3:location-input-local
Process:parameterType = ~@anyURI http://masii.cs.ukzn.ac.za/swap...
Process:parameterValue = ~@anyURI http://www.ontologies.com/unnam...
p3:wildfire-input-local
Process:parameterType = ~@anyURI http://masii.cs.ukzn.ac.za/swap...
Process:parameterValue = ~@anyURI http://www.ontologies.com/unnam...
has URI
eo-domain:Wildfire
io
p1:positiondata72-1174726600609
swap-data:hasNumericValue = ~@double 317.178588671875
time-entry:inCalendarClockDataType = ~@dateTime 2006-12-01T06:15:00Z
swap-agent:observedBy = p1:msg-serveri
swap-agent:usingDataSet = p1:msg-data-set
swap-data:hasUnit = p1:kelvin
swap-space:locatedAt = p1:position72-1174726600609
swap-data:hasUnit
swap-agent:observedBy
swap-space:locatedAt
swap-agent:usingDataSet
p1:kelvin
p1:msg-serveri
p1:position72-1174726600609
swap-space:hasLatitude = ~@double -15.68
swap-space:hasLongitude = ~@double 32.177636
p1:msg-data-set

Figure 6.35: Fire spread modeling request message

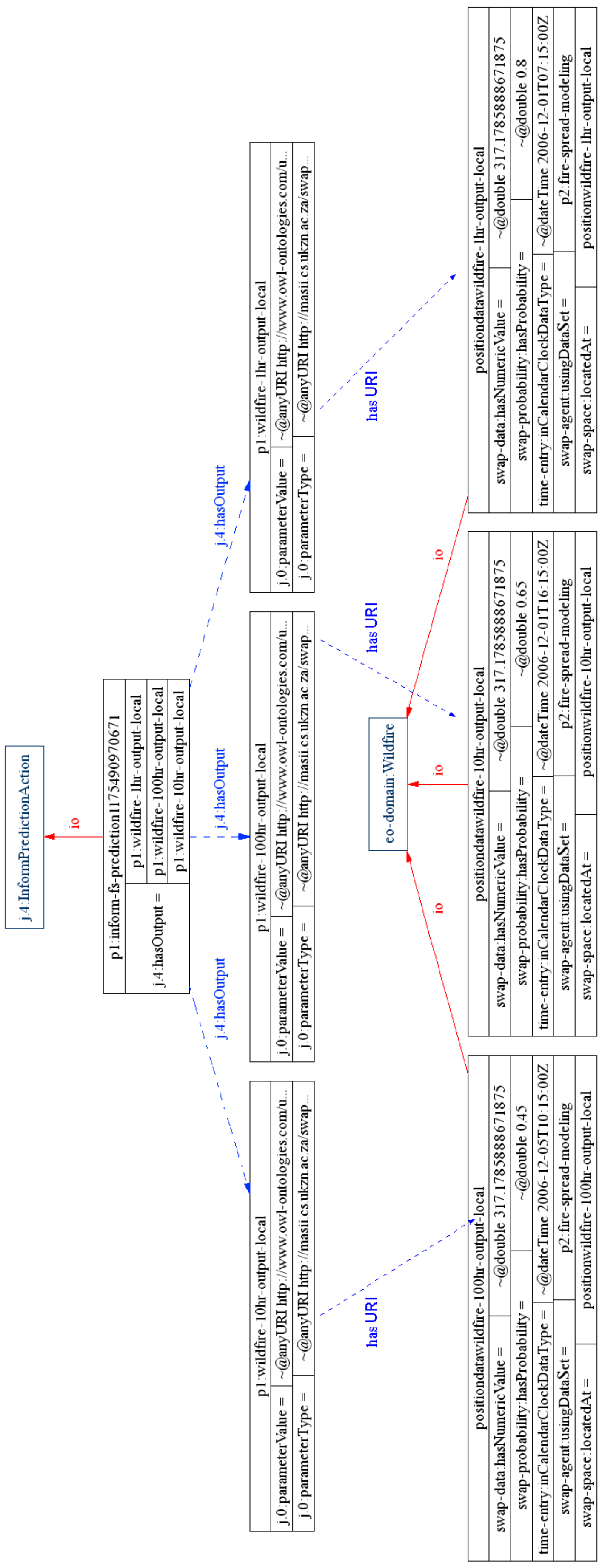

j.4:InformPredictionAction
io
p1:inform-fs-prediction1175490970671
j.4:hasOutput =
p1:wildfire-1hr-output-local
p1:wildfire-100hr-output-local
p1:wildfire-10hr-output-local
j.4:hasOutput
j.4:hasOutput
j.4:hasOutput
p1:wildfire-10hr-output-local
j.0:parameterValue =
~@anyURI http://www.owl-ontologies.com/u...
j.0:parameterType =
~@anyURI http://masii.cs.ukzn.ac.za/swap...
p1:wildfire-100hr-output-local
j.0:parameterValue =
~@anyURI http://www.owl-ontologies.com/u...
j.0:parameterType =
~@anyURI http://masii.cs.ukzn.ac.za/swap...
p1:wildfire-1hr-output-local
j.0:parameterValue =
~@anyURI http://www.owl-ontologies.com/u...
j.0:parameterType =
~@anyURI http://masii.cs.ukzn.ac.za/swap...
has URI
has URI
has URI
eo-domain:Wildfire
io
io
io
positiondatawildfire-100hr-output-local
swap-data:hasNumericValue =
~@double 317.178588867 1875
swap-probability:hasProbability =
~@double 0.45
time-entry:inCalendarClockDataType =
~@dateTime 2006-12-05T10:15:00Z
swap-agent:usingDataSet =
p2:fire-spread-modeling
swap-space:locatedAt =
positionwildfire-100hr-output-local
positiondatawildfire-10hr-output-local
swap-data:hasNumericValue =
~@double 317.178588867 1875
swap-probability:hasProbability =
~@double 0.65
time-entry:inCalendarClockDataType =
~@dateTime 2006-12-01T16:15:00Z
swap-agent:usingDataSet =
p2:fire-spread-modeling
swap-space:locatedAt =
positionwildfire-10hr-output-local
positiondatawildfire-1hr-output-local
swap-data:hasNumericValue =
~@double 317.178588867 1875
swap-probability:hasProbability =
~@double 0.8
time-entry:inCalendarClockDataType =
~@dateTime 2006-12-01T07:15:00Z
swap-agent:usingDataSet =
p2:fire-spread-modeling
swap-space:locatedAt =
positionwildfire-1hr-output-local

Figure 6.36: A fire spread modeling response message

6.2.9 Application Agent

The Application Agent provides end user Sensor Web applications in SWAP. End users interact with applications via their User Agents. The Application Agent provides an agent interface for User Agents to access and interact with Sensor Web applications. These applications provide higher level functionality that is abstracted from the complexity of the SWAP architecture and that is described by non technical higher level concepts that are more familiar to end users. Application Agents are not intended for reuse and are customised to offer specific end user applications. Currently, the SWAP Application Agent is designed to detect and store occurrences of phenomena that are of interest to end users. User Agents make requests to the Application Agent to be alerted whenever these phenomena are detected.

6.2.9.1 External Representation

The application provider role and the alert protocol that is used for requesting and receiving alerts are described in tables 6.10 and 6.11 respectively. The *Alert* protocol is used to request and receive alerts from the Application Agent. Alert requests are sent via *RequestAlertAction* messages. Various properties can be used to specify alert conditions. The *observesEntity* property is used to specify the phenomenon of interest. Users can specify spatial constraints, such as the region of interest by using any of the eight spatial relations discussed in section 4.4.2. For example, if raa is an instance of *RequestAlertAction* and roi is a *SpatialThing* with a location specified by the *locatedAt* property, then an alert request can be raa *intersects* roi. Temporal constraints may also be specified, such as whether current, past, or future alerts are required. The *swap-time* ontology provides concepts for $past$, $future$ and $present$. These values can be added to the request using the *inCalendarClock* property, e.g. raa *inCalendarClock* $future$ specifies requests for future alerts.

By default alerts are sent back to the agent that sent the *RequestAlertAction* message. However, the *recipient* property can be used to specify another agent to which the alerts should be sent. The Application Agent responds with *InformAlertAction* messages. An *InformAlertAction* instance has a single property, *hasAlert*, which has instances of *Alerts* as its range. An *Alert* instance represents an instance of the requested phenomenon with its appropriate properties such as location and observation time. An alert also has a *hasAlertRequest* property that contains the *RequestAlertAction* instance that triggered this alert.

Table 6.10: The Application role schema

Role Schema:	Application Provider
Description	Maintains an application data store, allows end users to specify alerts conditions on this data, and alerts users when these conditions are met. Registers and provides a description of its data and types of alerts that it supports.
Protocols and Activities	*Alert*, *DataRequest*, *WorkflowExecution*, *DescribeCapabilities*, *SWAPServiceRegistration*
Permissions	Update capabilities in registry
Responsibilities	• Register capabilities with Directory Agent • Answer requests for capabilities • Maintain application data store • Accepts and responds to alert requests on application data

Table 6.11: The *Alert* protocol schema

Protocol	Alert
Initiator	Any agent but typically a User Agent
Participants	Application Agent, initiating agent
Inputs	Alert Criteria
Outputs	Alerts corresponding to alert criteria
Message Types	*RequestAlertAction*, *InformAlertAction*
Description	An agent can request alerts from the Application Agent based on criteria specified in an alert request action. The Application Agent responds with alerts for phenomena that satisfy the alert criteria.

```
PROCEDURE InformResultHandler
BEGIN
    Add statements from current KB to past KB
    Delete all statements from current KB
    Add workflow results to current KB
    Run post workflow rules on current KB
    Set CHANGE flag to indicate change to current KB
END
```

Figure 6.37: Algorithm for processing incoming alerts at the Application Agent

6.2.9.2 Internal Operation

Internally the Application Agent has two key threads of operation, the first is to continuously update its application data store and the second is to maintain user alert conditions on its data store and to compose and send alert messages when alert conditions are satisfied.

The Application Agent integrates results from one or more Workflow Agents. Even though the Application Agent has the ability to directly invoke Tool, Modeling and even Sensor Agents, wherever possible the coordination of agent interactions and integration of data should be delegated to a Workflow Agent. The Application Agent retrieves the results of the Workflow Agent and applies a set of post processing rules which are used to filter and classify the results of the workflow as required for this application. This data is then integrated into the local knowledge base (KB). The data in the KB is split into two parts, the current KB and the past KB. The current KB contains the latest set of data received from the Workflow Agent, while the past KB contains all previous data received from the Workflow Agent. A *CHANGE* flag is set whenever the agent receives results from the Workflow Agent to signal to the alerting thread that the current KB has been modified. The algorithm for processing results from the Workflow Agent is shown in figure 6.37.

User agents register alert conditions with the Application Agent. Alert requests can be once off alert requests or persistent (future) alert requests, which are stored in a Persistent Alerts KB. When an alert request is received it is processed against the current KB. Thereafter, the alert request is either discarded if it is a once off request, or is stored as a persistent alert. By default all alert requests are assumed to be once off requests for current data. If a *RequestAlertAction* has an *inCalenderClock* property with a value of *future*, then the alert request is deemed to be a persistent alert and is added to the Persistent Alerts KB. A once of alert is processed immediately, while an alert thread runs each time the current KB is

```
PROCEDURE RequestAlertHandler(RequestAlertAction raa)
BEGIN
    ProcessAlert(raa)
    IF raa is a future alert request THEN
        Add raa to Persistent Alerts KB
    ENDIF
END

PROCEDURE ProcessAlert(raa)
    Create temp KB as a copy of current KB
    Add raa to tempKB
    Execute matching rules on temp KB
    Compose InformAlertAction instance and add matching alerts
    Create and send InformAlertAction message to requesting agent
END
```

Figure 6.38: Algorithm for processing alert requests at the Application Agent

```
PROCEDURE pAlertThread
BEGIN
 WHILE TRUE
     WHILE CHANGE flag not set
        sleep for t seconds
    END WHILE
    FOR each raa in Persistent Alerts KB
        ProcessAlert(raa)
    END FOR
    reset CHANGE flag
 END WHILE
END
```

Figure 6.39: Algorithm for processing persistent alerts at the Application Agent

updated to process persistent alerts. A set of matching rules are used to match alert requests with data in the current KB. When data in the current KB matches these alert conditions an *InformAlertAction* is composed and sent to the User Agent. The algorithm for processing alert requests is shown in figure 6.38.

Another thread within the Application Agent processes each *RequestAlertAction* instance in the Persistent Alert KB against the current KB whenever the CHANGE flag is set. The algorithm for the persistent alert thread is shown in figure 6.39

6.2.9.3 Capability Description

An Application Agent's service description is an instance of the *Application* service description. It contains the phenomenon which it detects as well as the spatial and temporal aspects of the phenomenon. In this way other Application or User Agents can search the SWAP Directory Agent for Application Agents that offer alerts for specific phenomena with specific temporal and spatial constraints.

6.2.10 The Wildfire Detection Application Agent

The Wildfire Detection Application Agent (WDAA) is an example of an Application Agent. It maintains a knowledge base of wildfires and responds to requests for wildfire alerts and spread predictions.

Temperature hotspots are considered to be potential wildfires. The WDAA delegates the detection of temperature hotspots to the Hotspot Detector Workflow Agent (HDWA). The FDDA sends an *Execute-Workflow* message to the HDWA to retrieve the latest temperature hotspots. Temperature hotspots are received via an *InformResultAction*, are classified as wildfires and are stored in the current knowledge base. The classification is carried out using a single logic rule:

```
(?x rdf:type eod:Hotspot) ->  (?x rdf:type eod:Wildfire)
```

This rule classifies all *Hotspots* instances as instances of *Wildfire*. Additional filtering rules could also be specified, e.g. rules to remove known hotspots that are not wildfires to reduce false alerts.

An alert request is made via a *RequestAlertAction* message. An example of a *RequestAlertAction* instance as shown in figure 6.40. The *observesEntity* property specifies the phenomena of interest, in this case *wildfire*. A spatial constraints is specified by the *intersects* property that has a location value of *substation-1*. This alert request also specifies two optional properties. The *recipient* property specifies that all alerts must be sent to *user-agent1*, and the value of *future* (*inCalendarClock* property) specifies that this is not a once off request, but a persistent request for all current and future wildfires. The WDAA also handles requests for wildfire spread. Wildfire request alerts that specify a *providePredictionsFor-Location* property triggers fire spread requests for the specified location value.

On receiving an alert request, the WDAA matches all wildfires that intersect with the geometry of *substation-1*. The two rules shown in figure 6.41 are used to match alert requests to wildfires. The first

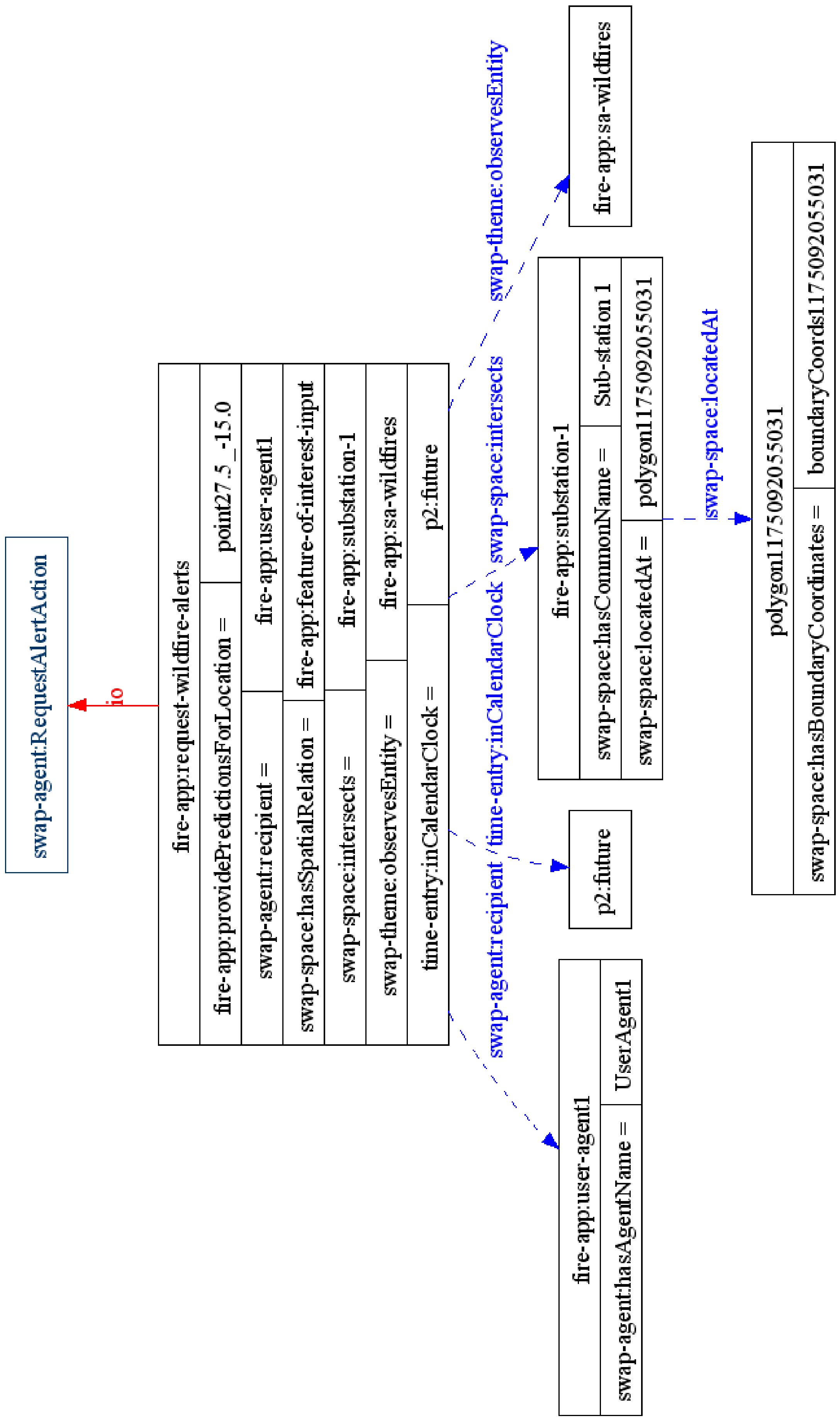

swap-agent:RequestAlertAction
io
fire-app:request-wildfire-alerts
fire-app:providePredictionsForLocation =
point27.5_-15.0
swap-agent:recipient =
fire-app:user-agent1
swap-space:hasSpatialRelation =
fire-app:feature-of-interest-input
swap-space:intersects =
fire-app:substation-1
swap-theme:observesEntity =
fire-app:sa-wildfires
time-entry:inCalendarClock =
p2:future
swap-theme:observesEntity
swap-space:intersects
time-entry:inCalendarClock
swap-agent:recipient
fire-app:sa-wildfires
fire-app:substation-1
swap-space:hasCommonName =
Sub-station 1
swap-space:locatedAt =
polygon1175092055031
swap-space:locatedAt
p2:future
polygon1175092055031
swap-space:hasBoundaryCoordinates =
boundaryCoords1175092055031
fire-app:user-agent1
swap-agent:hasAgentName =
UserAgent1

Figure 6.40: An alert request for wildfires

```
(?wf rdf:type agent:Alert) <-
    (?ra rdf:type agent:RequestAlertAction) (?ra ?spc_related ?foi)
    (?foi rdf:type spc:SpatialThing)
    (?spc_related rdfs:subPropertyOf spc:hasSpatialRelation)
    (?wf rdf:type eod:Wildfire) (?foi ?spc_related ?wf).
(?wf agent:hasAlertRequest ?ra) <-
    (?ra rdf:type agent:RequestAlertAction) (?ra ?spc_related ?foi)
    (?foi rdf:type spc:SpatialThing)
    (?spc_related rdfs:subPropertyOf spc:hasSpatialRelation)
    (?wf rdf:type eod:Wildfire) (?foi ?spc_related ?wf).
```

Figure 6.41: Rules for matching wildfires to alert requests

rule matches wildfires to alert requests and classifies these wildfires as *Alerts*, while the second rule adds the *hasAlertRequest* property to each alert to map the alert to the *RequestAlertAction* that triggered the alert. The WDAA responds with an *InformAlertAction* instance that contains the matching wildfires. An example of an *InformAlertAction* instance containing two wildfire alerts in the *hasAlert* property is shown in figure 6.42. Each alert contains the properties for the wildfire as well as the alert request that triggered these alerts. In this instance the alerts were triggered by the *RequestAlertAction* described above and shown in figure 6.40. Since the *providePredictionsForLocation* property is set in the request, a fire spread prediction request together with the specified spatial location is sent to the Fire Spread Modeling Agent. The *Alert* instance contains spread prediction values representing the spread predictions (1 hour, 10 hour and 100 hour) for this wildfire in the direction of the requested spatial location. In this way the WDAA is also able to provide spread predictions for detected wildfires.

6.2.10.1 Discovering the Wildfire Detection Application Agent

The WDAA registers an instance of the *Application* service description as shown in figure 6.43. The phenomenon being detected is an instance of the wildfire phenomena. As the wildfire detection process is based on the MSG data set, the spatial and temporal aspects of the service is similar to the MSG data set. The spatial extent is the spatial extent of South Africa, the temporal resolution is 15 minutes, and the detection process uses the Hotspot Detection Workflow Agent.

6.2.11 User Agent

The User Agent (UA) represents the end user in the system and allows end users to interact with SWAP applications provided by Application Agents.

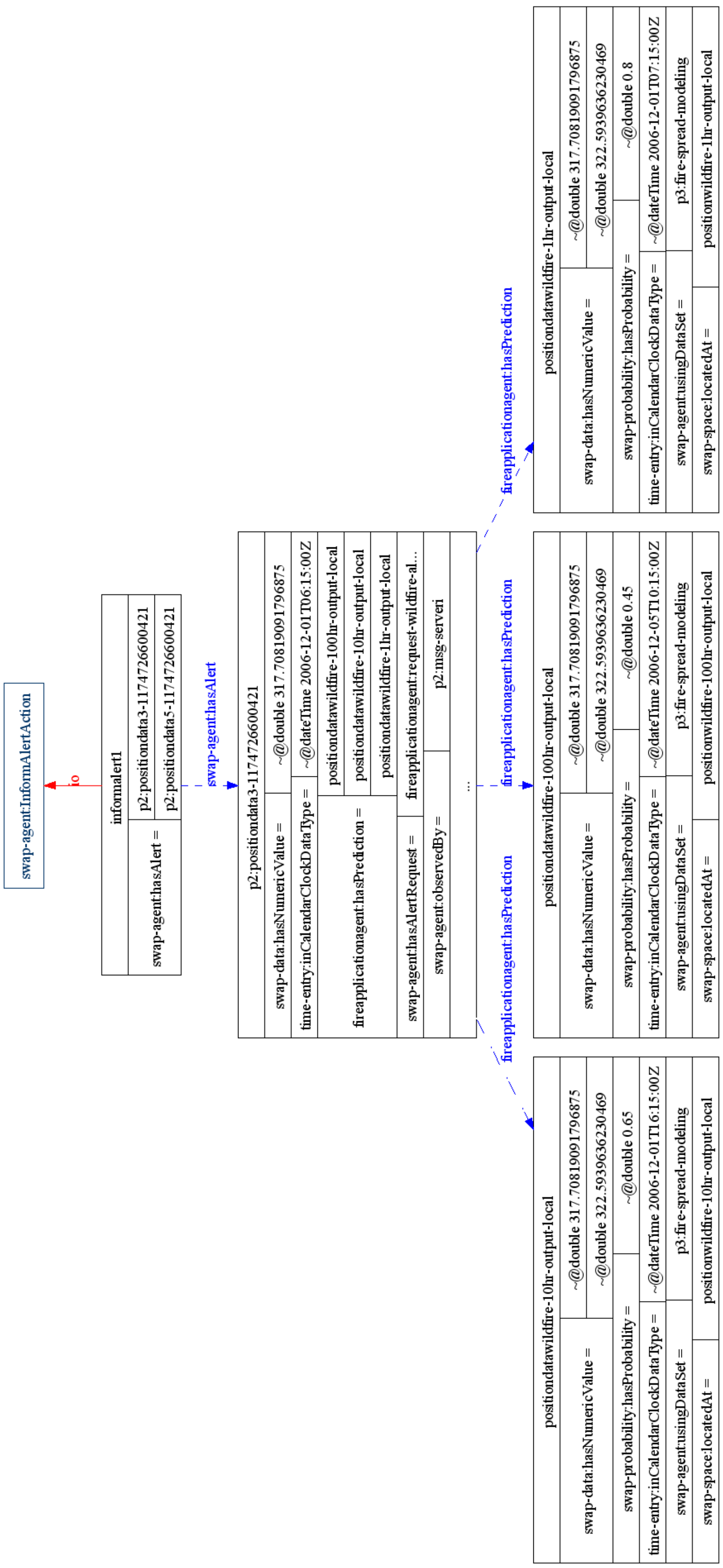

swap-agent:InformAlertAction
io
informalert1
swap-agent:hasAlert =
p2:positiondata3-1174726600421
p2:positiondata5-1174726600421
swap-agent:hasAlert
p2:positiondata3-1174726600421
swap-data:hasNumericValue =
~@double 317.70819091796875
time-entry:inCalendarClockDataType =
~@dateTime 2006-12-01T06:15:00Z
fireapplicationagent:hasPrediction =
positiondatawildfire-100hr-output-local
positiondatawildfire-10hr-output-local
positiondatawildfire-1hr-output-local
swap-agent:hasAlertRequest =
fireapplicationagent:request-wildfire-al...
swap-agent:observedBy =
p2:msg-serveri
...
fireapplicationagent:hasPrediction
fireapplicationagent:hasPrediction
fireapplicationagent:hasPrediction
positiondatawildfire-1hr-output-local
swap-data:hasNumericValue =
~@double 317.70819091796875
~@double 322.5939636230469
swap-probability:hasProbability =
~@double 0.8
time-entry:inCalendarClockDataType =
~@dateTime 2006-12-01T07:15:00Z
swap-agent:usingDataSet =
p3:fire-spread-modeling
swap-space:locatedAt =
positionwildfire-1hr-output-local
positiondatawildfire-100hr-output-local
swap-data:hasNumericValue =
~@double 317.70819091796875
~@double 322.5939636230469
swap-probability:hasProbability =
~@double 0.45
time-entry:inCalendarClockDataType =
~@dateTime 2006-12-05T10:15:00Z
swap-agent:usingDataSet =
p3:fire-spread-modeling
swap-space:locatedAt =
positionwildfire-100hr-output-local
positiondatawildfire-10hr-output-local
swap-data:hasNumericValue =
~@double 317.70819091796875
~@double 322.5939636230469
swap-probability:hasProbability =
~@double 0.65
time-entry:inCalendarClockDataType =
~@dateTime 2006-12-01T16:15:00Z
swap-agent:usingDataSet =
p3:fire-spread-modeling
swap-space:locatedAt =
positionwildfire-10hr-output-local

Figure 6.42: An alert response for wildfires

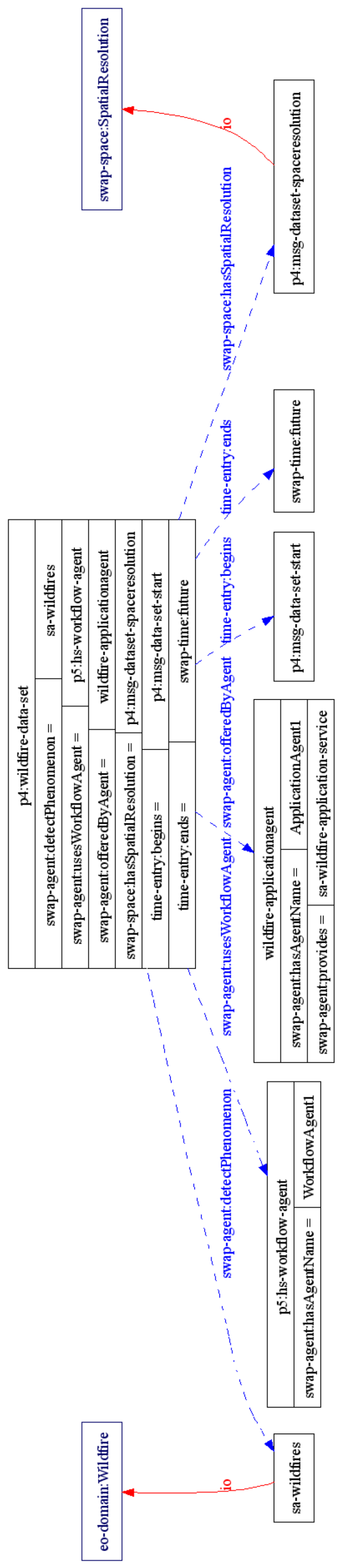

eo-domain:Wildfire
sa-wildfires
io
swap-agent:detectPhenomenon
p5:hs-workflow-agent
swap-agent:hasAgentName =
WorkflowAgent1
swap-agent:usesWorkflowAgent
p4:wildfire-data-set
swap-agent:detectPhenomenon =
sa-wildfires
swap-agent:usesWorkflowAgent =
p5:hs-workflow-agent
swap-agent:offeredByAgent =
wildfire-applicationagent
swap-space:hasSpatialResolution =
p4:msg-dataset-spaceresolution
time-entry:begins =
p4:msg-data-set-start
time-entry:ends =
swap-time:future
wildfire-applicationagent
swap-agent:hasAgentName =
ApplicationAgent1
swap-agent:provides =
sa-wildfire-application-service
swap-agent:offeredByAgent
time-entry:begins
p4:msg-data-set-start
time-entry:ends
swap-time:future
swap-space:hasSpatialResolution
p4:msg-dataset-spaceresolution
io
swap-space:SpatialResolution

Figure 6.43: The service description of the Wildfire Detection Application Agent

6.2.11.1 External representation

UAs do not offer a service but are used to access the alerting service provided by Application Agents. The UA provides a user interface for the end user to specify alert requests and to visualise received alerts. For each alert request a *RequestAlertAction* message is composed and sent to the relevant Application Agent. Application Agents respond with *InformAlertAction* messages containing alert instances that satisfy these requests.

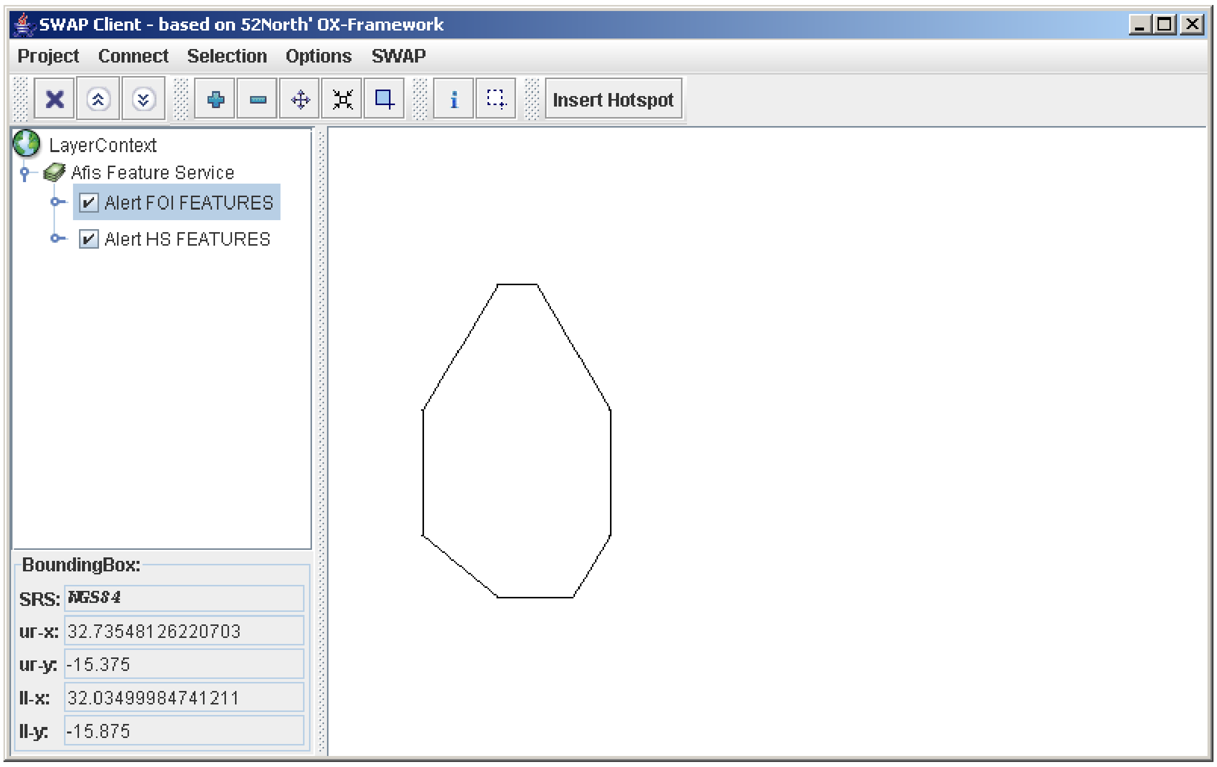
6.2.11.2 Internal operation

The User Agent provides graphical user interfaces (GUI) for specifying alert requests and for data visualisation, i.e. to visualise received alerts and how these alerts relate to user defined features. The Java Swing components from the OXFramework (OXF) client is used for the data visualisation GUI.

All features and alert instances are stored in a local knowledge base. The OXF Client is started within the agent environment and is passed a reference to the agent's knowledge base. The OXF client connects to the local KB and extracts and displays features of interest and alerts from the KB. The data mapping API is used to transform ontology data instances to OXFramework features. Alerts which are represented as single values, with a location value, are first converted to a Java Collection of OXFFeatures which is then displayed in the OXF Client. There is a mechanism to update the GUI when new alerts are received by the User Agent. When an *InformAlertAction* message is received, alert instances contained in the message are inserted into the local KB. The OXF client has a listener registered with the Jena knowledge base and receives a change event when any changes occur in the knowledge base. Thus, when new alerts are received and added to the knowledge base, the client is immediately notified and refreshes the alert layer with the new alerts.

6.2.12 The Wildfire Detection User Agent

The Wildfire Detection User Agent (WDUA) allows an end user to request, receive and visualise alerts for wildfires. A request for wildfire alerts is made by sending the Wildfire Detection Application Agent (WDAA) a *RequestAlertAction* message (shown in figure 6.40) specifying the spatial relation and a feature of interest. The WDAA responds with an *InformAlertAction* message containing wildfire instances



SRS: WGS84
ur-x: 32.735481262207O3
ur-y: -15.375
ll-x: 32.03499984741211
ll-y: -15.875

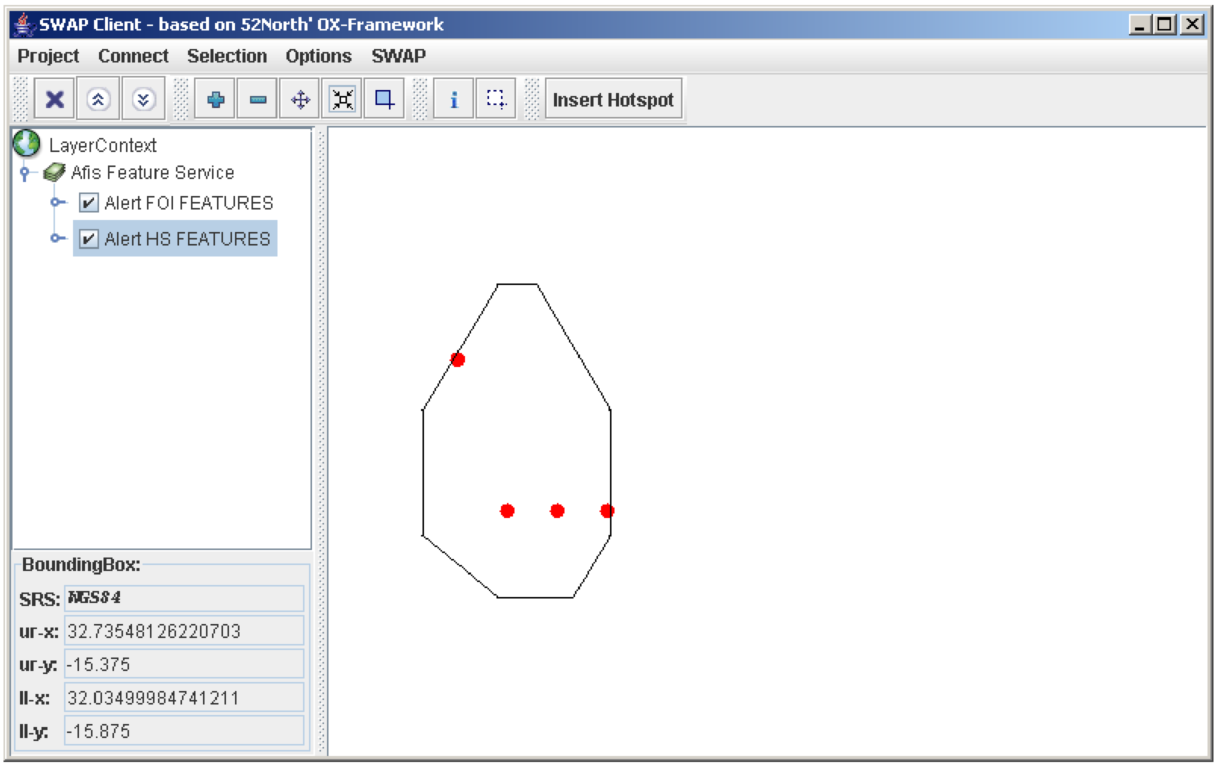
Figure 6.44: Visualising features of interest using the OXFramework client

that satisfy the request. Figure 6.44 shows the OXF Client with a single feature representing the feature of interest (*substation-1* in this case). A *RequestAlertMessage* is sent to the WDAA for all wildfires that intersects with *substation-1*. Four wildfires are detected in this region and are sent to the User Agent. These wildfire instances are inserted into the local knowledge base. The OXF Client receives a change event and refreshes its GUI and displays the wildfire alerts as shown in figure 6.45.

6.3 DEPLOYING THE WILDFIRE DETECTION APPLICATION

The wildfire detection application can be deployed to a User Agent using a MASII adapter as described in section 3.4.

A basic MASII User Agent is installed on the user's machine. As shown in figure 6.46, only the deployment protocol and service adapter required to retrieve and install adapters from the Adapter Agent is installed on the User Agent. The application catalogue contains a single application, the Wildfire Detection application (see figure 6.47). Besides an entry in the application catalogue, the User Agent has no capabilities to access the wildfire detection application.


SWAP Client - based on 52North' OX-Framework
Project Connect Selection Options SWAP
Insert Hotspot
LayerContext
Afis Feature Service
Alert FOI FEATURES
Alert HS FEATURES

BoundingBox:

SRS: WGS84

ur-x: 32.73548126220703

ur-y: -15.375

ll-x: 32.03499984741211

ll-y: -15.875

Figure 6.45: Visualising wildfire alerts using the OXFramework client

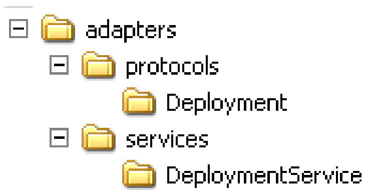


Figure 6.46: The adapter store of the User Agent before installing the wildfire detection adapter

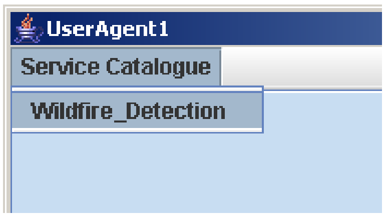
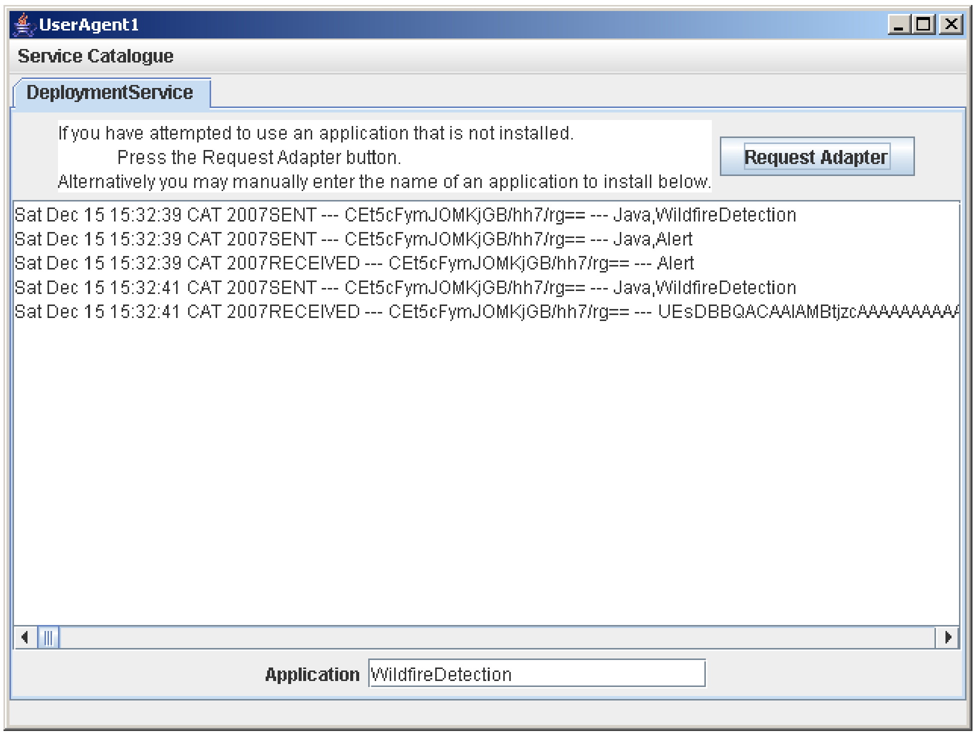


Figure 6.47: A screenshot of a User Agent before installing the wildfire detection application adapter



If you have attempted to use an application that is not installed.
Press the Request Adapter button.
Alternatively you may manually enter the name of an application to install below.

```
Sat Dec 15 15:32:39 CAT 2007SENT --- CEt5cFymJOMKjGB/hh7/rg== --- Java,WildfireDetection
Sat Dec 15 15:32:39 CAT 2007SENT --- CEt5cFymJOMKjGB/hh7/rg== --- Java,Alert
Sat Dec 15 15:32:39 CAT 2007RECEIVED --- CEt5cFymJOMKjGB/hh7/rg== --- Alert
Sat Dec 15 15:32:41 CAT 2007SENT --- CEt5cFymJOMKjGB/hh7/rg== --- Java,WildfireDetection
Sat Dec 15 15:32:41 CAT 2007RECEIVED --- CEt5cFymJOMKjGB/hh7/rg== --- UEsDBBQACAAIAMBtjzcAAAAAAAAA
```

Application: WildfireDetection

Figure 6.48: Downloading and installing the adapter and protocol required for the wildfire detection application

The user selects the wildfire detection application in the application catalogue (see figure 6.47). The User Agent checks its adapter store for this adapter. Since it is not present, the adapter and the required protocols, in this case the *WildfireDetection* adapter and the *Alert* protocol, are downloaded and installed locally at the User Agent as shown in figure 6.48.

The adapter folder, after the *WildfireDetection* adapter and the *Alert* protocol are installed, is shown in figure 6.49. The adapter is downloaded as a single Java jar file into the *dist* folder. This file is extracted into the *WildfireDetection* folder which creates a *lib* folder containing the libraries and an *ontologies* folder containing the ontologies for this application.

The adapter is dynamically activated and the User has immediate access to the wildfire detection application as shown in figures 6.44 and 6.45.

6.4 CASE STUDY 2: MONITORING INFORMAL SETTLEMENTS

This section describes the design and implementation of a second SWAP application, for detecting and monitoring informal settlements. This application is more complex than the application on wildfire

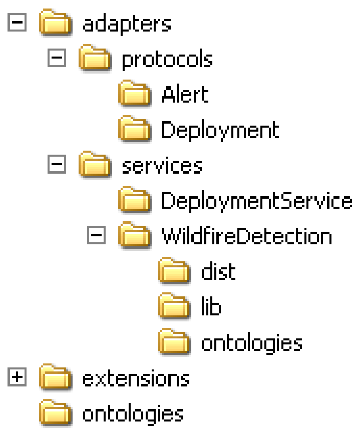


Figure 6.49: The adapter store of the User Agent after installing the wildfire detection

detection. It uses five processing steps (Tool Agents) as opposed to a single processing step (Tool Agent) in the wildfire detection application. The application also incorporates a supervised machine learning algorithm to perform classification of informal settlements.

The design and development of the application was undertaken as a Computer Science Masters research project under the author's supervision [156].

6.4.1 Application overview

The apartheid system was dismantled in South Africa in 1994. This resulted in a mass migration of mostly black South Africans from rural to urban areas in the hope of seeking employment and to gain access to the first world infrastructure that was common to major South African cities. Informal settlements [5] became a common occurrence in urban areas as a low cost accommodation option.

The Informal Settlement Information System (ISIS) aims to detect different types of informal settlements from high resolution satellite imagery. The current prototype detects informal settlements specifically within the Alexandra region in South African from high resolution Quickbird imagery that are produced every three days. The design allows for the incorporation of SPOT images that have a lower resolution but that are produced daily. The application is similar to the wildfire detection application in that both extract information from remotely sensed imagery. However, the wildfire and informal settlement domain and the type of imagery used differ widely.

[5]Informal settlements are areas where groups of housing units have been constructed on land to which the the occupants have no legal claim

Table 6.12: ISIS application components

Function	Component
Data provider	Quickbird, SPOT
Segmentation	Tiling
Feature Extraction	Edge, NDVI, image statistics
Classification	Naive Bayes classifier

Table 6.13: ISIS agent abstractions

Component	SWAP agent type
Quickbird, SPOT	Sensor Agents
Tiling, Edge, NDVI, image statistics	Tool Agents
Coordinating image processing steps	Workflow Agent
Naive Bayes classifier	Application Agent

6.4.2 Design

The first step in the design process was to identify and prototype the required data providers and image processing components. These components are listed in table 6.12. Corresponding SWAP agents (table 6.13) were then identified to host these components, which resulted in the architecture shown in figure 6.50.

At the Sensor Layer, QuickBird image data is offered by the Quickbird Sensor Agent. At the Knowledge Layer, the ISIS [6] Workflow Agent retrieves the QuickBird data from the Quickbird Sensor Agent and passes it to a series of Tool Agents that segment and extract appropriate features from the data. The Dimensions and Tiling Tool Agents break the image into a series of tiles. These tiles are then passed through the Edge and NDVI Tool Agents, which extract edges and the vegetation index [7] from each tile respectively. The Edge and NDVI images for each tile are then passed through the Stats Tool Agent, which extracts aggregate statistics from these images. This includes mean, standard deviation, density, skewness and kurtosis. The statistics for the NDVI and edges for each tile are then passed to the ISIS Application Agent (AA) for classification. A BayesNet classifier, from the WEKA [11] toolkit, is used within the AA to perform classification. The classifier is trained using historical data to detect informal settlements from Edge and NDVI image statistics. The ISIS AA uses the detected informal settlements to respond to informal settlements alert queries from ISIS User Agents.

[6]Informal Settlement Information System

[7]The Near Difference Vegetation Index (NDVI) assesses the percentage of vegetation in an image

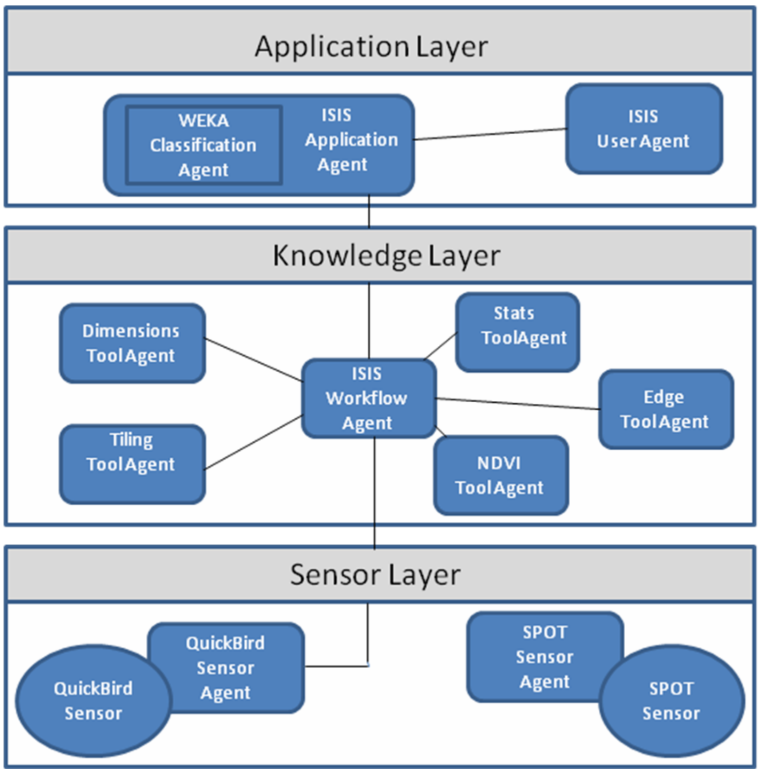


Figure 6.50: The ISIS architecture [156]

6.4.2.1 Ontology development

The SWAP technical and conceptual ontologies were extended to create the necessary domain and agent ontologies. The domain ontologies provide concepts to describe informal settlements and the surrounding infrastructure including the road network (tar road or dirt road) the building material from which homes are constructed (brick or corrugated tin), and the communications facility present (Internet or telephone lines).

Eight agent ontologies, one for each of the agents listed in table 6.13, were created. The agent ontologies are based on the ontologies created for the agents in the wildfire detection application. In most cases only minor modifications were made to the wildfire agent ontologies. Each agent ontology contains a description of the service offered by the agent and template messages to invoke and interpret the response of the service. The *DataSet*, *Tool*, *Workflow* and *Application* service descriptions used in the wildfire application (see section 6.2) were used to describe the services offered by the Sensor, Tool, Workflow and Application Agents respectively. Existing protocol and message types were reused. No new protocol or message types were required. Five *DataProcessing* protocols (see section 6.2.3) for

Table 6.14: Spatial and temporal resolution of Quickbird and SPOT image data [156]

	QuickBird data	SPOT data
Bands with their respective spatial resolutions	Panchromatic Band (0.60m)	Panchromatic Band (2.44m)
	Multispectral Band (2m) 1. Blue 2. Green 3. Red 4. NIR	Multispectral Band (10m) 1. Green 2. Red 3. NIR 4. MIR
Revisit time	1-3.5 days	daily

the Tool services, four *DataRequest* protocols (see section 6.2.1) and one *WorkflowExecution* protocol (see section 6.2.5) were created. The request and response message templates were also based on those specified for the agents in the wildfire detection application. An OWL-S workflow, shown in figure 6.51, was created to coordinate agent execution at the Knowledge Layer.

6.4.3 Implementation

This phase involved implementing the different agents identified in table 6.13. As no new message types or interaction protocols were introduced, the existing message handlers were used with minor adjustments. The different agent implementations from the wildfire detection application were also reused with minor adjustments. The bulk of the implementation phase was spent on developing and testing the two data adapters and the five tool adapters for the Sensor and Tool Agents respectively. The developer also spent a considerable time testing and modifying the agent and workflow ontologies in order to achieve the required individual agent behaviour. Brief descriptions of the operation of the agents are given below.

6.4.3.1 The Quickbird and SPOT Sensor Agents

The QuickBird Sensor Agent and the SPOT Sensor Agent provide sensor data captured by the QuickBird and SPOT-5 satellites respectively over the Alexandra area. Both sensors capture two images, a Panchromatic image and a Multispectral image. The spatial resolution and revisit time for these image bands are shown in table 6.14.

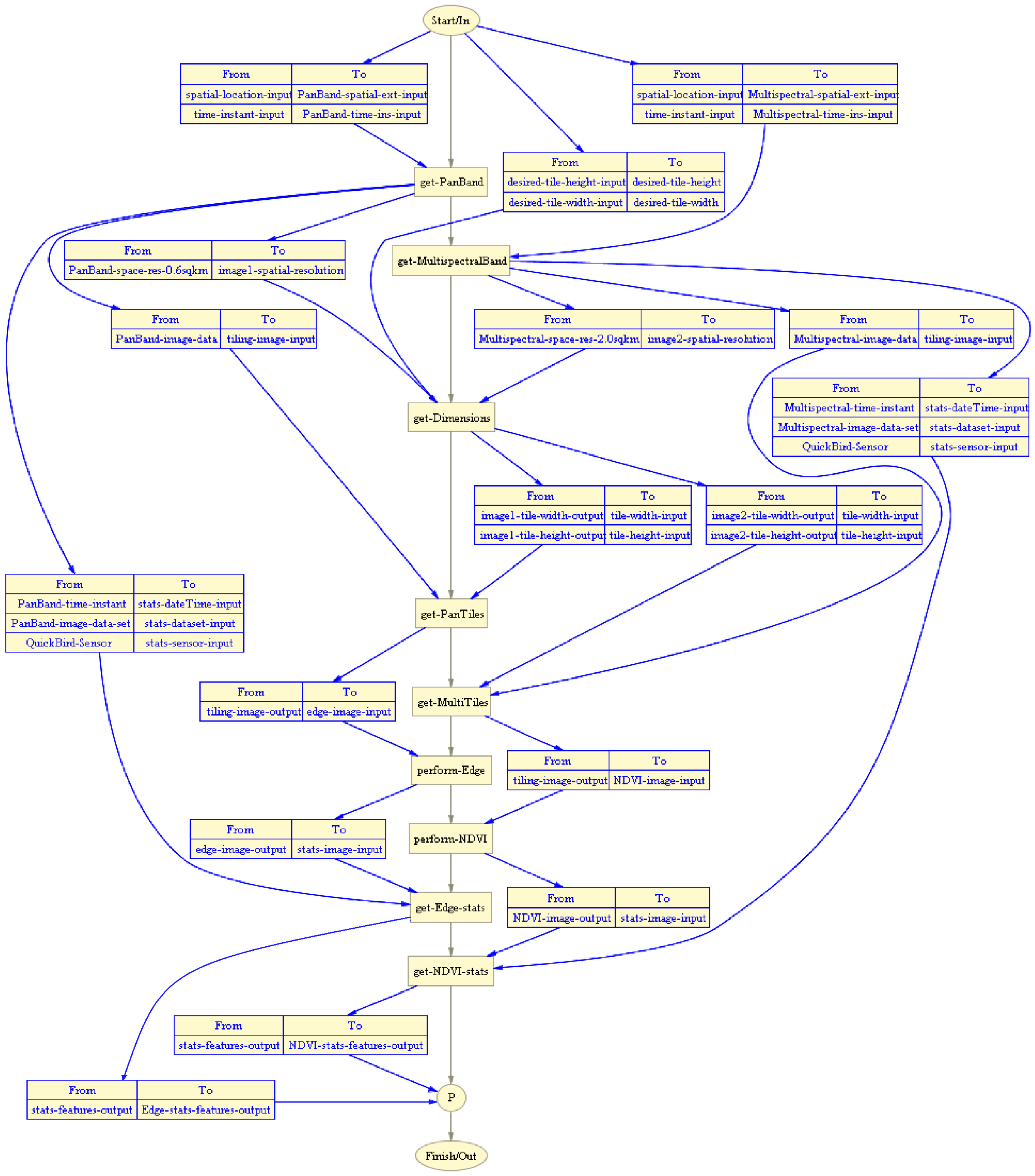

Start/In
From | To
spatial-location-input | PanBand-spatial-ext-input
time-instant-input | PanBand-time-ins-input
From | To
spatial-location-input | Multispectral-spatial-ext-input
time-instant-input | Multispectral-time-ins-input
get-PanBand
From | To
desired-tile-height-input | desired-tile-height
desired-tile-width-input | desired-tile-width
From | To
PanBand-space-res-0.6sqkm | image1-spatial-resolution
get-MultispectralBand
From | To
PanBand-image-data | tiling-image-input
From | To
Multispectral-space-res-2.0sqkm | image2-spatial-resolution
From | To
Multispectral-image-data | tiling-image-input
From | To
Multispectral-time-instant | stats-dateTime-input
Multispectral-image-data-set | stats-dataset-input
QuickBird-Sensor | stats-sensor-input
get-Dimensions
From | To
image1-tile-width-output | tile-width-input
image1-tile-height-output | tile-height-input
From | To
image2-tile-width-output | tile-width-input
image2-tile-height-output | tile-height-input
From | To
PanBand-time-instant | stats-dateTime-input
PanBand-image-data-set | stats-dataset-input
QuickBird-Sensor | stats-sensor-input
get-PanTiles
From | To
tiling-image-output | edge-image-input
get-MultiTiles
perform-Edge
From | To
tiling-image-output | NDVI-image-input
From | To
edge-image-output | stats-image-input
perform-NDVI
get-Edge-stats
From | To
NDVI-image-output | stats-image-input
get-NDVI-stats
From | To
stats-features-output | NDVI-stats-features-output
From | To
stats-features-output | Edge-stats-features-output
P
Finish/Out

Figure 6.51: The OWL-S workflow used to coordinate agent interactions in the ISIS application [156]

6.4.3.2 The ISIS Tool Agents

The Tiling Tool Agent is used to split the PanBand and Multispectral images acquired from the QuickBird and SPOT Sensor Agents into tiles. However, the bands of the different images have different spatial resolutions. The Dimensions Tool Agent is used to determine the correct dimensions (width and height) for tiles from two image data sets that have different spatial resolutions so that the tiles cover the same spatial location.

The Edge Tool Agent performs edge detection on a given satellite image. The NDVI Tool Agent calculates the Near Difference Vegetation Index (NDVI) of an image. This feature extraction technique assesses the percentage of vegetation in an image. The NIR and Red bands of the Multispectral images are used to calculate NDVI. The NDVI Tool Agent takes as input the NIR and Red bands of a Multispectral image, performs the NDVI calculation over the image and outputs an image of the NDVI signature.

The Stats Tool Agent calculates image statistics. It takes as input an image and outputs the statistics of the image. The Stats Tool Agent currently calculates the image mean, median, standard deviation, integrated density, skewness and kurtosis. The ISIS application uses the Stats Tool Agent to calculate the image statistics of the NDVI and Edge images produced by the NDVI and Edge Tool Agents respectively.

6.4.3.3 ISIS Workflow Agent

The ISIS Workflow Agent coordinates the execution of the image processing workflow shown in figure 6.51. The implementation of the Workflow Agent is similar to that of the Hotspot Detector Workflow Agent described in section 6.2.6.

To demonstrate the reusability of the ISIS agents, two additional workflows were created using different combinations of ISIS Tool and Sensor agents. The workflow in figure 6.52 takes as input a Panchromatic image from the QuickBird Sensor Agent, tiles the images into equal sized segments and then performs edge detection on each tile segment. The workflow in figure 6.53 takes as input a Multispectral image from the QuickBird Sensor Agent, performs NDVI on the image, tiles the image and then calculates image statistics on each tile segment.

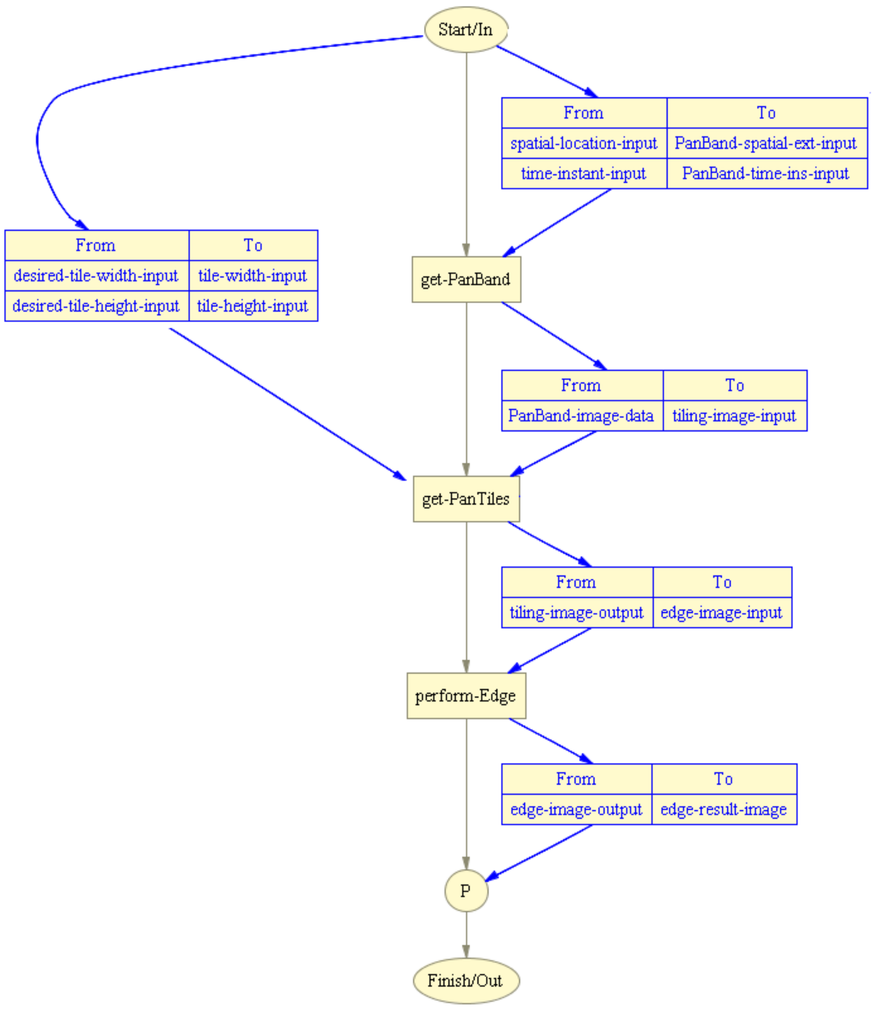
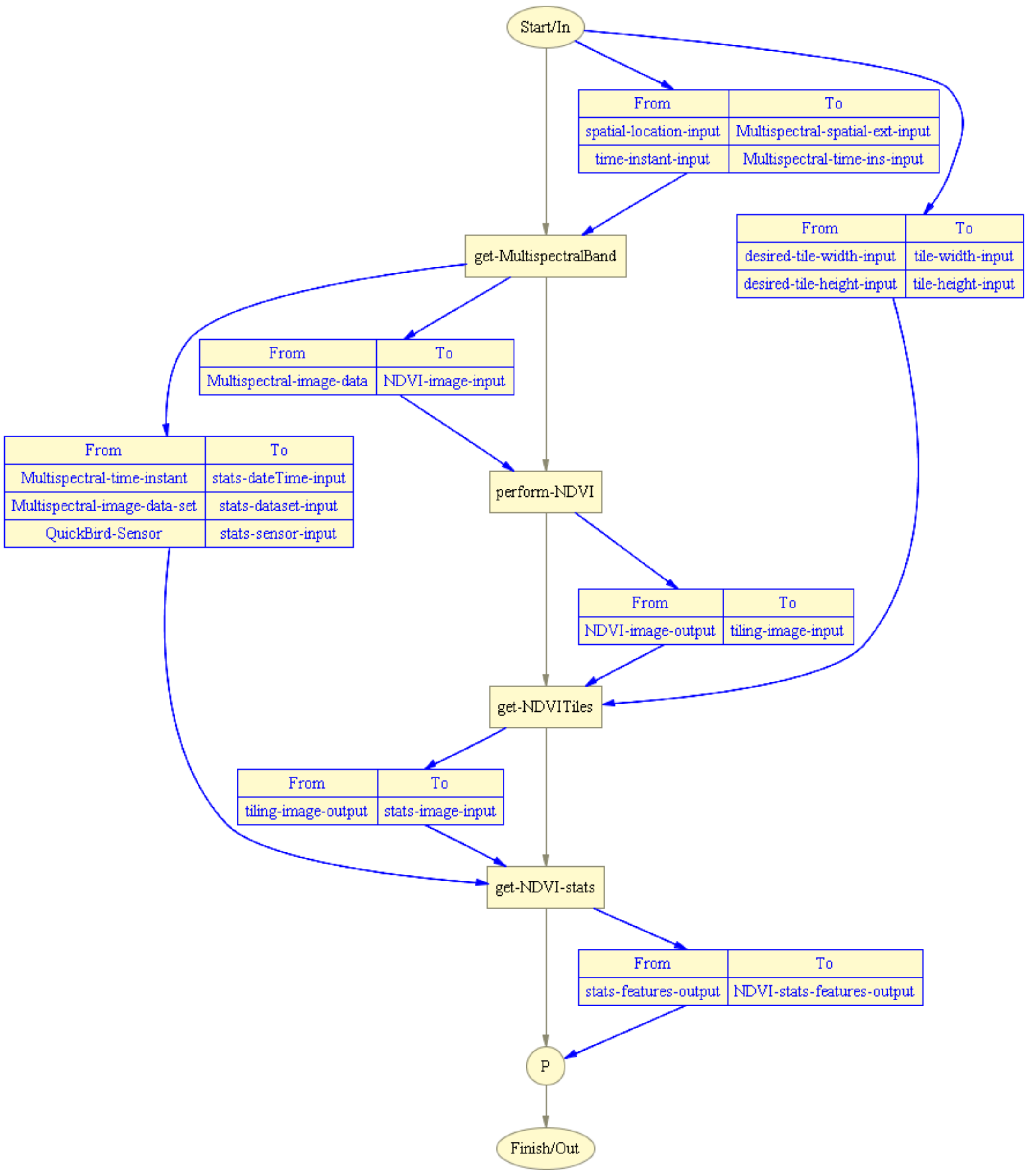
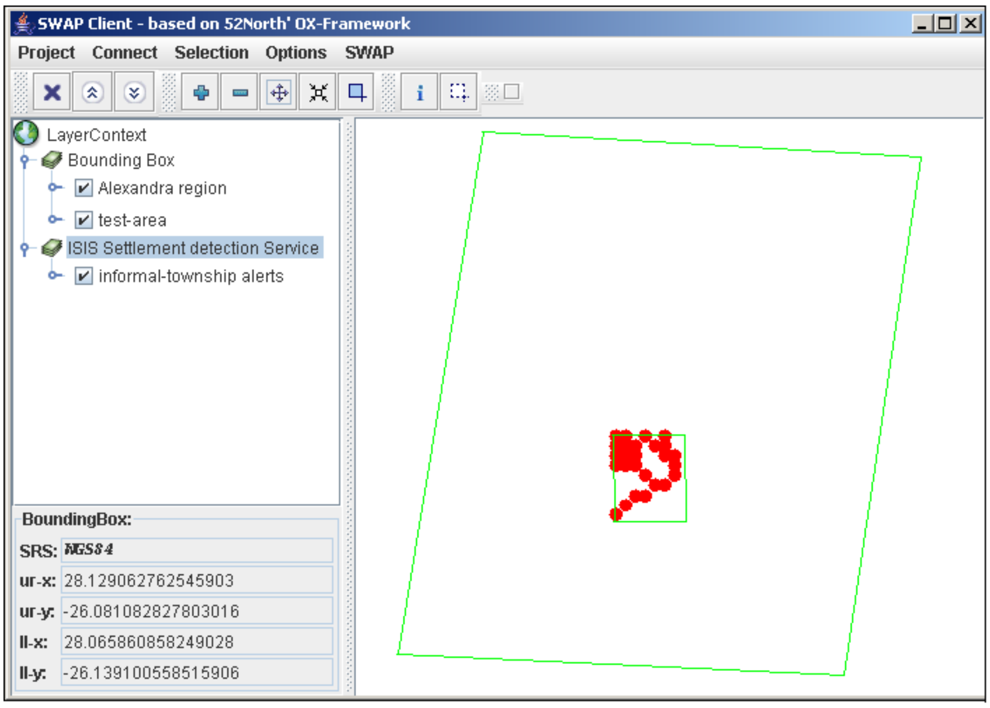


Figure 6.52: Alternate ISIS workflow 1 [156]



Start/In
From | To
spatial-location-input | Multispectral-spatial-ext-input
time-instant-input | Multispectral-time-ins-input
From | To
desired-tile-width-input | tile-width-input
desired-tile-height-input | tile-height-input
get-MultispectralBand
From | To
Multispectral-image-data | NDVI-image-input
From | To
Multispectral-time-instant | stats-dateTime-input
Multispectral-image-data-set | stats-dataset-input
QuickBird-Sensor | stats-sensor-input
perform-NDVI
From | To
NDVI-image-output | tiling-image-input
get-NDVITiles
From | To
tiling-image-output | stats-image-input
get-NDVI-stats
From | To
stats-features-output | NDVI-stats-features-output
P
Finish/Out

Figure 6.53: Alternate ISIS workflow 2 [156]



BoundingBox:

SRS: WGS84
ur-x: 28.129062762545903
ur-y: -26.081082827803016
ll-x: 28.065860858249028
ll-y: -26.139100558515906

Figure 6.54: The OXFClient showing the bounding box over Alexandra and the area of interest within which the informal Settlement results are displayed [156]

6.4.3.4 ISIS Application Agent

The ISIS Application Agent hosts and maintains the ISIS application. It invokes the ISIS Workflow Agent to retrieve statistical values for Edge and NDVI tile segments. The results are classified to detect informal settlements. The ISIS Application Agent provides settlement alerts to end-users according to spatial, temporal and thematic constraints specified by the end user.

6.4.3.5 ISIS User Agent

An end-user interacts with the ISIS Application through the ISIS User Agent. Each ISIS end user has their own ISIS User Agent. It allows a user to retrieve and visualise informal settlement alerts from the ISIS Application. Figure 6.54 shows the OXFClient interface at the User Agent. The outer bounding box represents the Alexandra region and the inner one is an area of interest demarcated by the user. A satellite image corresponding to the selected area is shown in figure 6.55. Figure 6.56 shows a magnification of the area of interest. Informal settlements are correctly clustered at the top of the image with some erroneous values spread through the bottom.

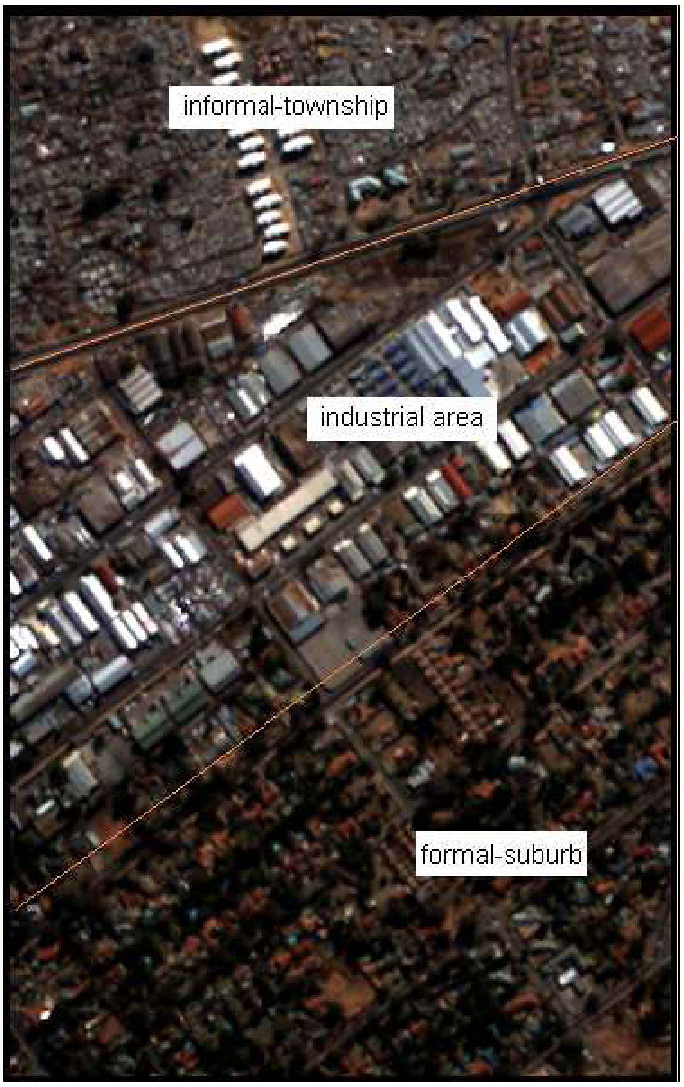
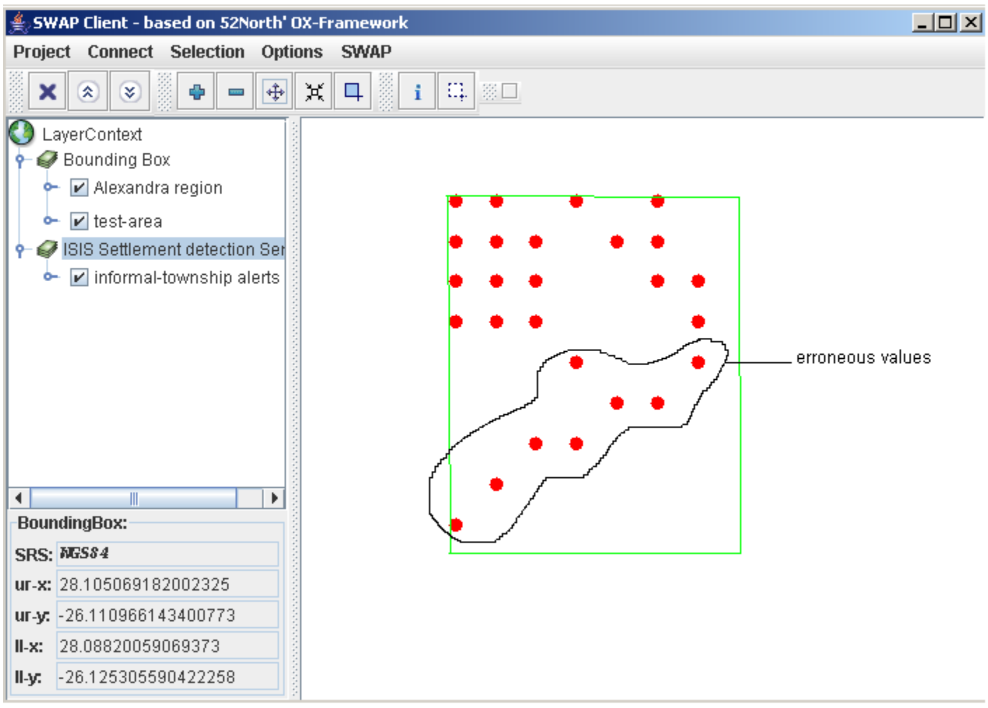


Figure 6.55: Sattelite image showing informal settlements over Alexandra [156]


SWAP Client - based on 52North' OX-Framework
Project Connect Selection Options SWAP
LayerContext
Bounding Box
Alexandra region
test-area
ISIS Settlement detection Ser
informal-township alerts
erroneous values

BoundingBox:
SRS: WGS84
ur-x: 28.105069182002325
ur-y: -26.110966143400773
ll-x: 28.08820059069373
ll-y: -26.125305590422258

Figure 6.56: Informal-townships being displayed on the OXFClient [156]

6.4.4 Discussion

This section described an application for detecting informal settlements that was successfully designed, implemented and tested on the SWAP framework.

The SWAP abstract architecture provided all the necessary abstractions required to model the ISIS application. Identifying and mapping application components (table 6.12) to agent types (table 6.13) was largely intuitive and straight forward. No additional abstractions were necessary. The only difficulty experienced by the developer was whether to model the WEKA classifier as a Modeling Agent in the Knowledge Layer or within the Application Agent at the Application Layer. Since the current model is tightly coupled to the ISIS application with little possibility for reuse, it was decided to incorporate the classifier within the ISIS Application Agent. However, it may be possible to provide machine learning algorithms via Tool Agents, and train and deploy specific classification models as Modeling Agents. This can allow different instances of the same classification model, e.g. the BayesNet classifier, to be reused across different applications. This requires further investigation on the training and management of different model instances.

Code reuse As no new protocols or message types were introduced, the existing message handlers proved adequate and were reused. However, new data, tool and application adapters were required.

Possibilities for reuse of agents Each agent at the Sensor and Knowledge Layers has the potential to be reused. Quickbird and SPOT sensor data can be reused for different applications. To increase the potential for reuse, the five Tool Agents were designed to be independent of the ISIS application. The Dimensions Tool Agent can take any pair of images and determine the appropriate tile size for each image such that the tile segments cover the same location. The Tiling Tool Agent can split any image into tile segments of the specified size. The NDVI tool agent can calculate NDVI values for the red and near infrared bands of both image data sets. The Edge Tool agent can perform edge detection on any image. The Stats Tool Agent can produce statistical values for any image. Thus all Sensor and Tool Agents have a large potential for reuse in other workflows for other applications.

6.5 SUMMARY

This chapter describes the design and implementation of two SWAP applications. The development and deployment of the first application for wildfire detection illustrates key aspects of the framework. The uncertainty ontology and reasoner is used to show how a Bayesian Network can be used to make inferences about the occurrence of wildfires from thermal satellite data. This application also illustrates the implementation and the operation of the different SWAP agent types as well as the deployment of end user applications. The second SWAP application for detecting informal settlements further illustrates how the framework is used to design and develop Sensor Web applications. It shows that the framework can be used to develop applications across different earth observation domains with different requirements.

A discussion and analysis of the Sensor Web Agent Platform as an enabling framework for developing ontology driven agent based applications is provided in the next chapter.

Chapter 7

DISCUSSION AND CONCLUSIONS

7.1 CONTEXT OF THIS RESEARCH

A single worldwide Sensor Web is an open and complex computing environment that must enable the sharing and reuse of dynamic geospatial data, knowledge, data processing and predictive modeling services by a wide user community with different requirements, skills and backgrounds. Services must be discovered and assembled in different configurations to extract information, to test theories and ultimately to capture and to advance our knowledge and understanding of the natural environment. It must also support the construction and deployment of real time end user alerting and monitoring applications that incorporate these services. Applications must be easily modified to reflect new service offerings in order to provide relevant and accurate information to decision makers. Ontologies have shown promise as a technology for sharing and integrating data in open environments, while software agents provide mechanisms to dynamically discover, invoke and assimilate these services. This research demonstrates how these two technologies can be integrated into an Ontology Driven Multi-Agent Sensor Web.

7.1.1 Software agents and multiagent systems

Software agents are active software components that represent an independent sphere of control with its own goals and purpose. The Sensor Web will encompass multiple information systems with different views of the world, that have varying degrees of complexity, that use different technologies with single or multiple users (heterogeneity) each with its own sphere of control (autonomy), and with each undergoing continuous change in line with its user(s) or organisation's requirements and goals (dynamism). Software agents are more appropriate design abstractions for Sensor Web software components than services as they better represent the autonomy, dynamism and heterogeneity of the different information systems on the Sensor Web.

An agent represents the interests of a particular information system and an interface to access specific functionality that the information system chooses to offer to other agents (information systems). In principle, a participating agent may change its service offering or limit or refuse service requests. Software agents have broad, high level interfaces that reflect the level of the agent's participation in the Sensor Web. Interacting and cooperating with other agents is crucial to an agent's operation. Typically agents communicate by message passing. The content of the messages have well defined semantics that are defined in a shared knowledge model. To interpret and react to these message agents must commit to the shared knowledge model. Communicating at the semantic level facilitates dynamic interoperability, agent discovery, agent coordination and agent service composition. Agents require some internal mechanism to interpret and react to changes in the system. Deliberate agents maintain an internal model of the world, for Internet agents this is usually based on symbolic logic, which they use for planning and to initiate actions to achieve their goals (see section 2.2.1.3). Their beliefs usually reflect their world model and their desires are their purpose or end state. An agent's goals are a non conflicting subset of the desires and its intentions is a commitment to undertake a series of planned actions for achieving these goals. This ability of an agent to automatically react to changes in the system, with minimal human interaction highlights an important characteristic of agents, that of autonomy.

Agent service descriptions are also specified according to a shared knowledge model. Agents are able to dynamically discover, invoke and assimilate the responses of agents of which they were not previously aware or not specifically programmed to interact with. The extent to which this can be achieved depends on the expressivity, level of detail and the degree of consistency when interpreting the shared knowledge model.

7.1.2 Ontologies

Ontologies are shared knowledge models that explicitly describe the meaning of and relations between concepts that are used within an information system. Ontologies facilitate data discovery as well as data integration. By committing to and publishing its ontology, an agent allows other agents with which it interacts to interpret and consume messages that it produces. Even though many applications have been developed which demonstrates the power of ontologies for discovering, integrating and making sense of disparate data sets (see section 2.4.2), ontologies have the potential to play a much larger role in an open and dynamic distributed information system. A compelling vision is the development of ontology driven information systems, where software agents are able to *autonomously* discover, access and assimilate the services provided by other agents. This vision can be achieved by providing ontologies that not only capture the meaning of the messages transferred between agents but as well as the descriptions of the agents themselves. In this way agent services can be dynamically queried, accessed, manipulated and assembled into complex executable workflows. By explicitly providing descriptions of the services themselves as well as the data models that drive their operation, the service descriptions and data models become runtime components that can be accessed, queried and modified. This allows for the dynamism of data models and service offerings while still allowing services to interoperate. Furthermore, this allows data models as well as service offerings to change and evolve naturally with minimal impact on and without having to re-engineer the system.

7.2 SUMMARY OF RESULTS

The main outcome of this research is an Ontology Driven Multi-Agent System (ODMAS) framework together with a middleware platform, i.e. the Sensor Web Agent Platform (SWAP). SWAP provides a: a semantic infrastructure which includes a set of ontologies and corresponding reasoners; an abstract architecture that guides the design of agent based applications, an internal agent architecture to guide the internal operation of an ontology driven agent; and a development platform which eases the implementation and deployment of individual agents. Two working case study applications were designed and implemented on SWAP. In this section we briefly describe the two implemented application and then summarise the key features of the framework.

The wildfire detection application, described in section 6.1, is a simple Sensor Web application that illustrates the operation of a complete end to end SWAP application. It provides working implementations and demonstrates: the roles of each of the SWAP agents; the role of the different SWAP ontologies in domain modeling, agent service descriptions and agent communication; as well as workflow composition and dynamic workflow execution. Sensor data more specifically brightness temperature observations is exposed via a Sensor Agent, the MSG Sensor Agent. A single Tool Agent, the Contextual Algorithm (CA) Tool Agent detects spatial anomalies (hotspots or lowspots) in image data. The Hotspot Detection

Workflow (HDW) Agent incorporates an OWL-S workflow which when executed, retrieves brightness temperature data from the MSG Sensor Agent and invokes a Tool Agent to extract hotspots from this data. The HDW Agent is triggered by the Wildfire Detection Application (WDA) Agent which uses these hotspots as indications of wildfires. Any SWAP User Agent that has downloaded and installed the wildfire detection application adapter is able to register customised alert requests with the WDA Agent. Appropriate alerts are then sent to and can be visualised by the SWAP User Agent. An implementation of a Wildfire Spread Prediction Modeling Agent is also provided which illustrates the operation of a Modeling Agent. Furthermore, the use of the SWAP uncertainty ontologies is demonstrated via a Bayesian Network that captures the causal relations between brightness temperature readings, hotspots and wildfires. In this way the uncertainty associated with a specific wildfire observation can be represented, shared and used in the system.

The second application, described in section 6.4, detects informal settlements from high resolution satellite imagery. While this application also involves the extraction of information from remotely sensed imagery, the two domains, i.e. wildfire and informal settlements, and the imagery used differ widely. Furthermore, this application is more reflective of a real world remote sensing application. It incorporates data sets from different satellites (Quickbird and SPOT), uses observations from different spectral frequency bands (Panband and Multispectral), uses a complex image processing chain, consisting of five processing steps for features extraction, and a classification step to classify these features into different classes of informal settlements. At the Sensor Layer, QuickBird image data is offered by the Quickbird Sensor Agent. At the Knowledge Layer, the ISIS Workflow Agent retrieves the QuickBird data from the Quickbird Sensor Agent and passes it to a series of Tool Agents which segment and extract appropriate features from the data. The Dimensions and Tiling Tool Agents segments the image into a series of tiles. These tiles are then passed through the Edge and NDVI Tool Agents, which determine the edges and the vegetation index for each tile respectively. Edge and NDVI images for each tile are then passed through the Stats Tool Agent, which calculates statistics such as mean, standard deviation, density, skewness and kurtosis for each image. The statistics for each tile are then passed to the ISIS Application Agent (AA) for classification. Within the ISIS AA a naive Bayes classifier is used to classify Edge and NDVI image statistics into different types of informal settlements. The ISIS User Agent is used to query the ISIS AA and to visualise detected informal settlements. The application also demonstrates how Quickbird image data can be replaced by SPOT image data, which has a higher temporal resolution (revisit time) but a lower spatial resolution.

7.2.1 Semantic framework

The SWAP semantic framework (see section 4.3) aims to bridge the semantic gap between machine and human. The SWAP ontology infrastructure delineates conceptual ontologies which provide support for modeling and representing observations and theories grounded in and about the physical world, from technical ontologies to model and represent the software entities (agents) that will host and process these observations within the Sensor Web.

7.2.1.1 Conceptual Ontologies: representing observations and theories

A set of four top level conceptual OWL ontologies and reasoners provide representational support for capturing the semantics of observations, algorithms and theories that can be applied to these observations. Their design is based on the human cognitive system, i.e. how humans store and reason about knowledge [132]. Mennis defines three systems of knowledge, i.e. theme, space and time. SWAP introduces a fourth system, i.e. uncertainty, to provide a more expressive and holistic approach for modeling observations and theories which form an integral part of the Sensor Web. Each system consists of an OWL ontology and an associated reasoner. The four conceptual ontologies allows for reasoning at an abstract level, without worrying about the technical detail.

In order to fully participate in SWAP all users must commit to the integrated conceptual model encapsulated by the four SWAP top level ontologies. There are many different models for space, time, theme and uncertainty. In order to appeal to a broad user community and gain wider participation, complex theories of space and time were avoided. The aim was to incorporate the simplest and most widely used and accepted models where possible. The *swap-theme* ontology provides for the representation of observations and is based on the OGC's model of observations and measurements [50]. The *eo-domain* ontology (figure 4.7) allows for the incorporation of domain concepts from the NASA SWEET [167] ontologies. The *swap-space* ontology defines the spatial operators defined in the OGC Simple SQL features spatial operators [49, 61, 150]. The *swap-time* ontology incorporates the temporal model defined in *OWLTime* [95, 96] and follows Allen's [20] widely used representation of intervals. The *swap-uncertainty* ontology incorporates Bayesian probability [171], which is widely used in practical applications to represent degrees of belief, and allows for the incorporation of Bayesian Networks to represent different theories of cause and effect relations between events in the physical world.

The effectiveness of these ontologies to model and represent typical artifacts in practical Sensor Web applications was demonstrated by their use in the two case study applications. Several ontologies were created using these top level conceptual ontologies to:

- **represent and query sensor data**: demonstrated for three disparate satellite image data sets (Quickbird, SPOT and MSG) served by three Sensor Agents
- **represent and invoke image processing algorithms and prediction models**: demonstrated for six different image processing algorithms offered by Tool Agents and a fire spread prediction model offered by a Modeling Agent
- **represent and query alerts for phenomena** demonstrated for wildfire and informal settlement alerts offered by two Application Agents
- **represent theories that encapsulate causal relations between events** demonstrated by capturing a theory for detecting wildfires, i.e. the causal relation between wildfires, temperature hotspots and brightness temperature observations (see section 6.1.2)

7.2.1.2 Reasoning about knowledge represented in the ontologies

Two independent rule based reasoners, a spatial and a temporal reasoner perform inferences about spatial and temporal entities specified in the *swap-space* and *swap-time* ontologies. A thematic reasoner uses the SWEET ontologies to reason about thematic concepts. A Bayesian inference engine dynamically populates Bayesian Networks with observable events, performs inferencing on these events and determines and records the occurrence of other events.

7.2.1.3 System modeling, services, workflows, applications

SWAP provides three technical ontologies, i.e. *swap-data*, *swap-agent* and *swap-task* that provide representational support to describe the system entities that are required for hosting and transmitting observations, and for executing the algorithms and theories which have been described above. The *swap-data* ontology provides descriptions of different data structures that can be exchanged between agents. This includes coverage (image) and feature data as well as units of measure.

The *swap-agent* ontology provides support for representing an agent, the service it hosts and the interaction protocol required to invoke the service. It provides support for representing all six Sensor Web agent types identified in the SWAP abstract architecture (see figure 4.1). These are data provider (Sensor) agents, processing or data transformation (Tool) agents, modeling (Modeling) agents and coordination (Workflow) and application (Application) agents. Each agent type has a corresponding service description with a set of common attributes that capture the conceptual functionality of the service. Sensor Agents provide a description of the observations that they provide, while Tool and Modeling Agents provide a description of the data processing algorithms and prediction models that they respectively provide. Service description attributes are grouped into the four different conceptual systems, i.e. spatial, temporal, thematic and uncertainty, and are specified using concepts from the appropriate top level ontology. Service descriptions also contain service invocation information in the form of input and output mappings. A request and a response message template is used for invoking and interpreting the response of the service. The request message template specifies all service invocation parameters, which may be mandatory or optional parameters that have default values. Users populate mandatory parameters and may also specify optional parameters for finer control of the service. These message templates are used to dynamically invoke a service and to consume and interpret its results. This mechanism is described in sections 4.5.2 and 4.5.4 and illustrated using the Hotspot Detection Workflow Agent (section 6.2.6) which shows the dynamic invocation of a Tool and Sensor Agent using appropriate message templates. This bridges the gap between service selection and use, i.e. once a suitable service has been identified it can be dynamically invoked and its results can be dynamically interpreted.

The SWAP service directory contains service descriptions for all agent services offered in the system. By committing to the SWAP top level ontologies, users are better able to query, navigate and filter the services on offer. Besides being able to discover the availability of new services in the system, users are better able to determine the nature of observations contained in data sets and identify algorithms in the system that can process these observations. Thus they are able to identify appropriate service(s) that best fits their requirements. In chapter 6 service (or capability) descriptions, for each of the different agents used in the wildfire detection application, is described. Examples of queries to the SWAP Directory Agent to discover these services are also shown.

The *swap-task* ontology is based on OWL-S, and provides algorithmic primitives to assemble multiple agents into executable agent workflows. An agent is represented as an atomic processes and OWL-S algorithmic constructs are used to assemble multiple agents into appropriate sequences of invocations or composite processes. The main extension to OWL-S is a process to agent mapping that allows OWL-S processing steps to be transformed into agent invocations at runtime. The mapping specifies request and response templates that are used to transform each processing step into an appropriate request and response message used to invoke an agent and to interpret its response (see section 4.5.4).

The technical and conceptual ontologies allow SWAP users to represent complex information processing chains or workflows. Users search semantic agent service descriptions and identify appropriate sensor data sets, algorithms and models to apply to these data sets. Once the appropriate agents are identified, users use the algorithmic constructs in the *swap-task* ontology to specify a processing workflow that assembles different agent services in an appropriate sequence for execution. Each workflow represents new functionality in the system and can be made persistent by deploying it on a Workflow Agent to create a persistent execution environment for the workflow to be executed on demand. As a workflow is fully specified and executed from its OWL-S specification, the appropriate ontologies (which contain the workflow) can be shared, downloaded and executed locally. Furthermore, once the workflow is downloaded it can be easily modified and executed locally by SWAP users. A workflow is represented as a composite processes, which means that it can be incorporated into other composite processes (workflows). This allows for reuse of existing workflows within other workflows and for creating and managing large and complex nested workflows. Currently, workflows are created and modified manually via an ontology editor. However, given that the semantics of both the conceptual and the technical aspects of each service are specified in the service description, this provides a sound foundation for automating workflow composition.

Two executable agent workflow ontologies, that capture the image processing workflows for each of the case study applications were created:

- **Extraction of hotspots workflow** This workflow (see section 6.2.6) combines two agents via two processing steps, and combines a single Sensor Agent with a Tool Agent to extract hotspots from brightness temperature image data.

- **Extraction of informal settlements** This is a more complex workflow (see section 6.4) that involves multiple invocations of six different agents, one Sensor Agent and five Tool Agents which tiles and extract features for informal settlement detection from satellite image data. Two data sets are retrieved from a Sensor Agent and both data sets are processed individually by all five Tool Agents.

A key feature that is demonstrated is that the entire workflow is encapsulated in the ontology which can be shared, downloaded and modified to incorporate new agents or to change invocation parameters.

7.2.2 Framework for designing, deploying and accessing agents and applications

The semantic infrastructure, discussed above, together with the SWAP abstract architecture, SWAP internal agent architecture and supporting middleware platform provide comprehensive support to faciliate the design, development, deployment and use of ontology driven Sensor Web agents as well as agent based alerting and monitoring applications.

7.2.2.1 Design

As described and illustrated in the informal settlement application (see section 6.4) developers uses the abstract architecture to identify the observations, algorithms, theories and models that are required to generate the required alerts. In parallel, the developer creates appropriate ontologies, by reusing and extending the top level ontologies, that describe the service offering and operation of each agent and the observations, algorithms, models, theories and alerts that are required for the application.

The abstract architecture aims to provide different conceptual layers and functional agent abstractions to ease system modeling. The architecture provides three layers of abstraction, the Sensor Layer, Knowledge Layer and the Application Layer. Each layer provides appropriate agents that encapsulate the typical functionality required at that layer, i.e. Sensor Agents or sensor data providers at the Sensor Layer; Tool, Modeling and Workflow Agents or information processing, data analysis and coordination functionality at the Knowledge Layer; and Application and User Agents or end user and decision making functionality at the Application Layer. For example, developers that may only want to deploy a sensor data provider, can easily identify Sensor Agents at the Sensor Layer as being the appropriate agent abstraction for this. By interrogating the appropriate service description for Sensor Agents, i.e. the sensor data service, and typical conceptual attributes they construct an appropriate service description. Terms from the SWAP ontologies are used or are extended accordingly if appropriate terms are not available. Data processing and modeling experts work at the Knowledge Layer and deploy new Tool, Modeling and Workflow Agents. Application developers reuse Tool, Modeling, Workflow and Sensor Agents from lower levels to expose applications to end users. Developers are thus able to identify which agents fit their requirements and can focus just on developing those agents at the appropriate layers while reusing functionality from lower layers and providing additional functionality to developers at higher layers. The abstract architecture explicitly avoids providing abstractions for MAS middleware services, such as agent directory services, as done in other architectures. Developers can thus concentrate on identifying and modeling the individual software agents required to achieve the required alerting functionality separately from identifying which agent middleware services to use.

7.2.2.2 Development and deployment

SWAP uses the MASII agent platform (see chapter 3) for implementing and deploying agents as well as agent based applications. All SWAP agents developed in the two case study applications were implemented on the MASII platform. A key feature of the MASII platform is the grouping of ontologies and message handlers into pluggable service adapters, which can be discovered, downloaded and installed at runtime. This allows SWAP end user applications to be discovered and installed at runtime (see section 6.3).

SWAP attempts to incorporate both the declarative programming paradigm usually used to develop agents and logic based systems and the imperative programming approach used in object oriented and web services development. The internal agent architecture (see section 4.7.1) allows the developer to use existing programming libraries, sensor data stores as well as OGC web services, to provide the underlying functionality of an agent. Two key features that enable this is a data mapping API that converts ontology instances into OpenGIS Java objects, as well as the provision of generic Java interfaces to implement the functionality of the different SWAP agent types.

Three Java interfaces, the *DataAdapter*, *ToolAdapter* and *ModelingAdapter* interface are used to implement the underlying functionality of the Sensor, Tool and Modeling agents respectively. The implementation of these adapter interfaces form a major part of the development of Sensor, Tool and Modeling Agents. These classes are used internally by these agents to fulfil service requests. External service requests are directed internally to the adapter class. The data mapping API transforms any incoming ontology data to Java objects and supplies this to the adapter class. After the adapter class completes its processing, the agent transforms any resultant Java objects to ontology instances which are used to compose the service response message.

The data mapping API and the adapter interfaces have been designed and demonstrated specifically to leverage the use of OpenGIS compliant libraries. The current implementation is demonstrated using 52 North's OXFramework environment but allows for accessing other OpenGIS compliant libraries as well as to access and incorporate data from existing spatial databases and OGC SWE services. SWAP developers are able to reuse and leverage existing software libraries and services available in the geospatial community to provide the underlying functionality of an agent. This not only reduces development time, but allows for the incorporation of legacy software, spatial databases as well as OGC SWE services into the system. It is thus possible to separate, but still leverage, the benefits of both the ontology-agent development paradigm and the Java development paradigm. Java developers may choose to concentrate only on implementing instances of the adapter classes, while developers familiar with the agent and ontology paradigm can concentrate on exposing the functionality of these adapters via appropriate Sensor, Tool and Modeling Agents. This also allows for developers to gradually transition from the object oriented to the ontology driven agent paradigm.

The above approach was used to implement all Sensor, Tool and Modeling Agent in the two case study applications. Ten adapters for the three Sensor Agents, six Tool Agents, and one Modeling Agent were implemented. An additional adapter for a Sensor Agent that offers weather data by forwarding requests to an OGC Sensor Observation Service (SOS) was implemented (see section 6.2.1.2) to demonstrate how an OGC web service can be incorporated and accessed. This approach can also be used to incorporate OpenGIS compliant visualisation and GIS client development libraries to implement the user interface and alert visualisation components of the SWAP User Agents. The data mapping API and the OXFramework Java Swing visualisation libraries are also used to implement the graphical user interface for visualising wildfire and informal settlement alerts within the two User Agents.

An Application and User Agent were implemented for both the informal settlement and wildfire applications. A separate visualisation interface and query mechanism is created for each application, thus allowing different applications to have different querying and visualisation interfaces. Users are also able to choose which applications to install on their User Agent. The current wildfire and informal settlement user interfaces demonstrate simple alert querying and visualisation. However, as both user interfaces were implemented using the OXFramework client libraries, these libraries can be used to provide better visualisation of alerts or to provide mechanisms for users to interact with or manipulate alerts.

7.2.3 Usage

SWAP can provide significant benefits to a wide range of users. These include end users for accessing, managing and visualising information provided by real time monitoring applications, developers for deploying agents and agent based applications and earth observation scientists who can use the Sensor Web as a scientific computing platform to facilitate knowledge sharing and discovery.

7.2.3.1 Earth observation scientists/ Domain experts

Earth observation scientists and domain experts can use SWAP to publish, share and reuse earth observation data and knowledge.

Capturing and sharing knowledge The ontology infrastructure provides a high level conceptual model which facilitates the capturing of geospatial concepts to describe observations, algorithms and prediction models as well as subjective theories about causal relations between events that occur in the physical environment. These concepts as well as the relations between concepts are captured by making unambiguous and logically consistent statements that are grouped into OWL ontologies. By extending and reusing the same high level conceptual model, ontology fragments created by different users can be shared, logically verified, and integrated and reused. Users thus have access to a vast globally distributed knowledge repository that can incorporate different domain models which can still be interpreted using the top level ontologies.

Using agent services for experimentation and knowledge discovery The ontology infrastructure provides a model for describing the data sets, data processing algorithms as well as the prediction models that are offered by agents. This facilitates and guides consistent service descriptions. By interrogating the top level and domain ontologies, scientists can navigate, query or filter services, place them in context, as well as identify and incorporate the functionality that they require.

An important aspect of an ODMAS is to apply multiple processing and modeling services to specific data to perform complex information processing chains or workflows. Scientists can use the shared ontologies to navigate service descriptions and select a number of agents required to fulfill a task. The algorithmic primitives from the swap-task ontologies are used to specify workflows that assemble the selected agents into an appropriate sequence for execution. These workflow specifications can be published and discovered as a new service offering in the system. The ontology containing the workflow specification can be retrieved, modified and executed locally. Scientists can thus experiment locally with removing and replacing individual agent invocations, changing the sequence in which agents are executed, and even adjusting invocation parameters. The modified workflow can be immediately executed and evaluated.

Furthermore, given that service descriptions as well as the algorithmic constructs have formal semantics, there is potential to achieve a degree of automation for agent discovery as well as workflow construction and modification. This will definitely ease experimentation and testing as well as introduce the possibility of discovering knowledge that may not be easily be discovered using existing investigation and experimentation methods.

7.2.3.2 Sensor Web application end users

Sensor Web application end users typically require real time monitoring of the environment. They are usually interested in the occurrence of phenomena that have impact on human life, both naturally occurring phenomena e.g. wildfire and floods, as well as man made phenomena, e.g. informal settlements. End users belong to a variety of organisations including government departments and utilities, disaster management response teams and urban planning departments. End users typically have limited or no software development skills and may even have limited domain knowledge. Their key requirement is to receive real time alerts when some phenomena occurs at a specified spatial location. Sensor Web end users must be able to discover existing alerting functionality, specify custom alerting criteria as well as receive and visualise alerts. End users interact with the system via User Agents. Application Agents generate, manage and supply alerts to User Agents for specific monitoring applications to end users. Each end user manages and configures their own User Agent and uses it to register custom alert subscriptions with one or more Application Agents. User Agents also store and visualise alerts that are received.

7.2.3.3 Sensor Web Developers

As described above, a comprehensive development framework is provided to guide developers through the design, development and deployment of agent based Sensor Web services and end user applications. The abstract architecture allows developers to identify the number and types of agents required for a particular Sensor Web application, the ontologies are used to specify the agent interfaces including the structure and content of messages transferred between agents, the internal agent architecture and the supporting middleware platform provides a programming toolkit including sample agents for implementing and deploying individual agents. As all agent interfaces and interactions are explicitly specified within agent ontologies, these ontologies form an online and dynamic model of the system. Agents can be reconfigured by modifying the appropriate ontologies. These ontologies can be accessed by new developers to understand the operation of specific agents in the system.

7.3 COMPARISON WITH OTHER SYSTEMS

The two main features that distinguish SWAP from other Sensor Web approaches are: firstly the agent based framework for modeling, developing and deploying Sensor Web services; and secondly that applications are supported by a comprehensive and coherent semantic infrastructure.

7.3.1 Agent based Sensor Web approaches

The definition of an agent in this work is similar to the definition proposed by Sengupta and Sieber [176] for Geospatial Software Agents. SWAP agents represent autonomous users or organisations, i.e. they have strong autonomy, and are inherently equipped to handle the unique qualities of geospatial data. However, SWAP agents do not currently have explicit geospatial locations in the system to facilitate mobility and optimise discovery of agents. However, the current design lends itself to such an extension.

Most existing agent based approaches focus on distributed processing, data filtering and scalability. However, the author to date has not found any approach that provides a comprehensive ontological infrastructure that supports interactions and the design of Sensor Web agents as proposed in this work.

The SWAP three layered architecture was inspired by the layered architectures proposed in [22, 29]. Conceptually Abacus [22] provides abstractions that are very similar to SWAP. The Sensor Layer is similar to the Contribution Layer, the Knowledge Layer is similar to the Management and Processing Layer, and the Application Layer is similar to the Distribution Layer. However, Abacus focusses on distributing processing based on location, where each agent is responsible for different spatial sectors and together provide coverage of the entire observation region. This approach has benefits for specific applications, such as radar applications and for distributed real-time processing and monitoring. However, this restricts how processing agents can be reused across different applications and different data sets with different spatial coverages. In SWAP, processing agents (Modeling and Tool Agents) are demarcated according to the function that they perform rather than a spatial region of responsibility, which makes it easy to discover and reuse these agents for different applications. The abstractions provide by Biswas et al. [28, 29] have many similarities to those provided by SWAP. The concept of Application Agents in an Application layer is similar to that of SWAP. However, Sensor Agents occur in the middle Service Layer rather than in the Data Layer as in SWAP. The Service Layer also provides middleware services. Conceptually, the SWAP abstract architecture is separate from the MAS infrastructure and the middleware services that it provides. It provides functional abstractions solely for designing and developing Sensor Web services. Middleware services such as discovery, security, communication and scalability reside in the MAS architecture. SWAP currently does not support mobility, but can be extended to do so.

The idea of composing and storing executable agent workflows for geospatial image analysis is a key feature in the AIGA architecture [143]. However, AIGA does not focus on the ontology infrastructure. SWAP provides a more comprehensive ontology infrastructure which, by incorporating OWL-S, provides richer control structures than that provided in AIGA for workflow compositions such as loops, conditionals and branching.

IRISNET [41, 77] provides a two tiered architecture that supports distributed data storage and processing and provides abstractions for data sources and information extraction. While it provides strong support for agent mobility and distributed processing, it does not provide a semantic infrastructure.

More recently the SWAMO platform [204] aims to provide an hierarchical architecture for intelligent agents which caters for planning and negotiating for resources. Their aim is to create an ontology that includes representation support for system components.

7.3.2 Non-agent based approaches

The Open Geospatial Consortium (OGC) has made great strides in their standardisation efforts. These include standard data formats and data encodings, standard software development APIs, such as the

GeoAPI, as well as standards for geospatial services such as the Web Mapping Service (WMS), Web Feature Service (WFS), and more recently within the Sensor Web Enablement (SWE) initiative, the Sensor Observation Service (SOS), and the Sensor Alerting Service (SAS). While SWAP attempts to address the drawbacks of SWE it also allows for the incorporation of SWE services, OGC data formats and application APIs.

SWE lacks formal semantics which limits service discovery, service composition and data integration. Furthermore, the SWE framework provides a limited methodology for application development and minimal support for application deployment. Service providers hide complex application logic behind OGC services. For example, in the SWE approach for detecting wildfire [189], a Sensor Observation Service (SOS) was used to extract hotspots from satellite image data sets and provide hotspot features. The data retrieval and hotspot detection process is hidden behind the SOS interface. Only by reading the system documentation would developers discover the algorithm and, if it is not specified in the derived data, the sensor data set that was used to produce the data. Thus complex application components, e.g. sensor data providers and data processing algorithms may be hidden behind the SOS interface. This together with the lack of semantics in service descriptions drastically limits the degree of reuse as well as the maintenance and changeability of the components that make up a service. New services must be built from scratch even if they incorporate similar data sources and algorithms used within existing services. This duplication of efforts will be compounded as the number and complexity of services increases. Furthermore, as a limited methodology is provided for developing SWE end-user applications, services as well as applications that use these services will be developed on a custom and adhoc basis and often must be manually upgraded to incorporate new services or changes to existing services.

There have been a number of initiatives to add semantics to OGC data standards and services. The Geospatial Semantic Web Interoperability Experiment [124] attempted to augment WFS with a semantic query capability, by defining a set of OWL ontologies. Lutz and Kolas [128] attempt to extend the limitations of OWL by incorporating rules which allow for more richer discovery and querying of distributed OGC data services. However, most OGC based approaches [124, 127, 128] tend to encode OGC standards, e.g. the Geographical Markup Language (GML) into OWL or layer semantics over OGC services, with the focus being on discovering and querying heterogenous distributed sensor data sets. Very little attention is given to representing the actions, i.e. the information extraction, filtering algorithms and models for processing this data.

Agent models offer a high level of disaggregation and address scale concerns at a conceptual and functional level [176]. While the importance and benefits of using agents as design abstractions has been highlighted [176], most OGC based approaches still take a service based approach, with the focus still on data integration [117] by augmenting OGC data provisioning services with semantic capabilities [124, 128]. While these approaches show the benefits of incorporating ontologies for improving discovery and querying of distributed and heterogenous sensor data sets, they do not provide semantics for discovering, selecting, invoking and composing other types of geospatial services (besides data provision) which is required for a geospatial web [117]. Some OGC based initiatives [113, 121] have specifically addressed the issue of service composition, and as done in this research also use OWL-S for representing executable workflows or geospatial service chains. More recently Zaharia et al. [211] have used the WSMO framework for workflow representation.

SWAP provides a comprehensive and coherent semantic framework to allow for representing conceptual aspects of data as well as to represent system entities to address this. None of the OGC based approaches [113, 121, 124, 127, 128] explicitly delineate space, time, theme and uncertainty. These approaches also do not provide an explicit methodology for designing and deploying services and applications. SWAP proposes a next generation ontology driven multi-agent architecture, but also provides a mechanism to incorporate OGC data standards and services. The ontology infrastructure is more comprehensive and bridges the gap between system and conceptual entities. It provides a stronger modeling and design framework for developing and designing Sensor Web services and applications, and aims to cater for a wide Sensor Web community comprising of thousands of different organisations and agents.

7.3.3 Other related work

The ontology and reasoning infrastructure is an essential part of the SWAP framework and is one of the main contributions of this work. The different conceptual ontologies, i.e. *swap-theme*, *swap-space*, *swap-time* and *swap-uncertainty* provide a comprehensive framework for representing entities in the real world. The different technical ontologies, i.e. *swap-data*, *swap-task* and *swap-agent*, provide concepts for representing software system entities. To the author's knowledge there is no other implemented framework that provides such comprehensive representational support.

The SOUPA ontology used in COBRA [45] provides support for representing geospatial data. It uses DAML-Time [94] for representing temporal concepts and OpenCyc Spatial Ontologies [169] and

Regional Connection Calculus (RCC) [165] and also provide independent rule based temporal and spatial reasoners. The decision to use separate rule based reasoners for space and time was based on the approach used in COBRA. This has two main advantages. Firstly, there is vast performance increase, by reasoning about and answering queries on a single conceptual dimension, rather than reasoning about all dimensions at once. Secondly, it eases the complexity of representing real world entities, by considering the entity separately along each dimension. However, SOUPA is limited in terms of its representational support. From a technical viewpoint, it does not provide for representing system entities, such as agents and tasks, and from a conceptual viewpoint, does not deal with uncertainty. It also has limited support for geospatial thematic representation as it concentrates on pervasive context-aware systems.

A more recent work by Perry et al. [161] also proposes the separation of ontologies for space, time and theme. However, their model lacks support for representing uncertainty or system entities. Even though they demonstrate an application case study, the emphasis is on the use of these ontologies, rather than providing a design and development methodology for building Sensor Web applications [88]. Furthermore, the approach does not incorporate OGC services and standards.

The incorporation of uncertainty is a key feature of the SWAP ontology infrastructure. The approach builds on BayesOWL[57, 58], which proposes a generic approach using an an OWL ontology to capture and perform inferencing on a Bayesian Network (BN). The uncertainty system, described in chapter 5), shows how Bayesian Networks can be incorporated and used in an ODMAS for the Sensor Web. Uncertainty resulting from weak theories, sensor inaccuracies or location specific anomalies can be captured and incorporated into a BN. The thematic, spatial and temporal aspects of event variables are described using the relevant SWAP ontologies. The approach distinguishes between observable events which are dynamically populated using observations captured in the system, and inferenced events, which are produced after inferencing is performed on the Bayesian Network. Inferred events are transformed into ontology instances and are specified using the top level ontologies. They can thus be interpreted, placed into context using the top level ontologies, integrated with other information and incorporated within individual agents or within end user applications.

7.4 LIMITATIONS AND FUTURE WORK

There are many aspects of the system that could be extended or require further investigation.

7.4.1 Creating additional SWAP applications

Two applications that involved information extraction from satellite imagery were modelled and implemented. While the wildfire detection was relatively simple and served to demonstrate the use of the framework, the informal settlement detection application was more reflective of the complexity of a real world satellite image processing application. It uses observations from different spectral frequency bands, incorporates multiple image processing algorithms and a classification algorithm to classify informal settlements. The framework was found to be adequate to model and implement both applications. These applications shows the adequacy of the framework for developing and deploying dynamic satellite image processing applications. In both applications scenarios for reusing and reconfiguring agent services were described. However, these applications were developed independently and in parallel. Agents were not reused across the applications. Additional applications must be implemented, ideally by independent developers, to provide further measures for reuse and interoperability.

7.4.2 Extending uncertainty and supporting quality of service

The representation and management of uncertain knowledge is an important aspect of the Sensor Web. The proposed model shows how Bayesian Networks can be incorporated and used to reason about cause and effect relations. However, the model assumes that all events represented in a Bayesian Network occur at the same time and space. The extension of the model to capture influences between past, current and future events and events occurring at different locations requires further investigation.

The model does not provide support representing the reliability or the quality of services. The response times of services may differ. The availability of local computational resources, external factors such as network latency, network bandwidth, and the response of other agents required for it to perform its function may affect the response time. The reputation of the hosting organisation may be an indicator of the quality of a service. Services offered by well known organisations such as NASA and ESA may be more trusted than those offered by less well known organisations. The extension of the model to allow for describing the quality and reliability of services requires further investigation.

7.4.3 Agent mobility and security

SWAP does not support agent mobility [41, 143]. Depending on the availability of network bandwidth and local computational resources an agent could migrate to a remote agent platform. Agent mobility can be used to increase agent performance and robustness. Agents can migrate at runtime from computing nodes with high resource utilisation to nodes with low resource utilisation. The current framework does not implement a security model. Security features such as authentication and verification is an important aspect that requires further investigation.

7.4.4 Automation

An ODMAS presents great potential for automating information extraction. When scientists pose queries to the system, there may not be an existing resource that satisfies the query. An agent could be tasked to identify potentially related resources, and use optimisation and scheduling techniques to assemble resources into possible workflows that may satisfy the query. The agent can also be tasked to monitor for and incorporate new resources as they become available. Efforts to automate service composition using OWL-S has been investigated [130], even in the geospatial domain [210]. Since workflows are composed manually these approaches could be investigated and incorporated to obtain some degree of automation for workflow composition.

Machine learning techniques can be incorporated within agents to construct models from historical analysis of data and use these as a basis for predicting future events. Such models could be updated when new data is available. Agents can learn to find aberrant patterns prior to certain phenomena, e.g. detect unusual patterns of events prior to the occurrence of a flood. This knowledge can be used to discover new relations between phenomena and to derive improved prediction models.

7.4.5 Tool support

Various tools can be developed to ease user interaction with SWAP. For example, a scientific desktop that allow scientists to construct and visual workflows. This could include support for service navigation, workflow construction, manipulation, execution and deployment. While examples of data and agent services queries were demonstrated this requires further investigation. A query tool could be created to query sensor data sets or service offerings.

7.5 IMPACT OF RESEARCH

This research describes a practical approach for an Ontology Driven Multi-Agent Sensor Web. It investigates the issues and challenges for designing, developing and deploying ontology driven agent based services and applications. The incorporation and tight integration of ontologies and agents into the Sensor Web has the potential to provide to forever change the way in which geospatial data and knowledge is accessed and used. Some key benefits of using such an approach are described below.

Capturing, sharing and integrating data, knowledge and services A set of top level ontologies guide the development of new ontologies that describe the observations, algorithms, theories and prediction models which form the key service offerings on the Sensor Web. Domain modelers reuse concepts and relationships from the top level ontologies or introduce new concepts and relationships between these concepts. As these ontologies extend the same high level conceptual model and consist of unambiguous and logically consistent statements, ontology fragments, even those created by different users, can be shared, logically verified and to an extent, integrated and reused. This can facilitate dynamic data integration as well as dynamic service interoperability. There is also the potential to create a vast globally distributed and continuously evolving Sensor Web knowledge repository that will provide an invaluable tool for earth observation practitioners to capture and exchange knowledge. This knowledge repository can also provide a valuable resource for teaching programs within academic institutions.

Facilitate human collaboration and accelerate scientific discovery Often, innovative models and processes published in traditional scientific literature are not easily reproducible, for a number of technical reasons. Publishing complex processes and models online, facilitates immediate reuse by other scientists within other processes or models. Researchers from different countries and different organisations are able to pool resources and expertise to work on large scale and more complex research projects. Resources developed by individual researchers can be validated and tested by others with alternate parameters or alternate data sets. Resources developed at different partner organisations can be assembled into workflows to address complex problems, which can themselves be shared, executed with alternate resources, or incorporated into other workflows.

Managing information overload and system complexity Users will be increasingly overwhelmed by the volumes of available data and multitude of services offered on the Sensor Web. Users may lack the necessary technical expertise and time to interpret and analyse all types of data. An ODMAS can assist users with information overload as well as shield users from the complexity of the Sensor Web. For example, a user could pose a vague query using concepts from the ontology and the system can attempt to discover relevant agents that may potentially assist to fulfill the query. These agents can be used without requiring knowledge of the internal workings of the agent. Users can also reuse complex information extraction workflows without requiring an understanding of the component agents that make up the workflow. In this way, researchers can perform tasks without requiring expert knowledge of all data, processes and models stored in the system. By accessing the ontologies, users are better able to place retrieved data into context and may view the data at different levels of granularity. Additionally, an ODMAS can provide different layers of abstraction with different types of agents at each layer encapsulating specific functionality that is required at that layer. This allows for consistency and also eases the design and management of agents and agent based applications.

Managing dynamism and towards automation and intelligence Service offerings and data models will continuously change. Agent service offerings, interaction protocols and data models are explicitly specified in the ontologies. These ontologies form a shared and dynamic model that drives the operation of the system. Many reconfiguration tasks can be performed by modifying the appropriate ontologies. Ontology fragments created by different users can be logically verified, integrated and reused. As ontology statements are machine interpretable, agents could be deployed to discover new relationships or even new concepts by integrating ontology fragments created by different users. The approach can lead to the automation of various tasks including: service discovery, service interoperability, workflow composition, data integration, data analysis, information extraction and scientific experimentation. The incorporation of machine learning and pattern recognition techniques can assist in this regard.

Appendix A

THE SWAP ONTOLOGIES AND RULES

A.1 THE SWAP-THEME ONTOLOGY

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY swap-data.owl "http://masii.cs.ukzn.ac.za/swap/swap-data.owl">
  <!ENTITY swap-theme.owl "http://masii.cs.ukzn.ac.za/swap/swap-theme.owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&swap-theme.owl;"
         xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;">

<!-- Ontology Information -->
  <owl:Ontology rdf:about="">
    <owl:imports>
      <owl:Ontology rdf:about="&swap-data.owl;"/>
    </owl:imports>
  </owl:Ontology>

<!-- Classes -->
  <owl:Class rdf:about="#Observable">
    <rdfs:comment xml:lang="en">Something that can be observed, e.g. the speed (property) of the wind (entity)</rdfs:comment>
  </owl:Class>

  <owl:Class rdf:about="#ObservedEntity"/>
  <owl:Class rdf:about="#ObservedProperty"/>
  <owl:Class rdf:about="#ThematicUnits">
    <rdfs:subClassOf rdf:resource="&swap-data.owl;#Units"/>
  </owl:Class>

  <owl:Class rdf:about="&swap-data.owl;#Units"/>

<!-- Annotation Properties -->
  <owl:AnnotationProperty rdf:about="&rdfs;comment"/>

<!-- Object Properties -->
  <owl:ObjectProperty rdf:about="#hasThematicProperties"/>
  <owl:ObjectProperty rdf:about="#observesEntity">
    <rdfs:domain rdf:resource="#Observable"/>
    <rdfs:range rdf:resource="#ObservedEntity"/>
    <rdfs:subPropertyOf rdf:resource="#hasThematicProperties"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#observesProperty">
    <rdfs:domain rdf:resource="#Observable"/>
    <rdfs:range rdf:resource="#ObservedProperty"/>
    <rdfs:subPropertyOf rdf:resource="#hasThematicProperties"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

A.2 THE SPATIAL ONTOLOGY AND RULES

A.2.1 The swap-space ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY ObjectList.owl "http://www.daml.org/services/owl-s/1.1/generic/ObjectList.owl">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY swap-data.owl "http://masii.cs.ukzn.ac.za/swap/swap-data.owl">
  <!ENTITY swap-space "http://masii.cs.ukzn.ac.za/swap/swap-space.owl#">
  <!ENTITY swap-space.owl "http://masii.cs.ukzn.ac.za/swap/swap-space.owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&swap-space.owl;"
         xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;"
         xmlns:swap-space="&swap-space;">

<!-- Ontology Information -->
  <owl:Ontology rdf:about="">
    <owl:imports>
      <owl:Ontology rdf:about="&swap-data.owl;"/>
    </owl:imports>
    <owl:imports>
      <owl:Ontology rdf:about="&ObjectList.owl;"/>
    </owl:imports>
  </owl:Ontology>

<!-- Classes -->
  <owl:Class rdf:about="#Geometry">
    <rdfs:subClassOf rdf:resource="#Location"/>
  </owl:Class>

  <owl:Class rdf:about="#LatLonBoundingBox">
    <rdfs:subClassOf rdf:resource="#Location"/>
  </owl:Class>

  <owl:Class rdf:about="#Line">
    <rdfs:subClassOf rdf:resource="#Geometry"/>
  </owl:Class>

  <owl:Class rdf:about="#Location">
    <rdfs:comment xml:lang="en">This has been replaced by SpatialRegion</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#SpatialEntity"/>
  </owl:Class>

  <owl:Class rdf:about="#Point">
    <rdfs:subClassOf rdf:resource="#Geometry"/>
  </owl:Class>

  <owl:Class rdf:about="#PointCoordinate">
    <rdfs:subClassOf rdf:resource="#Location"/>
  </owl:Class>

  <owl:Class rdf:about="#Polygon">
    <rdfs:subClassOf rdf:resource="#Geometry"/>
  </owl:Class>

  <owl:Class rdf:about="#SpatialEntity"/>
  <owl:Class rdf:about="#SpatialProjection">
    <rdfs:subClassOf rdf:resource="#SpatialEntity"/>
  </owl:Class>

  <owl:Class rdf:about="#SpatialReferenceSystem">
    <rdfs:subClassOf rdf:resource="#SpatialEntity"/>
  </owl:Class>

  <owl:Class rdf:about="#SpatialResolution">
    <rdfs:comment rdf:datatype="&xsd;string">the area to which one pixel corresponds</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#SpatialEntity"/>
  </owl:Class>

  <owl:Class rdf:about="#SpatialThing">
    <rdfs:comment rdf:datatype="&xsd;string">something that occupies a spatial region, i.e. has some geometry</rdfs:comment>
  </owl:Class>

  <owl:Class rdf:about="#SpatialUnits">
    <rdfs:subClassOf rdf:resource="&swap-data.owl;#Units"/>
  </owl:Class>

  <owl:Class rdf:about="&swap-data.owl;#NumericInterval"/>
  <owl:Class rdf:about="&swap-data.owl;#SingleValue"/>
  <owl:Class rdf:about="&swap-data.owl;#Units"/>
  <owl:Class rdf:about="&ObjectList.owl;#List"/>

<!-- Datatypes -->
  <rdfs:Datatype rdf:about="&xsd;double"/>

<!-- Annotation Properties -->
  <owl:AnnotationProperty rdf:about="&rdfs;comment"/>
```

```
<!-- Datatype Properties -->
  <owl:DatatypeProperty rdf:about="#has2DSpatialSimilarityThreshold">
    <rdfs:comment xml:lang="en">consider locations to be similar</rdfs:comment>
    <rdfs:subPropertyOf rdf:resource="#hasSpatialConstraints"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasCommonName">
    <rdfs:domain rdf:resource="#SpatialThing"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasLatitude">
    <rdfs:domain rdf:resource="#PointCoordinate"/>
    <rdfs:range rdf:resource="&xsd;double"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasLongitude">
    <rdfs:domain rdf:resource="#PointCoordinate"/>
    <rdfs:range rdf:resource="&xsd;double"/>
  </owl:DatatypeProperty>

<!-- Object Properties -->
  <owl:ObjectProperty rdf:about="#contains">
    <rdfs:subPropertyOf rdf:resource="#hasSpatialRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#crosses">
    <rdfs:subPropertyOf rdf:resource="#hasSpatialRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#disjointWith">
    <rdfs:subPropertyOf rdf:resource="#hasSpatialRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#equals">
    <rdfs:subPropertyOf rdf:resource="#hasSpatialRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasArea">
    <rdfs:range rdf:resource="&swap-data.owl;#SingleValue"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasBoundaryCoordinates">
    <rdfs:domain rdf:resource="#Polygon"/>
    <rdfs:range rdf:resource="&ObjectList.owl;#List"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasCoordinate">
    <rdfs:domain rdf:resource="#Point"/>
    <rdfs:range rdf:resource="#PointCoordinate"/>
  </owl:ObjectProperty>

  <owl:FunctionalProperty rdf:about="#hasLatitudeInterval">
    <rdf:type rdf:resource="&owl;ObjectProperty"/>
    <rdfs:domain rdf:resource="#LatLonBoundingBox"/>
    <rdfs:range rdf:resource="&swap-data.owl;#NumericInterval"/>
  </owl:FunctionalProperty>

  <owl:ObjectProperty rdf:about="#hasLineCoordinates">
    <rdfs:domain rdf:resource="#Line"/>
    <rdfs:range rdf:resource="&ObjectList.owl;#List"/>
  </owl:ObjectProperty>

  <owl:FunctionalProperty rdf:about="#hasLongitudeInterval">
    <rdf:type rdf:resource="&owl;ObjectProperty"/>
    <rdfs:domain rdf:resource="#LatLonBoundingBox"/>
    <rdfs:range rdf:resource="&swap-data.owl;#NumericInterval"/>
  </owl:FunctionalProperty>

  <owl:ObjectProperty rdf:about="#hasPosition">
    <rdfs:range rdf:resource="#PointCoordinate"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasSpatialConstraints"/>
  <owl:ObjectProperty rdf:about="#hasSpatialProperties">
    <rdfs:range rdf:resource="#SpatialEntity"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasSpatialRelation">
    <rdfs:domain rdf:resource="#SpatialThing"/>
    <rdfs:range rdf:resource="#SpatialThing"/>
    <rdfs:subPropertyOf rdf:resource="#hasSpatialProperties"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasSpatialResolution">
    <rdfs:range rdf:resource="#SpatialResolution"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasSpatialThing">
    <rdfs:range rdf:resource="#SpatialThing"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#intersects">
    <rdfs:subPropertyOf rdf:resource="#hasSpatialRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#locatedAt">
    <rdfs:range rdf:resource="#Location"/>
    <rdfs:subPropertyOf rdf:resource="#hasSpatialProperties"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#overlapsWith">
    <rdfs:subPropertyOf rdf:resource="#hasSpatialRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#producedByProjection">
    <rdfs:range rdf:resource="#SpatialProjection"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#touches">
    <rdfs:subPropertyOf rdf:resource="#hasSpatialRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#usesReferenceSystem">
    <rdfs:domain rdf:resource="#Geometry"/>
    <rdfs:range rdf:resource="#SpatialReferenceSystem"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#within">
    <rdfs:subPropertyOf rdf:resource="#hasSpatialRelation"/>
  </owl:ObjectProperty>

<!-- Instances -->
  <swap-space:SpatialUnits rdf:about="#kilometer"/>
  <swap-space:SpatialUnits rdf:about="#square-kilometer"/>
</rdf:RDF>
```

A.2.2 Spatial rules

```
(?x rdf:type spc:SpatialThing)<-(?x spc:locatedAt ?y).

(?x rdf:type spc:SpatialThing) -> (?x spc:contains ?x).
(?x rdf:type spc:SpatialThing) -> (?x spc:equals ?x).
(?x rdf:type spc:SpatialThing) -> (?x spc:intersects ?x).
(?x rdf:type spc:SpatialThing) -> (?x spc:within ?x).

(?x spc:contains ?y) <-
(?x rdf:type spc:SpatialThing) (?y rdf:type spc:SpatialThing)
(?x spc:locatedAt ?xExt) (?y spc:locatedAt ?yExt)
spatiallyContains(?xExt,?yExt).

(?x spc:within ?y) <- (?y spc:contains ?x).

(?x spc:crosses ?y) <-
(?x rdf:type spc:SpatialThing) (?y rdf:type spc:SpatialThing)
(?x spc:locatedAt ?xExt) (?y spc:locatedAt ?yExt)
spatiallyCrosses(?xExt,?yExt).

(?x spc:disjointWith ?y) <-
(?x rdf:type spc:SpatialThing) (?y rdf:type spc:SpatialThing)
(?x spc:locatedAt ?xExt) (?y spc:locatedAt ?yExt)
spatiallyDisjointWith(?xExt,?yExt).


(?x spc:equals ?y) <-
(?x rdf:type spc:SpatialThing) (?y rdf:type spc:SpatialThing)
(?x spc:locatedAt ?xExt) (?y spc:locatedAt ?yExt)
spatiallyEquals(?xExt,?yExt).


(?x spc:intersects ?y) <-
(?x rdf:type spc:SpatialThing) (?y rdf:type spc:SpatialThing)
(?x spc:locatedAt ?xExt) (?y spc:locatedAt ?yExt)
spatiallyIntersects(?xExt,?yExt).

(?x spc:overlapsWith ?y) <-
(?x rdf:type spc:SpatialThing) (?y rdf:type spc:SpatialThing)
(?x spc:locatedAt ?xExt) (?y spc:locatedAt ?yExt)
spatiallyOverlapsWith(?xExt,?yExt).
```

A.3 THE TEMPORAL ONTOLOGY AND RULES

A.3.1 The swap-time ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY swap-data.owl "http://masii.cs.ukzn.ac.za/swap/swap-data.owl">
  <!ENTITY swap-time.owl "http://masii.cs.ukzn.ac.za/swap/swap-time.owl">
  <!ENTITY time-entry "http://www.isi.edu/˜pan/damltime/time-entry.owl#">
  <!ENTITY time-entry.owl "http://www.isi.edu/˜pan/damltime/time-entry.owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&swap-time.owl;"
         xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;"
         xmlns:time-entry="&time-entry;">

<!-- Ontology Information -->
  <owl:Ontology rdf:about="">
    <owl:imports>
      <owl:Ontology rdf:about="&time-entry.owl;"/>
    </owl:imports>
    <owl:imports>
      <owl:Ontology rdf:about="&swap-data.owl;"/>
    </owl:imports>
  </owl:Ontology>

<!-- Classes -->
  <owl:Class rdf:about="#PreviousRecurringInstantEvent">
    <rdfs:subClassOf rdf:resource="&time-entry;InstantEvent"/>
  </owl:Class>

  <owl:Class rdf:about="#RecurringInstants">
    <rdfs:subClassOf rdf:resource="&time-entry;TemporalEntity"/>
  </owl:Class>

  <owl:Class rdf:about="#TemporalResolution"/>
  <owl:Class rdf:about="&swap-data.owl;#Units"/>
  <owl:Class rdf:about="&time-entry;Instant"/>
  <owl:Class rdf:about="&time-entry;InstantEvent"/>
  <owl:Class rdf:about="&time-entry;TemporalEntity"/>
  <owl:Class rdf:about="&time-entry;TemporalThing"/>
  <owl:Class rdf:about="&time-entry;TemporalUnit">
    <rdfs:subClassOf rdf:resource="&swap-data.owl;#Units"/>
  </owl:Class>

<!-- Datatypes -->
  <rdfs:Datatype rdf:about="&xsd;duration"/>

<!-- Datatype Properties -->
  <owl:DatatypeProperty rdf:about="#hasTemporalGap">
    <rdfs:domain rdf:resource="#TemporalResolution"/>
    <rdfs:range rdf:resource="&xsd;duration"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="&time-entry;durationDescriptionDataType"/>

<!-- Object Properties -->
  <owl:ObjectProperty rdf:about="#followedBy">
    <rdfs:domain rdf:resource="#PreviousRecurringInstantEvent"/>
    <rdfs:range rdf:resource="&time-entry;InstantEvent"/>
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
    <owl:inverseOf rdf:resource="#follows"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#follows">
    <rdfs:domain rdf:resource="#PreviousRecurringInstantEvent"/>
    <rdfs:range rdf:resource="&time-entry;InstantEvent"/>
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
    <owl:inverseOf rdf:resource="#followedBy"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasInstant"/>
  <owl:ObjectProperty rdf:about="#hasTemporalConstraint"/>
  <owl:ObjectProperty rdf:about="#hasTemporalProperty"/>
  <owl:ObjectProperty rdf:about="#hasTemporalRelation"/>
  <owl:ObjectProperty rdf:about="#hasTemporalResolution">
    <rdfs:range rdf:resource="#TemporalResolution"/>
    <rdfs:subPropertyOf rdf:resource="#hasTemporalProperty"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;after">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;before">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="&time-entry;durationDescriptionOf"/>
  <owl:ObjectProperty rdf:about="&time-entry;inside">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;intAfter">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;intBefore">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;intContains">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;intDuring">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;intEquals">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;intFinishedBy">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;intFinishes">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;intMeets">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;intMetBy">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;intOverlappedBy">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;intOverlaps">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;intStartedBy">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;intStarts">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;nonoverlap">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&time-entry;startsOrDuring">
    <rdfs:subPropertyOf rdf:resource="#hasTemporalRelation"/>
  </owl:ObjectProperty>

<!-- Instances -->
  <time-entry:Instant rdf:about="#future"/>
  <time-entry:Instant rdf:about="#past"/>
  <time-entry:Instant rdf:about="#present"/>
</rdf:RDF>
```

A.3.2 Temporal rules

```
(?x tme:begins ?x) <- (?x rdf:type tme:InstantThing).
(?x tme:ends ?x) <- (?x rdf:type tme:InstantThing).

(?x tme:begins ?y) <-
(?x rdf:type tme:InstantThing), (?x tme:inCalendarClockDataType ?timeX),
(?y rdf:type tme:InstantThing), (?y tme:inCalendarClockDataType ?timeY),
equal(?timeX,?timeY).

(?x tme:ends ?y) <-
(?x rdf:type tme:InstantThing), (?x tme:inCalendarClockDataType ?timeX),
(?y rdf:type tme:InstantThing), (?y tme:inCalendarClockDataType ?timeY),
equal(?timeX,?timeY).

(?x rdf:type tme:IntervalThing),
(?x tme:from ?fromTime) -> (?x tme:begins ?fromTime).

(?x rdf:type tme:IntervalThing),
(?x tme:to ?toTime) -> (?x tme:ends ?toTime).

(?x tme:before ?y) <-
     (?x rdf:type tme:InstantThing), (?x tme:inCalendarClockDataType ?timeX),
     (?y rdf:type tme:InstantThing), (?y tme:inCalendarClockDataType ?timeY),
     lessThan(?timeX,?timeY).

(?x tme:intBefore ?y) <-
    (?x rdf:type tme:IntervalThing), (?x tme:ends ?endsX),
    (?y rdf:type tme:IntervalThing), (?y tme:begins ?beginsY),
     (?endsX tme:before ?beginsY).

(?x tme:before ?y) <-
     (?x rdf:type tme:InstantThing),
     (?y rdf:type tme:IntervalThing), (?y tme:begins ?beginsY),
     (?x tme:before ?beginsY).

(?x tme:after ?y) <- (?y tme:before ?x).

(?x tme:inside ?y) <-
(?x rdf:type tme:InstantThing),
(?y rdf:type tme:IntervalThing),
(?y tme:begins ?beginsY), (?y tme:ends ?endsY),
(?beginsY tme:before ?x), (?x tme:before ?endsY).

(?x rdf:type tme:ProperIntervalThing) <-
(?x rdf:type tme:IntervalThing),
(?x tme:begins ?beginsX), (?x tme:ends ?endsX),
(?beginsX tme:before ?endsX).

(?x tme:intEquals ?y) <-
(?x rdf:type tme:ProperIntervalThing),
(?y rdf:type tme:ProperIntervalThing),
(?x tme:begins ?beginsX), (?x tme:ends ?endsX),
(?y tme:begins ?beginsY), (?y tme:ends ?endsY),
(?beginsX tme:inCalendarClockDataType ?bxTime), (?endsX tme:inCalendarClockDataType ?exTime),
(?beginsY tme:inCalendarClockDataType ?byTime), (?endsY tme:inCalendarClockDataType ?eyTime),
equal(?bxTime,?byTime),
equal(?exTime,?eyTime).

(?x tme:intMeets ?y) <-
(?x rdf:type tme:ProperIntervalThing),
(?y rdf:type tme:ProperIntervalThing),
(?x tme:ends ?endsX), (?endsX tme:inCalendarClockDataType ?exTime),
(?y tme:begins ?beginsY), (?beginsY tme:inCalendarClockDataType ?byTime),
equal(?exTime,?byTime).

(?x tme:intMetBy ?y) <- (?y tme:intMeets ?x).

(?x tme:intOverlaps ?y) <-
(?x rdf:type tme:ProperIntervalThing),
(?y rdf:type tme:ProperIntervalThing),
(?x tme:begins ?beginsX), (?x tme:ends ?endsX),
(?y tme:begins ?beginsY), (?y tme:ends ?endsY),
(?beginsY tme:before ?endsX),
(?beginsX tme:before ?beginsY),
(?endsX tme:before ?endsY).

(?x tme:intOverlappedBy ?y) <- (?y tme:IntOverlaps ?x).

(?x tme:intStarts ?y) <-
(?x rdf:type tme:ProperIntervalThing),
(?y rdf:type tme:ProperIntervalThing),
(?x tme:begins ?beginsX), (?x tme:ends ?endsX),
(?y tme:begins ?beginsY), (?y tme:ends ?endsY),
(?beginsX tme:inCalendarClockDataType ?bxTime),
(?beginsY tme:inCalendarClockDataType ?byTime),
equal(?bxTime,?byTime),
(?endsX tme:before ?endsY).

(?x tme:intStartedBy ?y) <- (?y tme:intStarts ?x).

(?x tme:intDuring ?y) <-
(?x rdf:type tme:ProperIntervalThing),
(?y rdf:type tme:ProperIntervalThing),
(?x tme:begins ?beginsX), (?x tme:ends ?endsX),
(?y tme:begins ?beginsY), (?y tme:ends ?endsY),
(?beginsY tme:before ?beginsX),
(?endsY tme:after ?endsX).

(?x tme:intContains ?y) <- (?y tme:intDuring ?x).

(?x tme:intFinishes ?y) <-
(?x rdf:type tme:ProperIntervalThing),
(?y rdf:type tme:ProperIntervalThing),
(?x tme:begins ?beginsX), (?x tme:ends ?endsX),
(?y tme:begins ?beginsY), (?y tme:ends ?endsY),
(?endsX tme:inCalendarClockDataType ?exTime),
(?endsY tme:inCalendarClockDataType ?eyTime),
equal(?exTime,?eyTime),
(?beginsY tme:before ?beginsX).

(?x tme:intFinishedBy ?y) <- (?y tme:intFinishes ?x).

(?x tme:StartsOrDuring ?y) <- (?x tme:intStarts ?y).
(?x tme:StartsOrDuring ?y) <- (?x tme:intduring ?y).

(?x tme:nonoverlap ?y) <-
(?x rdf:type tme:ProperIntervalThing),
(?y rdf:type tme:ProperIntervalThing),
(?x tme:before ?y).

(?x tme:nonoverlap ?y) <-
(?x rdf:type tme:ProperIntervalThing),
(?y rdf:type tme:ProperIntervalThing),
(?x tme:after ?y).

(?x tme:nonoverlap ?y) <-
(?x rdf:type tme:ProperIntervalThing),
(?y rdf:type tme:ProperIntervalThing),
(?x tme:intMeets ?y).

(?x tme:nonoverlap ?y) <-
(?x rdf:type tme:ProperIntervalThing),
(?y rdf:type tme:ProperIntervalThing),
(?x tme:intMetBy ?y).
```

A.4 THE SWAP-UNCERTAINTY ONTOLOGY

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY swap-data.owl "http://masii.cs.ukzn.ac.za/swap/swap-data.owl">
  <!ENTITY swap-uncertainty "http://masii.cs.ukzn.ac.za/swap/swap-uncertainty.owl#">
  <!ENTITY swap-uncertainty.owl "http://masii.cs.ukzn.ac.za/swap/swap-uncertainty.owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&swap-uncertainty.owl;"
         xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;"
         xmlns:swap-uncertainty="&swap-uncertainty;">

<!-- Ontology Information -->
  <owl:Ontology rdf:about=""
                owl:versionInfo="v1.0">
    <owl:imports>
      <owl:Ontology rdf:about="&swap-data.owl;"/>
    </owl:imports>
  </owl:Ontology>

<!-- Classes -->
  <owl:Class rdf:about="#BayesianNetwork"/>
  <owl:Class rdf:about="#CondProb">
    <rdfs:subClassOf rdf:resource="#ProbObj"/>
    <owl:disjointWith rdf:resource="#PriorProb"/>
  </owl:Class>

  <owl:Class rdf:about="#Condition"/>
  <owl:Class rdf:about="#DiscreteBooleanState">
    <rdfs:subClassOf rdf:resource="#DiscreteState"/>
  </owl:Class>

  <owl:Class rdf:about="#DiscreteRangeState">
    <rdfs:subClassOf rdf:resource="#DiscreteState"/>
    <rdfs:subClassOf rdf:resource="http://masii.cs.ukzn.ac.za/swap/swap-data.owl#NumericInterval"/>
  </owl:Class>

  <owl:Class rdf:about="#DiscreteState">
    <rdfs:subClassOf rdf:resource="#State"/>
  </owl:Class>

  <owl:Class rdf:about="#InferredVariable">
    <rdfs:subClassOf rdf:resource="#Variable"/>
  </owl:Class>

  <owl:Class rdf:about="#ObservationVariable">
    <rdfs:subClassOf rdf:resource="#Variable"/>
  </owl:Class>

  <owl:Class rdf:about="#PostProb">
    <rdfs:subClassOf rdf:resource="#ProbObj"/>
  </owl:Class>

  <owl:Class rdf:about="#PriorProb">
    <rdfs:subClassOf rdf:resource="#ProbObj"/>
    <owl:disjointWith rdf:resource="#CondProb"/>
  </owl:Class>

  <owl:Class rdf:about="#ProbObj"/>
  <owl:Class rdf:about="#SingleNumericState">
    <rdfs:subClassOf rdf:resource="#DiscreteState"/>
    <rdfs:subClassOf rdf:resource="http://masii.cs.ukzn.ac.za/swap/swap-data.owl#SingleValue"/>
  </owl:Class>

  <owl:Class rdf:about="#State"/>
  <owl:Class rdf:about="#Variable"/>
  <owl:Class rdf:about="http://masii.cs.ukzn.ac.za/swap/swap-data.owl#NumericInterval"/>
  <owl:Class rdf:about="http://masii.cs.ukzn.ac.za/swap/swap-data.owl#SingleValue"/>

<!-- Datatypes -->
  <rdfs:Datatype rdf:about="http://www.cs.umbc.edu/~zding1/owl/dt.xsd#between0and1"/>
  <rdfs:Datatype rdf:about="&xsd;anyURI"/>

<!-- Annotation Properties -->
  <owl:AnnotationProperty rdf:about="&owl;versionInfo"/>

<!-- Datatype Properties -->
  <owl:DatatypeProperty rdf:about="#hasClass">
    <rdfs:domain rdf:resource="#InferredVariable"/>
    <rdfs:range rdf:resource="&xsd;anyURI"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasProbValue">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="#ProbObj"/>
    <rdfs:range rdf:resource="http://www.cs.umbc.edu/~zding1/owl/dt.xsd#between0and1"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasProperty"/>
  <owl:DatatypeProperty rdf:about="#hasValueProperty">
    <rdfs:domain rdf:resource="#ObservationVariable"/>
  </owl:DatatypeProperty>

<!-- Object Properties -->
  <owl:ObjectProperty rdf:about="#hasCondition">
    <rdfs:domain rdf:resource="#CondProb"/>
    <rdfs:range rdf:resource="#Condition"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasInstanceValue">
    <rdfs:domain rdf:resource="#PostProb"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasPosteriorProb">
    <rdfs:range rdf:resource="#PostProb"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasState">
    <rdfs:domain rdf:resource="#Condition"/>
    <rdfs:domain rdf:resource="#ProbObj"/>
    <rdfs:domain rdf:resource="#Variable"/>
    <rdfs:range rdf:resource="#State"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasVariable">
    <rdfs:domain rdf:resource="#BayesianNetwork"/>
    <rdfs:domain rdf:resource="#Condition"/>
    <rdfs:domain rdf:resource="#ProbObj"/>
    <rdfs:range rdf:resource="#Variable"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#isInfluencedBy">
    <rdfs:domain rdf:resource="#Variable"/>
    <rdfs:range rdf:resource="#Variable"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="http://masii.cs.ukzn.ac.za/swap/swap-space.owl#hasSpatialConstraints">
    <rdfs:domain rdf:resource="#BayesianNetwork"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="http://masii.cs.ukzn.ac.za/swap/swap-theme.owl#observesEntity">
    <rdfs:domain rdf:resource="#ObservationVariable"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="http://masii.cs.ukzn.ac.za/swap/swap-theme.owl#observesProperty">
    <rdfs:domain rdf:resource="#ObservationVariable"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="http://masii.cs.ukzn.ac.za/swap/swap-time.owl#hasTemporalConstraint">
    <rdfs:domain rdf:resource="#BayesianNetwork"/>
  </owl:ObjectProperty>

<!-- Instances -->
  <swap-uncertainty:DiscreteBooleanState rdf:about="#False">
    <rdf:type rdf:resource="&owl;Thing"/>
  </swap-uncertainty:DiscreteBooleanState>

  <swap-uncertainty:DiscreteBooleanState rdf:about="#True">
    <rdf:type rdf:resource="&owl;Thing"/>
  </swap-uncertainty:DiscreteBooleanState>
</rdf:RDF>
```

A.5 THE TECHNICAL ONTOLOGIES

A.5.1 The swap-agent ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY eo-domain.owl "http://masii.cs.ukzn.ac.za/swap/eo-domain.owl">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY swap-agent.owl "http://masii.cs.ukzn.ac.za/swap/swap-agent.owl">
  <!ENTITY swap-data.owl "http://masii.cs.ukzn.ac.za/swap/swap-data.owl">
  <!ENTITY swap-space.owl "http://masii.cs.ukzn.ac.za/swap/swap-space.owl">
  <!ENTITY swap-time.owl "http://masii.cs.ukzn.ac.za/swap/swap-time.owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&swap-agent.owl;"
         xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;">

<!-- Ontology Information -->
  <owl:Ontology rdf:about="">
    <owl:imports>
      <owl:Ontology rdf:about="&eo-domain.owl;"/>
    </owl:imports>
    <owl:imports>
      <owl:Ontology rdf:about="&swap-data.owl;"/>
    </owl:imports>
    <owl:imports>
      <owl:Ontology rdf:about="&swap-space.owl;"/>
    </owl:imports>
    <owl:imports>
      <owl:Ontology rdf:about="&swap-time.owl;"/>
    </owl:imports>
  </owl:Ontology>

<!-- Classes -->
  <owl:Class rdf:about="#Action"/>
  <owl:Class rdf:about="#Agent"/>
  <owl:Class rdf:about="#AlertRequest">
    <rdfs:subClassOf rdf:resource="#Protocol"/>
  </owl:Class>

  <owl:Class rdf:about="#Application">
    <rdfs:subClassOf rdf:resource="#ServiceDescription"/>
  </owl:Class>

  <owl:Class rdf:about="#CapabilityRegistration">
    <rdfs:subClassOf rdf:resource="#Protocol"/>
  </owl:Class>

  <owl:Class rdf:about="#CapabilityRequest">
    <rdfs:subClassOf rdf:resource="#Protocol"/>
  </owl:Class>

  <owl:Class rdf:about="#DataProcessing">
    <rdfs:subClassOf rdf:resource="#Protocol"/>
  </owl:Class>

  <owl:Class rdf:about="#DataRequest">
    <rdfs:subClassOf rdf:resource="#Protocol"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom rdf:resource="#InformDataAction"/>
        <owl:onProperty rdf:resource="#hasResponse"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom rdf:resource="#QueryAction"/>
        <owl:onProperty rdf:resource="#hasRequest"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:about="#DataSet">
    <rdfs:subClassOf rdf:resource="#ServiceDescription"/>
  </owl:Class>

  <owl:Class rdf:about="#ExecuteWorkflowAction">
    <rdfs:subClassOf rdf:resource="#RequestAction"/>
  </owl:Class>

  <owl:Class rdf:about="#InformAlertAction">
    <rdfs:subClassOf rdf:resource="#ResponseAction"/>
  </owl:Class>

  <owl:Class rdf:about="#InformCapabilitiesAction">
    <rdfs:subClassOf rdf:resource="#ResponseAction"/>
  </owl:Class>

  <owl:Class rdf:about="#InformDataAction">
```

```
    <rdfs:subClassOf rdf:resource="#ResponseAction"/>
  </owl:Class>

  <owl:Class rdf:about="#InformPredictionAction">
    <rdfs:subClassOf rdf:resource="#ResponseAction"/>
  </owl:Class>

  <owl:Class rdf:about="#InformResultAction">
    <rdfs:subClassOf rdf:resource="#ResponseAction"/>
  </owl:Class>

  <owl:Class rdf:about="#InformServicesAction">
    <rdfs:subClassOf rdf:resource="#ResponseAction"/>
  </owl:Class>

  <owl:Class rdf:about="#Message"/>
  <owl:Class rdf:about="#Modeling">
    <rdfs:subClassOf rdf:resource="#ServiceDescription"/>
  </owl:Class>

  <owl:Class rdf:about="#PredictionRequest">
    <rdfs:subClassOf rdf:resource="#Protocol"/>
  </owl:Class>

  <owl:Class rdf:about="#ProcessAgentMapping"/>
  <owl:Class rdf:about="#ProcessDataAction">
    <rdfs:subClassOf rdf:resource="#RequestAction"/>
  </owl:Class>

  <owl:Class rdf:about="#Protocol"/>
  <owl:Class rdf:about="#QueryAction">
    <rdfs:subClassOf rdf:resource="#RequestAction"/>
  </owl:Class>

  <owl:Class rdf:about="#RegisterServiceAction">
    <rdfs:subClassOf rdf:resource="#RequestAction"/>
  </owl:Class>

  <owl:Class rdf:about="#RequestAction">
    <rdfs:subClassOf rdf:resource="#Action"/>
    <rdfs:subClassOf rdf:resource="#RequestAction"/>
  </owl:Class>

  <owl:Class rdf:about="#RequestAlertAction">
    <rdfs:subClassOf rdf:resource="#RequestAction"/>
  </owl:Class>

  <owl:Class rdf:about="#RequestCapabilitiesAction">
    <rdfs:subClassOf rdf:resource="#RequestAction"/>
  </owl:Class>

  <owl:Class rdf:about="#RequestPredictionAction">
    <rdfs:subClassOf rdf:resource="#RequestAction"/>
  </owl:Class>

  <owl:Class rdf:about="#ResponseAction">
    <rdfs:subClassOf rdf:resource="#Action"/>
  </owl:Class>

  <owl:Class rdf:about="#SearchDirectoryAction">
    <rdfs:subClassOf rdf:resource="#RequestAction"/>
  </owl:Class>

  <owl:Class rdf:about="#Service"/>
  <owl:Class rdf:about="#ServiceDescription"/>
  <owl:Class rdf:about="#ServiceQuery">
    <rdfs:subClassOf rdf:resource="#Protocol"/>
  </owl:Class>

  <owl:Class rdf:about="#ServiceRegistration">
    <rdfs:subClassOf rdf:resource="#Protocol"/>
  </owl:Class>

  <owl:Class rdf:about="#Tool">
    <rdfs:subClassOf rdf:resource="#ServiceDescription"/>
  </owl:Class>

  <owl:Class rdf:about="#ToolExecution">
    <rdfs:subClassOf rdf:resource="#Protocol"/>
  </owl:Class>

  <owl:Class rdf:about="#Workflow">
    <rdfs:subClassOf rdf:resource="#ServiceDescription"/>
  </owl:Class>

  <owl:Class rdf:about="#WorkflowExecution">
    <rdfs:subClassOf rdf:resource="#Protocol"/>
  </owl:Class>

<!-- Datatypes -->
  <rdfs:Datatype rdf:about="&xsd;anyURI"/>
  <rdfs:Datatype rdf:about="&xsd;dateTime"/>
  <rdfs:Datatype rdf:about="&xsd;long"/>
  <rdfs:Datatype rdf:about="&xsd;positiveInteger"/>
  <rdfs:Datatype rdf:about="&xsd;string"/>

<!-- Annotation Properties -->
  <owl:AnnotationProperty rdf:about="&rdfs;comment"/>
```

```
<!-- Datatype Properties -->
  <owl:DatatypeProperty rdf:about="#conversationId">
    <rdfs:domain rdf:resource="#Message"/>
    <rdfs:range rdf:resource="&xsd;long"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasAgentName">
    <rdfs:domain rdf:resource="#Agent"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasBlueband">
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasCommonName"/>
  <owl:DatatypeProperty rdf:about="#hasEndTime">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:range rdf:resource="&xsd;dateTime"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasGreenBand">
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasRedBand">
    <rdfs:comment rdf:datatype="&xsd;string">description of redband</rdfs:comment>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasSequenceNumber">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:range rdf:resource="&xsd;positiveInteger"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#type">
    <rdfs:domain rdf:resource="#Message"/>
    <rdfs:range rdf:resource="&xsd;anyURI"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="&swap-data.owl;#hasCharEncodedValue"/>
  <owl:DatatypeProperty rdf:about="&swap-data.owl;#hasType">
    <rdfs:domain rdf:resource="#DataSet"/>
  </owl:DatatypeProperty>

<!-- Object Properties -->
  <owl:ObjectProperty rdf:about="#attachment">
    <rdfs:domain rdf:resource="#Message"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#components"/>
  <owl:ObjectProperty rdf:about="#composedOf"/>
  <owl:ObjectProperty rdf:about="#content">
    <rdfs:domain rdf:resource="#Message"/>
    <rdfs:range rdf:resource="#Action"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#detectPhenomenon">
    <rdfs:domain rdf:resource="#Application"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#exhibitedBy"/>
  <owl:ObjectProperty rdf:about="#fromProcess"/>
  <owl:FunctionalProperty rdf:about="#hasAction">
    <rdf:type rdf:resource="&owl;ObjectProperty"/>
    <rdfs:range rdf:resource="#Action"/>
  </owl:FunctionalProperty>

  <owl:ObjectProperty rdf:about="#hasAgentMapping">
    <rdfs:range rdf:resource="#ProcessAgentMapping"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasAlert">
    <rdfs:domain rdf:resource="#InformAlertAction"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasCapabilities">
    <rdfs:domain rdf:resource="#InformCapabilitiesAction"/>
    <rdfs:range rdf:resource="#Service"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasDataDescription"/>
  <owl:ObjectProperty rdf:about="#hasDataQuery"/>
  <owl:ObjectProperty rdf:about="#hasDataSource"/>
  <owl:FunctionalProperty rdf:about="#hasDataStructure">
    <rdf:type rdf:resource="&owl;ObjectProperty"/>
  </owl:FunctionalProperty>

  <owl:ObjectProperty rdf:about="#hasIncreasedActivityIn"/>
  <owl:ObjectProperty rdf:about="#hasInput">
    <rdfs:domain rdf:resource="#ProcessDataAction"/>
    <rdfs:domain rdf:resource="#RequestPredictionAction"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasInputAction">
    <rdfs:range rdf:resource="#Action"/>
  </owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:about="#hasInputMapping">
  <rdfs:domain rdf:resource="#ProcessAgentMapping"/>
  <rdfs:range rdf:resource="#RequestAction"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasInputType">
  <rdfs:domain rdf:resource="#Tool"/>
</owl:ObjectProperty>

<owl:FunctionalProperty rdf:about="#hasObservable">
  <rdf:type rdf:resource="&owl;ObjectProperty"/>
</owl:FunctionalProperty>

<owl:ObjectProperty rdf:about="#hasObservationTime"/>
<owl:ObjectProperty rdf:about="#hasOutput">
  <rdfs:domain rdf:resource="#InformPredictionAction"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasOutputAction">
  <rdfs:range rdf:resource="#Action"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasOutputMapping">
  <rdfs:domain rdf:resource="#ProcessAgentMapping"/>
  <rdfs:range rdf:resource="#ResponseAction"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasOutputType">
  <rdfs:domain rdf:resource="#Tool"/>
</owl:ObjectProperty>

<owl:FunctionalProperty rdf:about="#hasRepresentation">
  <rdf:type rdf:resource="&owl;ObjectProperty"/>
</owl:FunctionalProperty>

<owl:ObjectProperty rdf:about="#hasRequest">
  <rdfs:domain rdf:resource="#Protocol"/>
  <rdfs:range rdf:resource="#RequestAction"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasResponse">
  <rdfs:domain rdf:resource="#Protocol"/>
  <rdfs:range rdf:resource="#ResponseAction"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasResult"/>
<owl:ObjectProperty rdf:about="#hasServiceDescriptionValue">
  <rdfs:domain rdf:resource="#SearchDirectoryAction"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasSpatialConstraints"/>
<owl:ObjectProperty rdf:about="#hasTemporalConstraints"/>
<owl:ObjectProperty rdf:about="#hasTemporalExtent"/>
<owl:ObjectProperty rdf:about="#hasWorkflow">
  <rdfs:domain rdf:resource="#ExecuteWorkflowAction"/>
  <rdfs:range rdf:resource="#Workflow"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#isDescribedBy">
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="#ServiceDescription"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#measuresphenomenan"/>
<owl:ObjectProperty rdf:about="#observedBy">
  <rdfs:comment rdf:datatype="&xsd;string">
  The instrument or person that generated this observation. [used-> procedure] in O&amp;M
  </rdfs:comment>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#providedBy">
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="#Agent"/>
  <owl:inverseOf rdf:resource="#provides"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#provides">
  <rdfs:domain rdf:resource="#Agent"/>
  <rdfs:domain rdf:resource="#RegisterServiceAction"/>
  <rdfs:range rdf:resource="#Service"/>
  <owl:inverseOf rdf:resource="#providedBy"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#recipient">
  <rdfs:domain rdf:resource="#Message"/>
  <rdfs:range rdf:resource="#Agent"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#sender">
  <rdfs:domain rdf:resource="#Message"/>
  <rdfs:range rdf:resource="#Agent"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#usesProtocol">
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="#Protocol"/>
</owl:ObjectProperty>
```

```
  <owl:ObjectProperty rdf:about="#usesWorkflowAgent">
    <rdfs:domain rdf:resource="#Application"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#usingDataSet">
    <rdfs:domain rdf:resource="#QueryAction"/>
    <rdfs:range rdf:resource="#DataSet"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&swap-data.owl;#hasData">
    <rdfs:domain rdf:resource="#InformDataAction"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&swap-data.owl;#hasValueStructure"/>
  <owl:ObjectProperty rdf:about="&swap-space.owl;#hasSpatialProperties">
    <rdfs:domain rdf:resource="#DataSet"/>
    <rdfs:domain rdf:resource="#QueryAction"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="&swap-space.owl;#hasSpatialRelation"/>
  <owl:ObjectProperty rdf:about="http://masii.cs.ukzn.ac.za/swap/swap-theme.owl#hasThematicProperties">
    <rdfs:domain rdf:resource="#DataSet"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="http://masii.cs.ukzn.ac.za/swap/swap-theme.owl#observesEntity"/>
  <owl:ObjectProperty rdf:about="&swap-time.owl;#hasTemporalProperty">
    <rdfs:domain rdf:resource="#DataSet"/>
    <rdfs:domain rdf:resource="#QueryAction"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="http://www.daml.org/services/owl-s/1.1/Service.owl#describedBy">
    <rdfs:domain rdf:resource="#ServiceDescription"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

A.5.2 The swap-task ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY Grounding.owl "file:///Grounding.owl">
  <!ENTITY Process.owl "file:///Process.owl">
  <!ENTITY Profile.owl "file:///Profile.owl">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY swap-task.owl "http://masii.cs.ukzn.ac.za/swap/swap-task.owl">
  <!ENTITY swrl.owl "http://www.w3.org/2003/11/swrl">
  <!ENTITY swrlb.owl "http://www.w3.org/2003/11/swrlb.owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&swap-task.owl;"
         xmlns:owl="&owl;"
         xmlns:rdf="&rdf;">

<!-- Ontology Information -->
  <owl:Ontology rdf:about="">
    <owl:imports>
      <owl:Ontology rdf:about="&Grounding.owl;"/>
    </owl:imports>
    <owl:imports>
      <owl:Ontology rdf:about="&Process.owl;"/>
    </owl:imports>
    <owl:imports>
      <owl:Ontology rdf:about="&Profile.owl;"/>
    </owl:imports>
    <owl:imports>
      <owl:Ontology rdf:about="&swrl.owl;"/>
    </owl:imports>
    <owl:imports>
      <owl:Ontology rdf:about="&swrlb.owl;"/>
    </owl:imports>
  </owl:Ontology>
</rdf:RDF>
```

A.5.3 The swap-data ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY ObjectList.owl "http://www.daml.org/services/owl-s/1.1/generic/ObjectList.owl">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY swap-data "http://masii.cs.ukzn.ac.za/swap/swap-data.owl#">
  <!ENTITY swap-data.owl "http://masii.cs.ukzn.ac.za/swap/swap-data.owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&swap-data.owl;"
         xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;"
         xmlns:swap-data="&swap-data;">

<!-- Ontology Information -->
  <owl:Ontology rdf:about="">
    <owl:imports>
      <owl:Ontology rdf:about="&ObjectList.owl;"/>
    </owl:imports>
  </owl:Ontology>

<!-- Classes -->
  <owl:Class rdf:about="#ArrayValue">
    <rdfs:subClassOf rdf:resource="#StructuredValue"/>
  </owl:Class>

  <owl:Class rdf:about="#Data"/>
  <owl:Class rdf:about="#DataSet"/>
  <owl:Class rdf:about="#FileValue">
    <rdfs:subClassOf rdf:resource="#StructuredValue"/>
  </owl:Class>

  <owl:Class rdf:about="#ImageFileValue">
    <rdfs:subClassOf rdf:resource="#FileValue"/>
  </owl:Class>

  <owl:Class rdf:about="#ImageFormats"/>
  <owl:Class rdf:about="#NumericInterval">
    <rdfs:subClassOf rdf:resource="#Value"/>
  </owl:Class>

  <owl:Class rdf:about="#PositionalValue">
    <rdfs:subClassOf rdf:resource="#Value"/>
  </owl:Class>

  <owl:Class rdf:about="#SingleValue">
    <rdfs:subClassOf rdf:resource="#Value"/>
  </owl:Class>

  <owl:Class rdf:about="#StructuredValue">
    <rdfs:subClassOf rdf:resource="#Value"/>
  </owl:Class>

  <owl:Class rdf:about="#Units">
    <rdfs:comment rdf:datatype="&xsd;string">use sweet units</rdfs:comment>
  </owl:Class>

  <owl:Class rdf:about="#Value"/>

<!-- Datatypes -->
  <rdfs:Datatype rdf:about="&xsd;anySimpleType"/>
  <rdfs:Datatype rdf:about="&xsd;anyURI"/>
  <rdfs:Datatype rdf:about="&xsd;double"/>
  <rdfs:Datatype rdf:about="&xsd;int"/>
  <rdfs:Datatype rdf:about="&xsd;string"/>

<!-- Annotation Properties -->
  <owl:AnnotationProperty rdf:about="&rdfs;comment"/>

<!-- Datatype Properties -->
  <owl:DatatypeProperty rdf:about="#hasCharEncodedValue">
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasColumns">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="#ArrayValue"/>
    <rdfs:range rdf:resource="&xsd;int"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasLocalPath">
    <rdfs:domain rdf:resource="#FileValue"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasLowerLimit">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="#NumericInterval"/>
    <rdfs:range rdf:resource="&xsd;double"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasNumericValue">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="#SingleValue"/>
    <rdfs:range rdf:resource="&xsd;anySimpleType"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasRows">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="#ArrayValue"/>
    <rdfs:range rdf:resource="&xsd;int"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasSpatialVariation">
    <rdfs:range rdf:resource="&xsd;double"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasType">
    <rdfs:range rdf:resource="&xsd;anyURI"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasURI">
    <rdfs:range rdf:resource="&xsd;anyURI"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#hasUpperLimit">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="#NumericInterval"/>
    <rdfs:range rdf:resource="&xsd;double"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#parameterType"/>
  <owl:DatatypeProperty rdf:about="#recurringIntValues">
    <rdfs:range rdf:resource="&xsd;int"/>
  </owl:DatatypeProperty>

<!-- Object Properties -->
  <owl:ObjectProperty rdf:about="#fromDataSet">
    <rdfs:range rdf:resource="#DataSet"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasData">
    <rdfs:range rdf:resource="#Data"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasImageFormat">
    <rdfs:domain rdf:resource="#ImageFileValue"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasSingleValue">
    <rdfs:range rdf:resource="#SingleValue"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasUnit">
    <rdfs:domain rdf:resource="#SingleValue"/>
    <rdfs:range rdf:resource="#Units"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#hasValueStructure">
    <rdfs:range rdf:resource="#StructuredValue"/>
  </owl:ObjectProperty>

<!-- Instances -->
  <swap-data:ImageFormats rdf:about="#GeoTiff"/>
</rdf:RDF>
```

BIBLIOGRAPHY

[1] 52 North [online]. Available from: http://52north.org.

[2] BNJ - Bayesian Networks tools in Java [online]. Available from: http://bnj.sourceforge.net.

[3] JENA - A Semantic Web Framework for Java [online]. Available from: http://jena.sourceforge.net.

[4] Plant Ontology Consortium (POC) [online]. Available from: http://www.plantontology.org.

[5] SNOMED CT [online]. Available from: http://www.ihtsdo.org/snomed-ct.

[6] The Java Agent Development Framework (JADE) [online]. Available from: http://jade.tilab.com.

[7] The Open Biomedical Ontologies [online]. Available from: http://obofoundry.org.

[8] The Open Geospatial Consortium [online]. Available from: http://www.opengeospatial.org.

[9] The Protege Ontology Editor [online]. Available from: http://protege.stanford.edu.

[10] The Suggested Upper Merged Ontology (SUMO) [online]. Available from: http://suo.ieee.org/SUO/SUMO/index.html.

[11] The Waikato Environment for Knowledge Analysis (WEKA) [online]. Available from: http://www.cs.waikato.ac.nz/ml/weka.

[12] The World Wide Web Consortium [online]. Available from: http://www.w3c.org.

[13] TopBraid Composer [online]. Available from: http://www.topbraidcomposer.com.

[14] WonderWeb Ontology Infrastructure for the Semantic Web [online]. Available from: http://wonderweb.semanticweb.org/.

[15] ABUGESSAISA, I. E. A., AND SIVERTUN, A. Ontological approach to modeling information systems. In *CIT '04: Proc. 4th International Conference on Computer and Information Technology* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 1122–1127.

[16] AGARWAL, P. Ontological considerations in GIScience. *International Journal of Geographical Information Science 19*, 5 (2005), 501–535.

[17] AGUILAR-PONCE, R., KUMAR, A., TECPANECATL-XIHUITL, J. L., AND BAYOUMI, M. A network of sensor-based framework for automated visual surveillance. *J. Netw. Comput. Appl. 30*, 3 (2007), 1244–1271.

[18] AKYILDIZ, I. F., MELODIA, T., AND CHOWDHURY, K. R. A survey on wireless multimedia sensor networks. *Comput. Networks 51*, 4 (2007), 921–960.

[19] ALANI, H. Position paper: ontology construction from online ontologies. In *WWW '06:Proc. 15th international conference on World Wide Web* (2006), ACM Press.

[20] ALLEN, J. F. Maintaining knowledge about temporal intervals. *Commun. ACM 26*, 11 (1983), 832–843.

[21] ATHANASIADIS, I. N. A review of agent-based systems applied in environmental informatics. In *MODSIM 2005: International Congress on Modelling and Simulation, Melbourne, Australia, Modelling and Simulation Society of Australia and New Zealand, December 2005* (2005).

[22] ATHANASIADIS, I. N., MILIS, M., MITKAS, P. A., AND MICHAELIDES, S. C. Abacus: A multi-agent system for meteorological radar data management and decision support. In *ISESS-05: International Symposium on Environmental Software Systems, Sesimbra, Portugal* (May, 2005).

[23] ATHANASIADIS, I. N., AND MITKAS, P. A. An agent-based intelligent environmental monitoring system. *Management of Environmental Quality 15*, 3 (2004), 238–249.

[24] BAADER, F., CALVANESE, D., MCGUINNESS, D., NARDI, D., AND PATEL-SCHNEIDER, P., Eds. *The Description Logic Handbook.* Cambridge University Press, 2003.

[25] BALAZINSKA, M., DESHPANDE, A., FRANKLIN, M. J., GIBBONS, P. B., GRAY, J., HANSEN, M., LIEBHOLD, M., NATH, S., SZALAY, A., AND TAO, V. Data management in the worldwide sensor web. *IEEE Pervasive Computing 6*, 2 (2007), 30–40.

[26] BEN-AMI, D., AND SHEHORY, O. A comparative evaluation of agent location mechanisms in large scale MAS. In *AAMAS '05: Proc. 4th international joint conference on Autonomous agents and multiagent systems* (2005), ACM Press.

[27] BERNERS-LEE, T., HENDLER, J., AND LASSILA, O. The semantic web. *Scientific American* (May 2001). Available from: `http://www.sciam.com`.

[28] BISWAS, P., AND PHOHA, S. A middleware-driven architecture for information dissemination in distributed sensor networks. In *Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004* (14-17 Dec. 2004).

[29] BISWAS, P. K., SCHMIEDEKAMP, M., AND PHOHA, S. An agent-oriented information processing architecture for sensor network applications. *Int. J. Ad Hoc Ubiquitous Comput. 1*, 3 (2006), 110–125.

[30] BORDINI, R., BRAUBACH, L., DASTANI, M., SEGHROUCHNI, A. E. F., GOMEZ-SANZ, J., LEITE, J., O'HARE, G., POKAHR, A., AND RICCI, A. A survey of programming languages and platforms for multi-agent systems. *Informatica 30*, 1 (2006), 33–44.

[31] BOTTS, M., PERCIVALL, G., REED, C., AND DAVIDSON, J. OGC sensor web enablement: Overview and high level architecture:version: 3.0. Tech. rep., Open Geospatial Consortium, OGC, December 28 2007. Available from: `http://portal.opengeospatial.org/files/?artifact_id=25562`.

[32] BRACHMAN, R., AND LEVESQUE, H. *Knowledge Representation and Reasoning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[33] BRATMAN, M. E. *Intentions, Plans, and Practical Reason.* Harvard University Press, 1987.

[34] BRAUBACH, L., POKAHR, A., BADE, D., KREMPELS, K.-H., AND LAMERSDORF, W. *5th International Workshop on Engineering Societies in the Agents World, LNAI 3451.* Springer-Verlag Berlin Heidelberg, 2005, ch. Deployment of Distributed Multi-agent Systems, pp. 261–276.

[35] BRICKLEY, D., AND GUHA, R. V. RDF vocabulary description language 1.0: RDF schema. Tech. rep., W3C, February 2004. Available from: `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/` [cited 24 June 2007].

[36] BROOKS, R. *Architectures for Intelligence*. Lawrence Erlbaum Assosiates, Hillsdale, NJ, 1991, ch. How to build complete creatures rather than isolated cognitive simulators, pp. 225–239.

[37] BRÖRING, A., FÖRSTER, T., AND SIMONIS, I. Sensor web enablement: The 52 North SWE suite. In *FOSS4G2006 - Free And Open Source Software for Geoinformatics,11-15 September 2006, Lausanne, Switzerland* (2006). Available from: `http://www.foss4g2006.org/contributionDisplay.py?contribId=134&sessionId=37&confId=1` [cited 3 March 2007].

[38] BURSTEIN, M., BUSSLER, C., ZAREMBA, M., FININ, T., HUHNS, M. N., PAOLUCCI, M., SHETH, A. P., AND WILLIAMS, S. A semantic web services architecture. *IEEE Internet Computing 9*, 5 (2005), 72–81.

[39] BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P., AND STAL, M. *Pattern-oriented software architecture: a system of patterns*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[40] C. MATHEUS, M. KOKAR, K. B., AND LETKOWSKI, J. Using SWRL and OWL to capture domain knowledge for a situation awareness application applied to a supply logistics scenario. In *RuleML-2005: International Conference on Rules and Rule Markup Languages for the Semantic Web, Galway, Ireland, November, 2005* (2005).

[41] CAMPBELL, J., GIBBONS, P. B., NATH, S., PILLAI, P., SESHAN, S., AND SUKTHANKAR, R. Irisnet: an internet-scale architecture for multimedia sensors. In *MULTIMEDIA '05: Proc. 13th annual ACM international conference on Multimedia* (New York, NY, USA, 2005), ACM Press, pp. 81–88.

[42] CARDOSO, J. *Semantic Web Services, Processes and Applications*. In Cardoso and Sheth [44], 2006, ch. Programming the Semantic Web, pp. 351–380.

[43] CARDOSO, J. The semantic web vision: Where are we? *IEEE Intelligent Systems 22*, 5 (2007), 84–88.

[44] CARDOSO, J., AND SHETH, A., Eds. *Semantic Web Services, Processes and Applications*. Springer Science+Business Media, LLC, New York, USA, 2006.

[45] CHEN, H., FININ, T., AND JOSHI, A. *Ontologies for Agents: Theory and Experiences*. Birkhuser Basel, 2005, ch. The SOUPA Ontology for Pervasive Computing, pp. 233–258.

[46] CHEN, H. L. *An Intelligent Broker Architecture For Pervasive Context-Aware Systems*. PhD thesis, University of Maryland, Baltimore County, 2004.

[47] CHIEN, S., CICHY, B., DAVIES, A., TRAN, D., RABIDEAU, G., CASTANO, R., SHERWOOD, R., MANDL, D., FRYE, S., SHULMAN, S., JONES, J., AND GROSVENOR, S. An autonomous earth-observing sensor web. *IEEE Intelligent Systems 20*, 3 (2005), 16–24.

[48] CHOI, N., SONG, I.-Y., AND HAN, H. A survey on ontology mapping. *SIGMOD Rec. 35*, 3 (2006), 34–41.

[49] CLEMENTINI, E., FELICE, P. D., AND VAN OOSTEROM, P. A small set of formal topological relationships suitable for end-user interaction. In *SSD '93: Proc. 3rd International Symposium on Advances in Spatial Databases* (London, UK, 1993), Springer-Verlag, pp. 277–295.

[50] COX, S. Observations and measurements. discussion paper. OGC 05-087r3 version 0.13.0 observations and measurements. 2006. Available from: `http://www.opengeospatial.org/standards/requests/37`.

[51] CRANEFIELD, S., PURVIS, M., NOWOSTAWSKI, M., AND HWANG, P. Ontologies for interaction protocols. In *Proc. Workshop on Ontologies in Agent Systems, Bologna, Italy, July 2002* (2002).

[52] DAVIES, J., FENSEL, D., AND VAN HARMELEN, F., Eds. *Towards the Semantic Web: Ontology-driven Knowledge Management*. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[53] DAVIS, M. Secrets of the JTS topology suite. In *Free and Open Source Software for Geospatial 2007, FOSS4G2007, Victoria, Canada* (24-27 September 2007). Available from: `http://www.foss4g2007.org/`.

[54] DECKER, K., SYCARA, K., AND WILLIAMSON, M. Middle agents for the Internet. In *Proc. 15th Int. Joint Conference on Artificial Intelligence Nagoya, Japan* (August 1997), pp. 578–584.

[55] DELIN, K., AND JACKSON, S. The sensor web: a new instrument concept. In *Proc. SPIE Symposium on Integrated Optics, 20-26 Jan. 2001, San Jose, CA* (2001), vol. 4284, pp. 1–9.

[56] DELIN, K. A. The sensor web: Distributed sensing for collective action. *Sensors Online* (July 2006).

[57] DING, Z. *BayesOWL: A Probabilistic Framework for Semantic Web*. PhD thesis, University of Maryland, Baltimore County, December 2005.

[58] DING, Z., PENG, Y., AND PAN, R. *BayesOWL: Uncertainty Modeling in Semantic Web Ontologies*. Studies in Fuzziness and Soft Computing. Springer-Verlag, October 2006, pp. 3–29.

[59] DRUMMOND, N., RECTOR, A., STEVENS, R., MOULTON, G., HORRIDGE, M., WANG, H. H., AND SEIDENBERG, J. Putting OWL in order: Patterns for sequences in OWL. In *OWLED 2006: Proc. of the Workshop on OWL: Experiences and Directions, Athens, Georgia (USA) November 10-11, 2006* (2006), B. C. Grau, P. Hitzler, C. Shankey, and E. Wallace, Eds.

[60] EGENHOFER, M. J. Toward the semantic geospatial web. In *GIS '02: Proc. 10th ACM international symposium on Advances in geographic information systems* (New York, NY, USA, 2002), ACM, pp. 1–4.

[61] EGENHOFER, M. J., AND HERRING, J. Categorizing binary topological relationships between regions, lines and points in geographic databases. Tech. rep., Department of Surveying Engineering, University of Maine, Orono, ME, 1991.

[62] ELENIUS, D., DENKER, G., MARTIN, D., GILHAM, F., KHOURI, J., SADAATI, S., AND SENANAYAKE, R. *The Semantic Web: Research and Applications, Proc. 2nd European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29-June 1, 2005*. Springer Berlin/Heidelberg, 2005, ch. The OWL-S Editor A Development Tool for Semantic Web Services, pp. 78–92.

[63] FLEMING, G., VAN DEN BERGH, F., CLAUDEL, F., AND FROST, P. Sensor web enabling the advanced fire information system. In *ISESS 2005: Proc. 2005 International Symposium for Environmental Software Systems, Hotel do Mar, Sesimbra, Portugal* (2005). Available from: `http://www.isess.org/documents/2005/presentations/`.

[64] FONSECA, F., EGENHOFER, M., AGOURIS, P., AND CAMARA, G. Using ontologies for integrated geographic information systems.

[65] FOUNDATION FOR INTELLIGENT AND PHYSICAL AGENTS. Fipa ontology service specification: Xc00086d, 2001. Available from: `http://www.fipa.org/` [cited 18 October 2006].

[66] FOUNDATION FOR INTELLIGENT AND PHYSICAL AGENTS. Fipa abstract architecture specification: Sc000011, 2002. Available from: `http://www.fipa.org/` [cited 18 October 2006].

[67] FOUNDATION FOR INTELLIGENT AND PHYSICAL AGENTS. Fipa agent configuration management specification: Fipa00090, 2002. Available from: `http://www.fipa.org/` [cited 18 October 2006].

[68] FOUNDATION FOR INTELLIGENT AND PHYSICAL AGENTS. Fipa communicative act library specification, 03 2002. Available from: `http://www.fipa.org/` [cited 18 October 2006].

[69] FOUNDATION FOR INTELLIGENT AND PHYSICAL AGENTS. Fipa agent management specification: Fipa00023, 2004. Available from: `http://www.fipa.org/` [cited 18 October 2006].

[70] FRANCONI, E., AND TESSARIS, S. Rules and queries with ontologies: a unified logical framework. In *Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR-04), 2004.* (2004), E. Franconi and S. Tessaris, Eds.

[71] FROST, P., AND VOSLOO, H. Providing satellite-based early warnings of fires to reduce fire flashovers on transmission lines. *ESI Africa 2* (2006), 48–51.

[72] GANGEMI, A., GUARINO, N., MASOLO, C., AND OLTRAMARI, A. Sweetening WORDNET with DOLCE. *AI Mag. 24*, 3 (2003), 13–24.

[73] GASPARI, M. Concurrency and knowledge-level communication in agent languages. *Artif. Intell. 105*, 1-2 (1998), 1–45.

[74] GASSER, L. *MAS Infrastructure, Definitions, Needs, and Prospects.* Springer-Verlag, 2001.

[75] GENESERETH, M. R., AND KETCHPEL, S. P. Software agents. *Commun. ACM 37*, 7 (1994).

[76] GEROIMENKO, V., AND CHEN, C. *Visualizing the Semantic Web: XML-based Internet and Information Visualization.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[77] GIBBONS, P. B., KARP, B., KE, Y., NATH, S., AND SESHAN, S. IrisNet: An architecture for a worldwide sensor web. *IEEE Pervasive Computing 02*, 4 (2003), 22–33.

[78] GINSBERG, A., POSTON, J. D., AND HORNE, W. D. Experiments in cognitive radio and dynamic spectrum access using an ontology-rule hybrid architecture. In *RuleML-2006, Second International Conference. Athens, Georgia, USA, 10-11 November 2006* (2006).

[79] GOLBREICH, C., AND WALLACE, E. K. OWL 2 web ontology language new features and rationale, 22 September 2009. Available from: `http://www.w3.org/TR/owl2-new-features/` [cited 30 September 2009].

[80] GOMEZ-PEREZ, A., CORCHO-GARCIA, O., AND FERNANDEZ-LOPEZ, M. *Ontological Engineering.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[81] GREAVES, M., HOLMBACK, H., AND BRADSHAW, J. What is a conversation policy? In *Issues in Agent Communication* (London, UK, 2000), Springer-Verlag, pp. 118–131.

[82] GROSOF, B. N., HORROCKS, I., VOLZ, R., AND DECKER, S. Description logic programs: combining logic programs with description logic. In *WWW '03: Proc. 12th international conference on the World Wide Web* (New York, NY, USA, 2003), ACM, pp. 48–57.

[83] GROUP ON EARTH OBSERVATIONS. *10-Year Implementation Plan Reference Document*. ESA Publication Division, Noordwijk, The Netherlands, February 2005. Available from: http://www.earthobservations.org.

[84] GRUBER, T. R. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies 43*, 5,6 (1995), 907–928.

[85] GRUNINGER, M., AND MCILRAITH, S. Specifying a web service ontology in first-order logic. In *2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering March 27 – 29, 2006* (2006).

[86] GUARINO, N. *Formal Ontology in Information Systems*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 1998.

[87] HAARSLEV, V., AND MÖLLER, R. Racer: A core inference engine for the semantic web. In *Proc. 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), located at the 2nd International Semantic Web Conference ISWC 2003, Sanibel Island, Florida, USA, October 20* (2003), pp. 27–36.

[88] HAKIMPOUR, F., ALEMAN-MEZA, B., PERRY, M., AND SHETH, A. Data processing in space, time and semantics dimensions. In *Terra Cognita 2006 - Directions to the Geospatial Semantic Web; An International Semantic Web Conference (ISWC) 2006 Workshop* (2006).

[89] HENDLER, J. Is there an intelligent agent in your future?, March 1999. Available from: http://www.nature.com/nature/webmatters/agents/agents.html [cited 31 Oct. 2006].

[90] HENDLER, J. Where are all the intelligent agents? *IEEE Intelligent Systems 22*, 3 (2007), 2–3.

[91] HENDLER, J., BERNERS-LEE, T., AND MILLER, E. Integrating applications on the semantic web. *Journal of the Institute of Electrical Engineers of Japan 122*, 10 (October, 2002), 676–680.

[92] HEPP, M. Semantic web and semantic web services: father and son or indivisible twins? *IEEE Internet Computing 10*, 2 (March-April 2006), 85–88.

[93] HEWITT, C., AND DE JONG, P. Open systems. In *On Conceptual Modelling (Intervale)* (1982), pp. 147–164.

[94] HOBBS, J. R. A DAML ontology of time, 2002. Available from: http://www.cs.rochester.edu/~ferguson/daml/daml-time-nov2002.txt [cited 11 November 2007].

[95] HOBBS, J. R., AND PAN, F. An ontology of time for the semantic web. *ACM Transactions on Asian Language Information Processing (TALIP) 3*, 1 (2004), 66–85.

[96] HOBBS, J. R., AND PAN, F. Time ontology in OWL, W3C editor's draft 6 september 2006, 09 2006. Available from: http://www.w3.org/2001/sw/BestPractices/OEP/Time-Ontology-20060906.

[97] HORROCKS, I., PATEL-SCHNEIDER, P. F., BOLEY, H., TABET, S., GROSOF, B., AND DEAN, M. SWRL: A semantic web rule language combining OWL and RuleML, 21 May 2004. Available from: http://www.w3.org/Submission/SWRL/ [cited 31 December 2007].

[98] HORROCKS, I., PATEL-SCHNEIDER, P. F., BOLEY, H., TABET, S., GROSOF, B., AND DEAN, M. SWRL: A semantic web rule language, 21 May 2004. Available from: http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/ [cited 25 February 2007].

[99] HUHNS, M. N. A research agenda for agent-based service-oriented architectures. In *Cooperative Intelligent Agents* (2006), pp. 8–22.

[100] HUHNS, M. N., SINGH, M. P., BURSTEIN, M., DECKER, K., DURFEE, E., FININ, T., GASSER, L., GORADIA, H., JENNINGS, N., LAKKARAJU, K., NAKASHIMA, H., PARUNAK, H. V. D., ROSENSCHEIN, J. S., RUVINSKY, A., SUKTHANKAR, G., SWARUP, S., SYCARA, K., TAMBE, M., WAGNER, T., AND ZAVALA, L. Research directions for service-oriented multiagent systems. *IEEE Internet Computing 9*, 6 (November 2005), 65–70.

[101] JENNINGS, N. R. An agent-based approach for building complex software systems. *Commun. ACM 44*, 4 (2001), 35–41.

[102] JENNINGS, N. R., SYCARA, K., AND WOOLDRIDGE, M. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems 1*, 1 (1998), 7–38.

[103] JIANG, G., CHUNG, W., AND CYBENKO, G. Semantic agent technologies for tactical sensor networks. In *Proc. 5th SPIE Conference on Unattended Ground Sensor Technologies and Applications* (Orlando, FL, September 2003), E. M. Carapezza, Ed., vol. 5090, SPIE, pp. 311–320.

[104] KALFOGLOU, Y., AND SCHORLEMMER, M. Ontology mapping: the state of the art. *Knowl. Eng. Rev. 18*, 1 (2003), 1–31.

[105] KALYANPUR, A., PARSIA, B., SIRIN, E., GRAUA, B. C., AND HENDLER, J. SWOOP: A web ontology editing browser. *Web Semantics: Science, Services and Agents on the World Wide Web 4*, 2 (June 2006), 144–153.

[106] KASHYAP, V., AND SHETH, A. *Semantic Heterogeneity in Global Information Systems: The Role of Metadata, Context and Ontologies*. Academic Press, San Diego, 1997, pp. 139–178.

[107] KELLER, U., LARA, R., LAUSEN, H., AND FENSEL, D. *Semantic Web Services: Theory, Tools and Applications*. IGI Global, 2007, ch. Semantic Web Service Discovery in the WSMO Framework, pp. 281–316.

[108] KLUSCH, M. *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[109] KLUSCH, M. Information agent technology for the Internet: a survey. *Data Knowl. Eng. 36*, 3 (2001), 337–372.

[110] KLUSCH, M., BERGAMASCHI, S., AND PETTA, P. *European Research and Development of Intelligent Information Agents: The AgentLink Perspective.* 2003, pp. 1–21.

[111] KLUSCH, M., ROVATSOS, M., AND PAYNE, T. R., Eds. *Cooperative Information Agents X, 10th International Workshop, CIA 2006, Edinburgh, UK, September 11-13, 2006, Proceedings* (2006), vol. 4149 of *Lecture Notes in Computer Science*, Springer.

[112] KLYNE, G., AND CARROLL, J. Resource description framework (RDF): Concepts and abstract syntax, February 2004. Available from: `http://www.w3.org/TR/rdf-concepts/` [cited 28 May 2006].

[113] KOLAS, D., HEBELER, J., AND DEAN, M. Geospatial semantic web: Architecture of ontologies. In *GeoS 2005, LNCS 3799* (2005), M. R. et al., Ed., Springer-Verlag Berlin Heidelberg, p. 183 194.

[114] KONE, M. T., SHIMAZU, A., AND NAKAJIMA, T. The state of the art in agent communication languages. *Knowledge and Information Systems 2*, 3 (2000), 259–284.

[115] Kozlenkov, A., Penaloza, R., Nigam, V., Royer, L., Dawelbait, G., and Schroeder, M. PROVA: Rule-based Java scripting for distributed web applications: A case study in bioinformatics. In *Reactivity on the Web Workshop, Munich 2006* (2006).

[116] Kublauch, H. Ontology driven software development in the context of the semantic web: An example scenario with ProtegeOWL, 2005.

[117] Kuhn, W. Geospatial semantics: Why, of what, and how? In *J. Data Semantics III* (2005), S. Spaccapietra and E. Zimányi, Eds., vol. 3534 of *Lecture Notes in Computer Science*, Springer, pp. 1–24.

[118] Labrou, Y., Finin, T., and Peng, Y. Agent communication languages: The current landscape. *IEEE Intelligent Systems 14*, 2 (1999), 45–52.

[119] Laclavik, M., Balogh, Z., Babik, M., and Hluchý, L. AgentOWL: Semantic knowledge model and agent architecture. *Computers and Artificial Intelligence 25*, 5 (2006).

[120] Langley, B. K., Paolucci, M., and Sycara, K. Discovery of infrastructure in multi-agent systems. In *AAMAS '03: Proc. 2nd international joint conference on Autonomous agents and multiagent systems* (New York, NY, USA, 2003), ACM Press, pp. 1046–1047.

[121] Lemmens, R., de By, R., Gould, M., Wytzisk, A., Granell, C., and van Oosterom, P. Enhancing geo-service chaining through deep service descriptions. *Transactions in GIS 11*, 6 (March 2007), 849–871.

[122] Li, K., Verma, K., Mulye, R., Rabbani, R., Miller, J., and Sheth, A. *Semantic Web Services, Processes and Applications*. In Cardoso and Sheth [44], 2006, ch. Designing Semantic Web Processes: The WSDL-S Approach.

[123] Liang, S. H. L., Croitoru, A., and Tao, C. V. A distributed geospatial infrastructure for sensor web. *Computers & Geosciences 31*, 2 (2005), 221–231.

[124] Lieberman, J. Geospatial semantic web interoperability experiment report, version 0.5. Tech. rep., Open Geospatial Consortium, 2006.

[125] Liu, J., and Zhao, F. Towards semantic services for sensor-rich information systems. In *The 2nd IEEE/CreateNet International Workshop on Broadband Advanced Sensor Networks (Basenets 2005), Boston, MA* (Oct. 3, 2005).

[126] Luck, M., McBurney, P., Shehory, O., and Willmott, S. Agent technology roadmap: A roadmap for agent based computing, 2005. Available from: `http://www.agentlink.org/roadmap/index.html` [cited 28 May 2006].

[127] Lutz, M., and Klein, E. Ontology-based retrieval of geographic information. *Journal of Geographical Information Science 20*, 3 (2006), 233–260.

[128] Lutz, M., and Kolas, D. Rule-based discovery in spatial data infrastructure. *Transactions in GIS 11*, 3 (2007), 317–336.

[129] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K. OWL-S: Semantic markup for web services, 22 November 2004. Available from: `http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/`.

[130] Martin, D., Burstein, M., Mcdermott, D., Mcilraith, S., Paolucci, M., Sycara, K., Mcguinness, D. L., Sirin, E., and Srinivasan, N. Bringing semantics to web services with OWL-S. *World Wide Web 10*, 3 (2007), 243–277.

[131] McGuinness, D. L., and van Harmelen, F. OWL: Web ontology language overview, February 2004. Available from: `http://www.w3.org/TR/2004/REC-owl-features-20040210/` [cited 24 June 2007].

[132] Mennis, J., Peuquet, D., and Qian, L. A conceptual framework for incorporating cognitive principles into geographical database representation. *International Journal of Geographical Information Science 1414*, 6 (2000), 501–520.

[133] Michener, W. K., Beach, J. H., Jones, M. B., Ludäscher, B., Pennington, D. D., Pereira, R. S., Rajasekar, A., and Schildhauer, M. A knowledge environment for the biodiversity and ecological sciences. *J. Intell. Inf. Syst. 29*, 1 (2007), 111–126.

[134] Moodley, D. The future of the Internet: The semantic web, web services and a multi-agent system infrastructure for the Internet. In *Proc. South African Computer Lecturers Association 2004, 4-6 July Durban, 2004* (2004).

[135] Moodley, D., and Kinyua, J. A multi-agent system for electronic job markets. In *Proc. 6th International conference on Business Information Systems, Colorado Springs, USA, 4-6 June 2003, published by Dept. of Management Info. Systems, The Poznan University of Economics, Poznan* (2003), pp. 42–48.

[136] Moodley, D., and Kinyua, J. D. M. Towards a multi-agent infrastructure for distributed Internet applications. In *8th Annual Conference on WWW Applications, Bloemfontein, South Africa, 5-6 September* (2006).

[137] Moodley, D., and Simonis, I. A new architecture for the sensor web: the SWAP-framework. In *Semantic Sensor Networks Workshop, A workshop of the 5th International Semantic Web Conference ISWC 2006, November 5-9, Athens, Georgia, USA* (2006).

[138] Moodley, D., Terhorst, A., Simonis, I., McFerren, G., and van den Bergh, F. Using the sensor web to detect and monitor the spread of wild fires. In *Second International Symposium on Geo-information for Disaster Management (Gi4DM) September 25-26, Pre-Conference Symposium to ISPRS TC-IV and ISRS Symposium on Geospatial Databases for Sustainable Development September 27-30, at Goa, India* (2006).

[139] Moodley, D., Vahed, A., Simonis, I., McFerren, G., and Zyl, T. V. Enabling a new era of earth observation research: scientific workflows for the sensor web. *Ecological Circuits 1* (2008), 20–23.

[140] Motik, B. On the properties of metamodeling in OWL. In *Proc. of ISWC 2005* (2005).

[141] Müller, J. P., and Pischel, M. The agent architecture InteRRaP: Concept and application. Tech. rep., German Research Center for Artificial Intelligence, 1993. RR 93-26. Available from: `http://www.dfki.uni-sb.de/mas/interrap/interrap.v3.ps`.

[142] Negri, A., Poggi, A., Tomaiuolo, M., and Turci, P. Agents for e-business applications. In *AAMAS '06: Proc. 5th international joint conference on Autonomous agents and multiagent systems* (New York, NY, USA, 2006), ACM Press, pp. 907–914.

[143] Nolan, J. *An agent-based architecture for distributed imagery and geospatial computing*. PhD thesis, George Mason University, 2003.

[144] Noy, N., and Rector, A. Defining n-ary relations on the semantic web: Use with individuals, W3C working group note, 12 April 2006. Available from: `http://www.w3.org/TR/swbp-n-aryRelations/`.

[145] NOY, N. F. Semantic integration: a survey of ontology-based approaches. *SIGMOD Rec. 33*, 4 (2004), 65–70.

[146] NWANA, H. S., AND NDUMU, D. T. A perspective on software agents research. *Knowledge Engineering Revue 14*, 2 (1999), 125–142.

[147] OBERLE, D. *Semantic Management of Middleware*. Springer, New York, 2006.

[148] OGSTON, E., AND VASSILIADIS, S. Matchmaking among minimal agents without a facilitator. In *AGENTS '01: Proc. 5th international conference on Autonomous agents* (New York, NY, USA, 2001), ACM Press, pp. 608–615.

[149] OMICINI, A., OSSOWSKI, S., AND RICCI, A. *Coordination Infrastructures in the Engineering of Multiagent Systems*, vol. 11 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Kluwer Academic Publishers, June 2004.

[150] OPEN GIS CONSORTIUM, INC. OpenGIS simple features specification for SQL revision 1.1. Tech. rep., OpenGIS Project Document 99-049, May 5, 1999.

[151] OUKSEL, A. M. *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*. Springer, Berlin, Germany, 1999, ch. A framework for a scalable agent architecture of cooperating heterogeneous knowledge sources, pp. 100–124.

[152] OUKSEL, A. M., AND SHETH, A. Semantic interoperability in global information systems. *SIGMOD Rec. 28*, 1 (1999), 5–12.

[153] OZDEMIR, S., AND XIAO, Y. Secure data aggregation in wireless sensor networks: A comprehensive overview. *Computer Networks 53*, 12 (2009), 2022 – 2037.

[154] PAN, J. Z. A flexible ontology reasoning architecture for the semantic web. *IEEE Transactions on Knowledge and Data Engineering 19*, 2 (2007), 246–260.

[155] PAN, J. Z., AND HORROCKS, I. OWL-EU: Adding customised datatypes into OWL. *Web Semantics: Science, Services and Agents on the World Wide Web 4*, 1 (January 2006), 29–39.

[156] PARBHOO, C. An ontology driven sensor web application for detecting and classifying informal settlements. Master's thesis, School of Computer Science, University of KwaZulu-Natal, 2009.

[157] PARUNAK, H. V. D. *Multiagent systems: a modern approach to distributed artificial intelligence*. In [198], 1999, ch. Industrial and practical applications of DAI, pp. 377–421.

[158] PAYNE, T., SINGH, R., AND SYCARA, K. Communicating agents in open multi-agent systems. In *First GSFC/JPL Workshop on Radical Agent Concepts (WRAC)* (2002).

[159] PEARL, J., AND RUSSEL, S. *Handbook of Brain Theory and Neural Networks*. MIT Press, Cambridge, MA, 2003, ch. Bayesian networks, pp. 157–160.

[160] PECHOUCEK, M., THOMPSON, S. G., BAXTER, J. W., HORN, G. S., KOK, K., WARMER, C., KAMPHUIS, R., MARIK, V., VRBA, P., HALL, K. H., MATURANA, F. P., DORER, K., AND CALISTI, M. Agents in industry: The best from the AAMAS 2005 industry track. *IEEE Intelligent Systems 21*, 2 (2006), 86–95.

[161] PERRY, M., HAKIMPOUR, F., AND SHETH, A. Analyzing theme, space, and time: an ontology-based approach. In *GIS '06: Proc. 14th annual ACM international symposium on Advances in geographic information systems* (New York, NY, USA, 2006), ACM Press, pp. 147–154.

[162] PETRIE, C. J. Agent-based engineering, the web, and intelligence. *IEEE Expert: Intelligent Systems and Their Applications 11*, 6 (1996), 24–29.

[163] PHIL TETLOW, I., OBERLE, J. Z. P. D., WALLACE, E., USCHOLD, M., AND KENDALL, E. Ontology driven architectures and potential uses of the semantic web in systems and software engineering, 2006. Available from: `http://www.w3.org/2001/sw/BestPractices/SE/ODA/060211/` [cited 3 October 2009].

[164] PRUD'HOMMEAUX, E., AND SEABORNE, A. SPARQL query language for RDF, 14 June 2007. Available from: `http://www.w3.org/TR/rdf-sparql-query/` [cited 3 November 2007].

[165] RANDELL, D., AND COHN, A. A spatial logic based on regions and connection. In *B. Nebel, W. Swartout, C. Rich (Eds.), Proc. 3rd international Conference on the Principles of Knowledge Representation and Reasoning, Morgan Kaufmann, Los Altos, CA* (1992), pp. 165–176.

[166] RAO, A. S., AND GEORGEFF, M. P. Modeling rational agents within a BDI-architecture. In *Proc. 2nd international Conference on Principles of Knowledge Representation and Reasoning* (1991), J. Allen, R. Fikes, and E. Sandewall, Eds., Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, pp. 473–484.

[167] RASKIN, R. Guide to SWEET ontologies. Available from: `http://sweet.jpl.nasa.gov/guide.doc` [cited 16 February 2007].

[168] RASKIN, R. G., AND PAN, M. J. Knowledge representation in the semantic web for earth and environmental terminology (SWEET). *Computers & Geosciences 31*, 9 (November 2005), 1119–1125.

[169] REED, S., AND LENAT, D. Mapping ontologies into CYC. In *AAAI 2002 Conference Workshop on Ontologies For The Semantic Web, Edmonton, Canada* (July 2002).

[170] REITSMA, F., AND ALBRECHT, J. Modeling with the semantic web in the geosciences. *IEEE Intelligent Systems 20*, 2 (2005), 86–88.

[171] RUSSELL, S., AND NORVIG, P. *Artificial Intelligence: A Modern Approach*, 2nd edition ed. Prentice-Hall, Englewood Cliffs, NJ, 2003.

[172] RUSSOMANNO, D. J., KOTHARI, C. R., AND THOMAS, O. A. Building a sensor ontology: A practical approach leveraging ISO and OGC models. In *IC-AI* (2005), H. R. Arabnia and R. Joshua, Eds., CSREA Press, pp. 637–643.

[173] SCHARL, A., AND TOCHTERMANN, K., Eds. *The Geospatial Web*. Springer, 2007.

[174] SCHREINER, K. HPKBs and beyond. *IEEE Intelligent Systems and Their Applications 14*, 2 (Mar/Apr 1999), 80–84.

[175] SEARLE, J. R. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University PressCambridge, 1969.

[176] SENGUPTA, R., AND SIEBER, R. Geospatial agents, agents everywhere . . . *Transactions in GIS 11*, 4 (2007), 483–506.

[177] SHADBOLT, N., BERNERS-LEE, T., AND HALL, W. The semantic web revisited. *IEEE Intelligent Systems 21*, 3 (2006), 96–101.

[178] SHAMSFARD, M., AND BARFOROUSH, A. A. The state of the art in ontology learning: a framework for comparison. *Knowl. Eng. Rev. 18*, 4 (2003), 293–316.

[179] SHETH, A. *Interoperating Geographic Information Systems*. Kluwer Academic Publishers, 1998, ch. Changing Focus on Interoperability in Information Systems: From System, Syntax, Structure to Semantics, pp. 5–30.

[180] SINGH, M. P., CHOPRA, A. K., DESAI, N., AND MALLYA, A. U. Protocols for processes: programming in the large for open systems. vol. 39, ACM Press, pp. 73–83.

[181] SINGH, M. P., AND HUHNS, M. N. *Service-oriented computing: Semantics, Processes, Agents*. John Wiley & Sons Ltd, Chichester, England, 2005.

[182] SIRIN, E., PARSIA, B., GRAU, B. C., KALYANPUR, A., AND KATZ, Y. Pellet: A practical OWL-DL reasoner. *Web Semant. 5*, 2 (2007), 51–53.

[183] SMITH, B., ASHBURNER, M., ROSSE, C., BARD, J., BUG, W., CEUSTERS, W., GOLDBERG, L. J., EILBECK, K., IRELAND, A., MUNGALL, C. J., CONSORTIUM, T. O., LEONTIS, N., ROCCA-SERRA, P., RUTTENBERG, A., SANSONE, S.-A., SCHEUERMANN, R. H., SHAH, N., WHETZEL, P. L., AND LEWIS, S. The OBO foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology 25* (2007), 1251–1255.

[184] SUJANANI, A., RAY, P., PARAMESH, N., AND BHAR, R. The development of ontology driven multi-agent systems: a case study in the financial services domain. In *BSN '05: Proc. IEEE EEE05 international workshop on Business services networks* (Piscataway, NJ, USA, 2005), IEEE Press.

[185] SUN DEVELOPER NETWORK. J2EE CORBA technology and the Java platform. Available from: `http://java.sun.com/j2ee/corba/index.html` [cited 13 November 2007].

[186] SYCARA, K. Multiagent systems. *AI Magazine 10*, 2 (1998), 79–93.

[187] SYCARA, K. *Multi-agent infrastructure, agent discovery, middle agents for Web services and interoperation*. Springer-Verlag New York, Inc., New York, NY, USA, 2001, pp. 17–49.

[188] SYCARA, K., PAOLUCCI, M., VELSEN, M. V., AND GIAMPAPA, J. A. The RETSINA MAS infrastructure. *Autonomous Agents and Multi-Agent Systems 7*, 1/2 (July 2003), 29–48. also appears as CMU-RI-TR-01-05.

[189] TERHORST, A., MOODLEY, D., SIMONIS, I., FROST, P., MCFERREN, G., ROOS, S., AND VAN DEN BERGH, F. *Geosensor Networks, Lecture Notes in Computer Science, Volume 4540/2008*. Springer-Verlag, 2008, ch. Using the Sensor Web to Detect and Monitor the Spread of Vegetation Fires in Southern Africa, pp. 239–251.

[190] TERHORST, A., SIMONIS, I., AND MOODLEY, D. A service-oriented multi-agent systems architecture for the sensor web. In *SAEON Summit, Centurion, South Africa* (2006).

[191] THE OWL-S COALITION. OWL-S 1.1 release. Tech. rep., Novmber 2004. Available from: `http://www.daml.org/services/owl-s/1.1/` [cited 25 February 2006].

[192] TWEEDALE, J., ICHALKARANJE, N., SIOUTIS, C., JARVIS, B., CONSOLI, A., AND PHILLIPS-WREN, G. Innovations in multi-agent systems. *J. Netw. Comput. Appl. 30*, 3 (2007), 1089–1115.

[193] USCHOLD, M. Ontology-driven information systems: Past, present and future. In *FOIS* (2008), pp. 3–18.

[194] USCHOLD, M., AND GRUNINGER, M. Ontologies and semantics for seamless connectivity. *SIGMOD Rec. 33*, 4 (2004), 58–64.

[195] VALCKENAERS, P., SAUTER, J., SIERRA, C., AND RODRIGUEZ-AGUILAR, J. A. Applications and environments for multi-agent systems. *Autonomous Agents and Multi-Agent Systems* (2006).

[196] VO, Q. B., AND PADGHAM, L. Conversation-based specification and composition of agent services. In *ciax06* (2006), pp. 168–182.

[197] WALTON, C. *Agency and the Semantic Web.* Oxford University Press, Inc., New York, NY, USA, 2006.

[198] WEISS, G. *Multiagent systems: a modern approach to distributed artificial intelligence.* MIT Press, 1999.

[199] WELTY, C. Guest editorial: ontology research. *AI Mag. 24*, 3 (2003), 11–12.

[200] WEYNS, D., AND HOLVOET, T. On the role of environments in multiagent systems. *Informatica 29*, 4 (2005), 409–422.

[201] WHITEHOUSE, K., ZHAO, F., AND LIU, J. Semantic streams: A framework for composable semantic interpretation of sensor data. K. Rmer, H. Karl, and F. Mattern, Eds., Springer, pp. 5–20.

[202] WICKLER, G., AND TATE, A. Capability representations for brokering: A survey. 1999.

[203] WIJNGAARDS, N. J. E., OVEREINDER, B. J., VAN STEEN, M., AND BRAZIER, F. M. T. Supporting internet-scale multi-agent systems. *Data Knowl. Eng. 41*, 2-3 (2002), 229–245.

[204] WITT, K. J., STANLEY, J., SMITHBAUER, D., MANDL, D., LY, V., DERBRINK, A. U., AND METHENY, M. Enabling sensor webs by utilizing SWAMOfor autonomous operations. In *8th NASA Earth Science Technology Conference* (2008).

[205] WOOLDRIDGE, M. *An Introduction to Multiagent Systems.* John Wiley & Sons (Chichester, England), 2002.

[206] WOOLDRIDGE, M., AND JENNINGS, N. R. Agent theories, architectures, and languages: a survey. In *ECAI-94: Proc. workshop on agent theories, architectures, and languages on Intelligent agents* (New York, NY, USA, 1995), Springer-Verlag New York, Inc., pp. 1–39.

[207] YANG, Y., AND CALMET, J. OntoBayes: An ontology-driven uncertainty model. In *CIMCA '05: Proc. International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-1 (CIMCA-IAWTIC'06)* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 457–463.

[208] YANG, Y., AND CALMET, J. *Foundations of Intelligent Systems, LNCS 4203.* Springer, 2006, ch. OntoBayes Approach to Corporate Knowledge, pp. 274–283.

[209] YICK, J., MUKHERJEE, B., AND GHOSAL, D. Wireless sensor network survey. *Computer Networks 52*, 12 (2008), 2292 – 2330.

[210] YUE, P., DI, L., YANG, W., YU, G., AND ZHAO, P. Semantics-based automatic composition of geospatial web service chains. *Comput. Geosci. 33*, 5 (2007), 649–665.

[211] ZAHARIA, R., VASILIU, L., HOFFMAN, J., AND KLIEN, E. Semantic execution meets geospatial web services: A pilot application. *Transactions in GIS 12*, Suppl. 1 (2008), 59–73.

[212] ZAMBONELLI, F., JENNING, N. R., OMICINI, A., AND WOOLDRIDGE, M. J. Agent-oriented software engineering for Internet agents. 326–346.

[213] ZAMBONNELLI, F., J. N. R. W. M. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology 12*, 3 (2003), 317–370.

[214] ZAREMBA, M., KERRIGAN, M., MOCAN, A., AND MORAN, M. *Semantic Web Services, Processes and Applications*. In Cardoso and Sheth [44], 2006, ch. Web Services Modeling Ontology, pp. 63–87.

[215] ZHOU, L. Ontology learning: state of the art and open issues. *Inf. Tech. and Management 8*, 3 (2007), 241–252.

[216] ZOU, Y., FININ, T., DING, L., CHEN, H., AND PAN, R. Using semantic web technology in multi-agent systems: a case study in the TAGA trading agent environment. In *ICEC '03: Proc. 5th international conference on Electronic commerce* (New York, NY, USA, 2003), ACM, pp. 95–101.